ACCELERATING PHYSICS SCHEMES IN NUMERICAL WEATHER PREDICTION CODES AND PRESERVING POSITIVITY IN THE PHYSICS-DYNAMICS COUPLING

by

Timbwaoga Aime Judicael Ouermi

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

August 2022

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Timbwaoga Aime Judicael Ouermi**
has been approved by the following supervisory committee members:

Martin Berzins,	Chair(s)	________ Date Approved
Robert M. Kirby,	Member	________ Date Approved
Hari Sundar,	Member	________ Date Approved
Zhaoxia Pu,	Member	________ Date Approved
Alex Reinecke,	Member	________ Date Approved

by **Mary Hall**, Chair/Dean of
the Department/College/School of **Computing**
and by **David B. Kieda**, Dean of The Graduate School.

ABSTRACT

Accurate forecasts have a direct impact on how we prepare for different weather events at personal, regional, and global levels. Many of the current numerical weather prediction (NWP) systems use legacy codes that are not adequately designed to take advantage of current and future modern compute resources. As we prepare for the Exascale era and the next generation of weather forecast systems, the ability of theses codes to efficiently use the compute resources is paramount for meeting the time requirement of forecasting and the desired resolution of 1 km. Many of the NWP codes are multidisciplinary in nature, combining building blocks from various areas of physics and atmospheric sciences which introduces the challenge of stitching these building blocks together. For example, some NWP systems use different meshes for the dynamics and the physics. This difference introduces negative and nonphysical quantities when mapping between physics and dynamics meshes. In this context, a mapping that does not preserve positivity leads to unstable simulation and a positive bias in the prediction of quantities such as moisture. This research focuses on 1) investigating different approaches for accelerating the physics schemes in NWP codes; and 2) developing a high-order positivity-preserving method (`https://github.com/ouermijudicael/HiPPIS`) for mapping solution values between different meshes.

For Zambende (father), Rosalie (mother), and Melissa (wife)

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Accurate weather forecasting has a direct impact on how we prepare for different weather events at personal, regional, and global levels. For example, accurately predicting severe weather events helps save lives, minimize economic losses, and support emergency management and mitigation. The Weather Research and Forecasting (WRF) [97] model is an example of a widely adopted numerical weather prediction (NWP) software suite used by atmospheric researchers and weather forecasters at operational centers worldwide. WRF was developed to help scientists study and better understand weather phenomena.

The improvements to NWP and the WRF model have been made possible because of the advances in science and technology over several decades. These advances are tightly coupled with the availability and improvement of computational resources. Fig. 1.1, taken from [74], shows the computational performances required for weather and climate prediction at different scales. For example, running global NWP models with 1 km resolution requires solving about 1000 prognostic variables over 10^8 grid points with small time steps for multiple ensemble members [2]. The increase in resolution allows for physical processes to be replaced by explicit representations based on fundamental principles. These simulations require extremely large computational resources, sophisticated computational techniques, and numerical methods to efficiently utilize large-scale systems. The Exascale systems such as Frontier are possible candidates for reaching the desired resolution of 1 km. However, running NWP codes on such systems presents several performance challenges that need to be addressed to be able to reach the desired resolution. As a part of the effort to modernize and advance NWP for exascale systems and beyond, this dissertation focuses on: 1) investigating different approaches for accelerating the physics schemes in NWP codes;

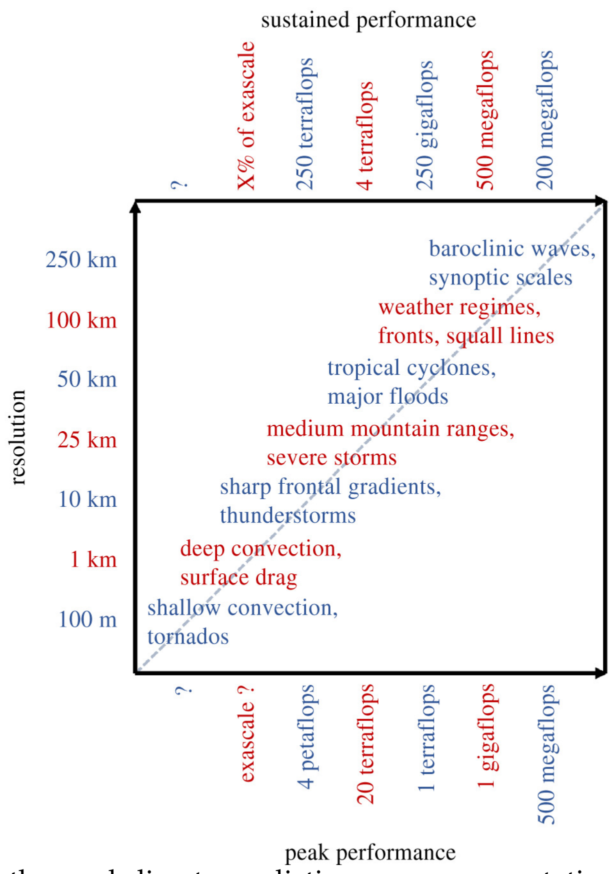


Fig. 1.1: Scales in weather and climate prediction versus computational resources required to resolve them [74].

and 2) developing a positivity-preserving mapping between the physics and dynamics coupling (PDC) in NWP.

1.1.1 Optimization of Physics Schemes

As part of the effort to prepare NWP for the exascale era and beyond, the first part of this dissertattion focuses on developing performance optimization techniques to help accelerate the physics schemes in NWP codes. The physics schemes represent the physical parameterization of processes that are unresolved at the grid resolution. These legacy physics codes are often written with complex control flow and key words such as exit, goto, and cycle that prevent thread and vector parallelism at the node level. Traditionally, MPI-level [31] distribution of serial codes has been the primary vehicle for exploiting parallelism weather codes. In the last decade, various computational architectures have increased the core counts per node, decreased the clock frequency, and adopted wide SIMD vector units. This growing complexity of computing architectures makes it difficult to develop and maintain performance-portable codes. For this reason, codes such as WRF must be restructured to leverage thread and SIMD parallelism on modern architectures while maintaining data and temporal locality.

One example of a modern code written with future architectures in mind is the Navy Environmental Prediction System Utilizing a Nonhydrostatic Engine (NEPTUNE) [49]. NEPTUNE couples the scalable dynamical core NUMA, proposed by Giraldo et al. [34], with physics schemes, such as the WSM6 and GFS, for unresolved physical processes. For instance, WSM6 uses a physical parameterization that simulates processes in the atmosphere that cause precipitation in the form of rain, snow, graupel, water vapor, cloud water, and cloud ice. The dynamics part of NEPTUNE is both fast and scalable [73], but one of a number of remaining challenges is the performance of the physics routines. For this reason, the work in this dissertation focuses on optimizing GFS and WSM6 using approaches that have applications to other numerical methods. These performance optimizations focus on restructuring the legacy physics code to enable and improve parallelism on computational nodes. The optimization efforts described here target the Intel multicore systems and potential future computer architectures, which may employ similar multicore architectures that achieve performance through vector units. This work employs OpenMP 4 as a vehicle for portability across various platforms, as OpenMP 4 is a well-established and widely adopted interface for shared-memory parallelism. Node level performance is necessary for efficiently using large-scale systems to help reach the desired 1 Km resolution for global NWP.

1.1.2 Positivity-Preserving Mapping

The second part of this dissertation introduces a novel high-order data-bounded and positivity-preserving interpolation and evaluates the use of both methods for mapping solutions values between physics and dynamics meshes. A number of key scientific computing applications that are based upon high-order methods over tensor-product grid constructions, such as numerical weather prediction (NWP) and combustion simulations, require property-preserving interpolation. Property preservation often manifests itself as a requirement for either data boundedness or positivity preservation. The particular application motivating this work is the NWP code NEPTUNE. NEPTUNE makes use of the Nonhydrostatic Unified Model of the Atmosphere (NUMA) [34] three-dimensional spectral element dynamical core, but currently uses physics routines that were developed assuming uniform grid spacing. At least two options are available for combining these two NWP building blocks: either 1) evaluate the physics routines at the (nonuniformly spaced) quadrature points on the spectral element with an acknowledgment that a modeling “crime" has been accomplished; or 2) interpolate between the grid (quadrature points) on which the dynamics is calculated to a grid on which the physics is calculated, and hence incur an interpolation error. Since there is a long-standing history of using the validated physics routines designed for use on uniformly spaced grids, there is a strong incentive to apply the second option. However, interpolating density or other key physical quantities without accounting for property preservation may lead to negative values that are nonphysical and result in inaccurate representations and/or interpretations of the physical data. For example, Skamrock et al. [98] demonstrated that not preserving positivity may lead to a positive bias in a predicted physical quantity of interest (e.g., prediction of moisture).

1.2 Overview of WRF/NEPTUNE

The Weather Research and Forecasting (WRF) [97] Model is open-source NWP software developed for both atmospheric research and operational forecasting needs. WRF uses different dynamical cores to solve to fundamental governing equations in NWP. The advanced research model (ARW) [97] and the nonhydrostatic mesoscale model (NMM) are example of widely used dynamical cores/solvers in WRF. The WRF model include several physics packages used for parameterization. As with many other computational models, significant effort is put into modernizing numerical weather codes for current and future architectures. In the case of weather codes, the goals are to improve the accuracy and reduce the time requirements of forecasts.

The code optimization work described here is related to an activity to improve the performance of the Navy Environmental Prediction System Utilizing a Nonhydrostatic Engine (NEPTUNE) [49]. The NEPTUNE code couples the Nonhydrostatic Unified Model of the Atmosphere (NUMA), of Giraldo et al. [34], with physics schemes such as the WSM6 and GFS schemes considered here. The same physics schemes are used in both NEPTUNE and WRF. NUMA is novel in that it makes use of a three-dimensional hexahedral spectral element technique with a sphere-centered Cartesian coordinate system. The NUMA spectral element method is potentially a good choice for modern computer architectures as it has relatively large floating point operations count for a relatively small communication footprint, which helps with large scalability. However, to make use of this potential for good performance, it is important to ensure that the appropriate physics schemes, such as WSM6 and GFS, perform well. The first part of this dissertation evaluates and develops performance approaches for the physics schemes in NEPTUNE. Given the same physics routines are used in both NEPTUNE and the WRF models, the performance optimization strategies developed for the physics routines are suitable for both NEPTUNE and WRF.

The dynamical core used in NEPTUNE is different from the one in the WRF model. For example, the AWR model uses weighted essentially non-oscillatory methods to solve the fundamental PDE equations whereas NUMA, used in NEPTUNE, is based on high-order spectral methods [34]. Whereas the same mesh is used for the physics and dynamics calculations in the WRF model, NEPTUNE uses different meshes and the solution values are mapped from between the meshes. The interpolation methods developed in this dissertation for preserving positivity when mapping solution values between meshes is not required for the WRF models because the same mesh is used for both Physics and Dynamics. However, a positivity-preserving method is required used in cases where the meshes are different, and preserving positivity of quantities such mass, density, and concentration are required.

1.3 Contributions

We propose to advance NWP forecasting and accuracy by: 1) developing techniques to improve computational performance of physics schemes on current and emerging architecture; and 2) introducing an interpolation method to preserve positivity and improve accuracy in physics-dynamics coupling.

- The first contribution, presented in Chapter 2, outlines code transformations necessary to enable thread vector parallelism in NWP. This research effort resulted in the publication of [79]. Legacy codes often use complex control flow and keywords such as *exit* and *goto* that prevent parallelism because in these cases the termination criteria are not known a priori. Ouermi et al. [79] use WSM6 to demonstrate the transformation required to enable parallelism without a full rewrite of the routines. Chapter 2 dissects WSM6, a micro-physics scheme used in NEPTUNE, and examines OpenMP optimization of individual, using synthetic examples and subsequently in WSM6. This study incorporates the overhead associated with thread invocation and uses OpenMP directives for thread and vector parallelism on KNL. Extending the lessons learned from the synthetic examples to WSM6 delivers over 50x speed-up over serial for several loops.

- The second contribution, presented in Chapter 3, introduces optimization approaches that consist of further code restructuring and data layout transformations to improve shared memory parallelism. This effort has led to the publication of [80] and [81]. Parallelization of physics schemes is challenging because of different parameterization models and the adaptive state transition in each model that leads to load imbalances. Both [80] and [81] couple the thread-local structure of arrays (thread-local SOA) with code and data reorganization to expose more parallelism and locality and reduce memory traffic. The studies and examples in Chapter 3 demonstrate the benefits of a high-level optimization using thread-local SOA, coupled with low-level SIMD using OMP SIMD. The optimized versions of WSM6, GFS physics, and GFS radiation run 70, 27, and 23 times faster, respectively, on KNL, and 26, 18, and 30, times faster, respectively, on Haswell compared to their respective original serial versions. Although this work targets WRF physics schemes, the findings are transferable to other performance optimization contexts and provide insight into the optimization of codes with complex physical model models for present and near-future architectures with many core and vector units.

- The third contribution, presented in Chapter 4, develops new data-bounded interpolation (DBI) and positivity-preserving interpolation (PPI) methods that preserve data boundedness and positivity for function approximation and mapping solution values between different meshes. This research is published in [82]. Due to physical constraints, quantities such as mass, concentration, density, and the cloud mixing ratio must remain positive when mapping between different meshes. Building on previous work by Berzins [5], the work in Chapters 4 improves upon the DBI method and introduces a new constrained PPI approach that can be used to ensure that positivity is preserved when mapping solution values between nonuniform structured meshes. The DBI and PPI methods provide theoretical estimates for sufficient conditions used to ensure data boundedness and positivity preservation. The new approach used here both generalizes the DBI method to nonuniform meshes and extends the approach to preserve positivity (positivity-preserving interpolation PPI) rather than the more restrictive data-bounded approach in [5].

- The fourth contribution, presented in Chapter 5 and 6, is open-source software for high-order data-bounded and positivity-preserving interpolation (HPPIS) and an extensive evaluation of the DBI and PPI methods for function approximation and mapping data values between meshes with examples pertaining to NWP. This effort led to a technical report [77] and a manuscript submitted for publication [78]. The new approaches are compared against several typical algorithms in use on a range of test problems. The results obtained show that the new methods are competitive in terms of observed accuracy while at the same time preserving the underlying positivity of the functions being interpolated. The different test functions include smooth, C^0-continuous, discontinuous, and steep-gradients. The comparison undertaken in this chapter focuses on how accurately the different methods can represent this underlying set of test examples, including representative weather examples. In addition to the software, this work provides an analysis of the mapping error in the context of PDEs, uses several one-dimensional and two-dimensional numerical examples to demonstrate the benefits and limitations of HPPIS, and introduces different strategies to improve locality, vectorization, and overall, the performance of the data-bounded and positivity-preserving interpolation methods in HPPIS.

1.4 Outline

The remaining chapters of the dissertation are organized as follows: Chapter 2 investigates parallelism challenges in NWP codes using WSM6 as a starting example. These challenges are addressed by developing and evaluating different code restructuring approaches to enable thread and vector parallelism in NWP codes. In addition, Chapter 2 evaluates the overhead associated with using OpenMP directives for thread and vector parallelism.

Chapter 3 focuses on improving the performance of NWP physics scheme by refining the data and temporal locality and further extending the code restructuring to expose more parallelism. These data structures and code transformations employ thread-local structure of arrays to increase locality vectorization. The second part of this dissertation starts with Chapter 4, which introduces the mathematical framework developed to build a new data-bounded and constrained positivity-preserving method. Chapter 5 presents a set of test examples used to compare both the data-bounded and positivity-preserving methods against other interpolation methods. Chapter 6 provides a description of an open-source software implementation of the DBI and PPI methods and evaluates both methods for function approximation and for mapping solutions values between meshes with examples pertaining to NWP. Chapter 7 provides concluding remarks and potential future research directions in preparation for NWP codes for Exascale systems and beyond.

CHAPTER 2

PARALLELISM CHALLENGES IN WRF CODES

This chapter investigates the parallelism challenges encountered in WRF physics schemes with a particular focus on WSM6 single-moment 6-class [44], as a starting example. We dissect the WSM6 single-moment 6-class [44] microphsyics code in the context of NEPTUNE [49] and examine OpenMP optimization of individual loops, first using synthetic examples and subsequently in WSM6 itself. The experiments suggest several interesting findings – particularly involving thread invocation time on Xeon and KNL and the effectiveness of OMP SIMD on code with branching and nested subroutines. Extending these lessons to WSM6, straightforward OMP DO SIMD constructs can deliver greater than 50x speed-up over the serial versions for several loops. Moreover, although not the most effective, low-level OpenMP with OMP DO SIMD can be a valid approach for accelerating serial codes with minimal changes to the source code.

The WRF single-moment 6-class microphysics scheme (WSM6) is a physical parameterization that simulates processes in the atmosphere that cause rain, snow, graupel, water vapor, cloud water, and cloud ice. WSM6 is an improved version of WSM5 that introduces graupel particles and other variables to better model the precipitation of hydrometeors. The computation in the scheme is organized along both horizontal and vertical directions. There is no interaction among the horizontal grid points, which allows straightforward parallelism cases.

2.1 Background

Traditionally, MPI-level [31] distribution of serial codes has been the primary vehicle for exploiting parallelism in these predominately serial Fortran weather codes. However, in the last decade in particular, computational architectures have increased core counts, decreased clock speeds, and adopted wide SIMD vector units. The Intel Xeon Phi Knights

Landing (KNL) [99] architecture, for example, employs dual 8-lane double precision (DP) floating point units on each of 64 cores running at 1.3 GHz. MPI alone is not suited for this fine granularity; codes must be rearchitected to exploit thread and SIMD parallelism.

OpenMP [76] is a compelling model for portable parallelism in that it requires relatively little modification of potentially large, complex codes. However, actual best practices for OpenMP vary widely with the code in question, compiler implementation, and underlying architecture. In the past, most effective OpenMP optimizations have used high-level parallel constructs for threading (i.e., mirroring MPI-level parallelism), carefully aligned arrays, and explicit rewrites to eliminate branching. These optimizations are no doubt effective, but require significant modification of existing codes. However, new architectures such as KNL boast lower thread creation times and no longer carry the same penalty for unaligned memory access. OpenMP 4 features such as OMP SIMD promise control over how vectorization is expressed, beyond the autovectorization capabilities of the compiler. Consequently, as OpenMP matures, "naive" approaches may prove almost as effective as wholesale rewrites.

Various optimization approaches have been applied to different components of NEPTUNE and other weather prediction systems. Michalakes et al. [67] optimized the Weather Model Radiative Transfer Physics by restructuring the code to expose concurrency, vectorization, and locality. In this approach, they explicitly reorganized the arrays dimension, and lowered the inner loop size to fit into the vector lane in order to take advantage of the vector units. These optimizations yielded about 3x speed-up.

OpenMP is increasingly becoming the standard for shared memory parallelism. It offers a simple high-level abstraction for thread and vector parallelism. In order to leverage OpenMP features, different groups had investigated the overheads associated with OpenMP directives [6], [7], [8], [57]. The overheads are dependent not only on the OpenMP implementation but also on the architecture. LaGrone et al. [57] developed a benchmark for measuring the overhead associated with the tasking model and the synchronization in OpenMP on 2.27 GHz 8-core Intel Xeon Nethalem E5520 processors. Dimakopoulos et al. [20] studied OpenMP overheads under nested parallelism for different compilers. In both these studies, the authors extended the EPCC benchmark to include the OpenMP directives of interest. In this work, we investigated the overheads on the Intel Knights

Landing and the effort necessary to minimize such overhead.

2.2 Experimental Setup and Methodology

2.2.1 Methodology and Measurement Parameters

In order to systematically and rigorously investigate the performance bottlenecks in WSM6, this work used a methodology that consisted of four steps.

1. Understanding code: This consisted of analyzing the loop structures and the data dependencies that exist among the loops.

2. Identifying bottlenecks: This step profiles the code using Intel VTune and wall clock timers to identify the bottlenecks.

3. Building and testing standalone experiments base on bottlenecks: We designed standalone experiments to address the bottlenecks identified in the previous steps. These experiments allow us to identify which approach is better suited for a specific bottleneck in WSM6.

4. Applying findings to WSM6: The findings in step three guide the different optimization decision in WS6 loops.

This section summarizes experiments conducted to explore various optimization strategies for the WRF WSM6 module on the Intel Knights Landing (KNL). This effort focuses on understanding the KNL and the steps necessary to effectively exploit the resources offered by the KNL architecture. Performance of a given code is evaluated using seven attributes: number of threads, serial time, parallel time, work/thread, overhead, speed-up, and efficiency.

- **Overhead**: the overhead associated with thread creation, thread binding, scheduling, etc. can be defined as in [57]:

$$Overhead = \frac{n \times T_p - T_s}{n} \tag{2.1}$$

where n is the number of threads, T_s is the serial time, and T_p is the parallel time.

- **Work/thread**: this corresponds to the average work, in Floating Point Operations Per Second (FLOPS), per thread. The work per thread is calculated by dividing the total amount of work in a loop by the number of threads used:

$$w_{thread} = \frac{\text{work in loop}}{\text{number of threads}}. \tag{2.2}$$

- **Efficiency**:

$$Efficency = \frac{T_s}{n \cdot T_p} \tag{2.3}$$

where n is the number of threads, T_s is the serial time, and T_p is the parallel time.

2.2.2 KNL Architecture

This study used KNL because it was the intended architecture to be used for NEPTUNE. The insights and code transformations required for parallelism are transferable to other multicore systems. The Intel Knights Landing architecture consists of 36 tiles interconnected with a 2D mesh, MCDRAM of 16G High Bandwidth, and one socket. It has a clock frequency of 1.3 GHz, which is lower than the 2.5 GHz of Haswell. The Knights Landing tile is the basic unit that is replicated across the entire chip. The tile consist of two cores, each connected to two vector processing units (VPUs). Both cores share a 1 MB L2 cache. Two AVX-512 vector units process eight double-precision lanes each; a single core can execute two 512-bit vector multiply-add instructions per clock cycle. The results and experiments presented in this chapter use the default thread and processor binding.

2.3 Standalone OpenMP Fortran Experiments

This section describes standalone experiments designed to verify the functionality of OpenMP, and mimic the behavior of WSM6 in a minimal reproducible fashion. In the following pseudocode, $work(i, j)$ denotes computations similar to

$$a(i, j, 1) = 0.1 * b(i, j, 1) + c(i, j, 1)/d(i, j, 1).$$

The computation is always the same, but different outer dimensions are used to simulate access of multiple arrays. This behavior is similar to the array operations in WSM6.

2.3.1 Overhead Associated with OpenMP on KNL

2.3.1.1 Overhead Per Thread Minimization

This experiment analyzes different methods that aim to minimize the overhead/thread. It determines the amount of work in FLOPS per thread required to minimize the overhead and maximize the speed-up. Synthetic examples similar to the loops in WSM6 are used for this experiment. Code 1, shown below, is used to measure a baseline overhead. Code 2 is used to analyze the overhead/thread of a WSM6-like loop. The variables ie and je are 10592, and 39, respectively. In theory, given “sufficient" work for each thread, the performance results from Code 2 should be comparable to Code 1 results.

```
Code 1:
!$OMP PARALLEL
!$OMP DO
DO i=1,100
  work(i)
ENDDO
!$OMP END DO
!$OMP END PARALLEL
```

```
Code2:
 !$OMP PARALLEL
 !$OMP DO
 do j=1, je
   do i=1, ie
     work(i,j)
  end do
end do
!$OMP END DO
!$OMP END PARALLEL
```

The results from Table 2.1 show that the average overhead/thread is less than 1 microsecond. By increasing the number of threads, the number of FLOPS per thread decreases. This decrease causes the overhead per thread to increase. The minimal overhead/thread, 0.1 microsecond, is observed at about 1 million FLOPS per thread. However, with 0.03 million FLOPS per thread, the measured overhead remains below 1 microsecond.

The results from Table 2.2 show higher overheads and lower speed-ups compared to the Table 2.1 results’. Code 2 is structured differently compared to Code 1. The computation in *work(i)* is done with a 1D array, and the computation in *work(i,j)* is done with a 2D array. Furthermore, the dimensions of the arrays are different. These differences explain different observed overheads and speed-ups.

When the !$OMP PARALLEL and !$OMP DO are moved to the i loop or !$OMP PARALLEL at the j loop and !$OMP DO at the i loop, larger overheads and lower speed-ups occur, compared to the results from Tables 2.1 and 2.2.

Table 2.1: Performance results from Code 1.

n	T_s	T_p	Overhead	w_{thread}	Speed-up	Efficiency
2	92.797	46.500	0.22	1000000	1.995	99.78
4	92.865	23.335	0.51	500000	3.98	99.49
8	92.770	12.209	5.00	250000	7.59	94.98
16	92.826	6.608	12.26	125000	14.05	87.79
32	92.944	3.831	24.27	62500	24.26	75.82

Table 2.2: Modified Code 2 with !$OMP DO placed outside j loop.

n	$T_s(\mu s)$	$T_p(\mu s)$	Overhead (μs)	w_{thread} (FLOPS)	Speed-up	Efficiency %
2	27.636	15.848	2.03	1858896	1.74	87.19
4	27.352	8.099	1.26	929448	3.38	84.43
8	27.439	4.108	0.68	464724	6.68	83.49
16	27.422	2.586	0.87	232362	10.60	66.27
32	27.507	1.780	0.92	116181	15.45	48.29

2.3.1.2 Keeping Threads Active/Alive

Keeping threads active/alive during computation reduces thread creation and cancellation overheads. !$OMP PARALLEL is the directive that creates the pool of threads (fork), and !$OMP END PARALLEL cancels the created threads (join). Thus, creating threads at the beginning of a computation and canceling them at the end should reduce the overhead associated with the creation and cancellation of threads. Furthermore, the OpenMP environment variable KMP_BLOCKTIME can be used to keep threads alive for a certain time. This experiment compares a single parallel block performance against multiple parallel blocks. One would expect Code 4 to outperform Code 3. Because Code 4 is constructed with a single parallel block, it does not incur the thread creation overhead caused by the multiple !$OMP PARALLEL blocks.

```
Code 3:
!$OMP PARALLEL
!$OMP DO
  do j=1, je
    do i=1, ie
      3 x work(i,j)
     end do
     end do
 !$OMP END DO
 !$OMP END PARALLEL

   !$OMP PARALLEL
```

```
Code 4:
!$OMP PARALLEL
!$OMP DO
  do j=1, je
    do i=1, ie
      3 x work(i,j)
    end do
  end do
  !$OMP END DO

  !$OMP DO
  do j=1, je
```

```
!$OMP DO
do j=1, je
  do i=1, ie
    3 x work(i,j)
  end do
end do
!$OMP END DO
!$OMP END PARALLEL

!$OMP PARALLEL
!$OMP DO
do j=1, je
  do i=1, ie
   3 x work(i,j)
  end do
end do
!$OMP END DO
!$OMP END PARALLEL
```

```
  do i=1, ie
    3 x work(i,j)
  end do
end do
!$OMP END DO

!$OMP DO
do j=1, je
  do i=1, ie
    3 x work(i,j)
  end do
end do
!$OMP END DO
!$OMP END PARALLEL
```

Table 2.3 and Table 2.4 show performance results for multiple parallel blocks and a single parallel block, respectively. The single parallel block from Code 4 performs slightly better than the multiple blocks case from Code 3, which supports the initial assumptions.

Table 2.3: Multiple parallel blocks with KMP_BLOCKTIME in Code 3.

n	T_s	T_p	Overhead	Speed-up	Efficiency
2	81.597	47.119	6.32	1.73	86.58
4	81.398	24.080	3.73	3.38	84.50
8	81.222	12.360	2.20	6.57	82.14
16	81.357	7.479	2.39	10.87	67.98
32	81.755	5.150	2.59	15.87	49.60

Table 2.4: Single parallel block with KMP_BLOCKTIME in Code 4.

n	T_s	T_p	Overhead	Speed-up	Efficiency
2	81.597	46.300	5.50	1.76	88.12
4	81.398	23.560	3.21	3.45	86.37
8	81.222	12.021	1.87	6.75	84.46
16	81.930	7.188	2.07	11.39	71.24
32	81.755	5.028	2.47	16.26	50.81

2.3.1.3 OpenMP Versus Pthreads Overhead

This experiment analyzes the overhead associated with thread creation and context switches in the OpenMP and Pthreads libraries. In order to establish a fair comparison of both libraries, the synthetic experiment has been done in C, because Pthreads does not have an equivalent in Fortran.

Table 2.5 shows the thread creation overhead and context switches measurements for Pthreads and OpenMP. These results indicate that OpenMP has significantly higher thread creation overhead compared to Pthreads. The context switch measurements observed in OpenMP are slightly but not significantly higher than the ones in Pthreads but not significant. These experiments show that the use of a single parallel block coupled with the environment variable KMP_BLOCKTIME contributes to reducing the overheads and increasing the speed-ups slightly.

2.3.1.4 KNL vs Haswell Overhead

Thread overhead is dependent on the implementation of OpenMP and the architecture used. Here, performances of Code 1 on KNL and Haswell are compared. The results from Tables 2.1 and 2.6 indicate that KNL has a lower overhead than Haswell. The average overhead on KNL is about 0.51 whereas the overhead on Haswell is about 0.61. In addition, the speed-ups observed on KNL are greater than the ones on Haswell.

Table 2.5: Thread creation and context switch overhead measurements with Pthreads and OpenMP.

	Pthreads		OpenMP	
n	Thread creation	Context	Thread creation	Context
2	199	0.631	14311	6.017
4	183	0.736	8047	3.336
8	121	1.182	4209	1.375
16	107	1.067	4115	1.046
32	102	1.041	1654	0.797
64	99	1.035	1417	0.943
128	120	1.27	653	1.579

Table 2.6: Performance results from Code 1 on Haswell.

n	T_s	T_p	Overhead	w_{thread}	Speed-up	Efficiency
2	67.825	34.225	0.31	1000000	1.981	99.08
4	67.77	17.263	0.32	500000	3.925	98.14
8	67.729	9.079	0.61	250000	7.459	93.25
16	68.099	5.015	0.76	125000	13.579	84.86
32	67.973	2.861	0.73	62500	23.758	74.25

2.3.2 Thread Scalability

2.3.2.1 Base Case

The previous section analyzed examples of loops that do not exhibit significant complexity. This section focuses on understanding the performance impact of function calls. The example examined here is a loop with nested functions calls. This experiment analyzes a base case that is used as a reference. Code 5 measures how its performance scales with the number of threads before transformation.

```
Code 5 : WSM6 loop with conditionals and function calls
do k = kte, kts, -1
  do i = its, ite
    ...
    if(t(i,k).gt.t0c) then
        ...
      work2(i,k) = venfac(p(i,k),t(i,k),den(i,k))
      if(qrs(i,k,2).gt.0.) then
       ...
        psmlt(i,k) = xka(t(i,k),den(i,k))
        ...
      endif
      if(qrs(i,k,3).gt.0.) then
        ...
        pgmlt(i,k) = xka(t(i,k),den(i,k))
        ...
      endif
    endif
  enddo
enddo
```

A close analysis of Code 5 shows two function calls: *xka* and *venfac*. These functions are implemented by calling two other functions, *viscos* and *diffus*. Furthermore, all the functions call intrinsic math functions such as *sqrt*, for which most current compilers now emit vector instructions.

Table 2.7 shows that Code 5 scales up to eight threads. After eight threads, the overhead increases drastically and the speed-up plateaus at about 9x. As mentioned before, Code 5 has nested functions and conditionals. Such complexities may cause performance limitations for threading and vectorization. Table 2.8 shows performance results from Code 5 with all function calls and conditionals removed. By removing the function calls, the amount of computation in the loop is significantly reduced. This reduction resulted in the serial time in Table 2.8 being much smaller than that in Table 2.7. Table 2.8 has significantly higher speed-ups and lower overheads than Table 2.7. These results indicate that the conditionals and the function calls are responsible for the performance limitations observed.

2.3.2.2 Function Calls Performance Analysis

As mentioned above, Code 5 has nested function calls. This experiment compares the performance of a modified version of Code 5 against Code 6. In the modified version of Code 5, the conditionals have been removed but the function calls are left intact. In Code 6, the function calls are replaced by some code that performs the same task as the functions.

```
Code 6 : WSM6 complex Code with no function calls.
    !$OMP PARALLEL DEFAULT(shared) PRIVATE(i, k)
```

Table 2.7: Performance results from Code 5.

n	T_s	T_p	Overhead	Speed-up	Efficiency
2	2109.055	944.316	-110.21	2.23	111.67
4	2107.635	499.088	-27.82	4.23	105.57
8	2109.226	262.901	-0.75	8.02	100.27
16	2109.020	234.1	102.28	9.01	56.30
32	2110.18	231.294	165.35	9.12	28.51
40	2109.158	218.183	165.45	9.67	24.16

Table 2.8: Code 5 without function calls and conditionals.

n	T_s	T_p	Overhead	Speed-up	Efficiency
2	90.567	57.241	11.95	1.58	79.11
4	90.427	29.06	6.45	3.11	77.79
8	90.480	14.7601	3.45	6.13	76.62
16	90.581	8.721	3.06	10.39	64.92
32	90.631	5.882	3.05	15.41	48.15
40	90.973	3.064	0.79	29.69	74.22

```
  !$OMP DO
  do k = kte, kts, -1
    do i = its, ite
      ...
      !!--work2(i,k) = venfac(p(i,k),t(i,k),den(i,k))
      temp0 =  1.496e-6 * (t(i,k)*sqrt(t(i,k))) /  &
                  (t(i,k)+120)/den(i, k)
      temp1 =  8.794e-5 * exp(log(p(i,k))* (1.81)) / t(i,k)
      work2(i,k) = exp(log((temp0/temp1))* ((.3333333)))  &
                 /sqrt(temp0)*sqrt(sqrt(den0/den(i,k)))

      !!-- xka(t(i, k), den(i, k))
      temp3 = 1.414e3*1.496e-6 * (t(i,k)*sqrt(t(i,k)))/
                  (t(i,k)+120)/den(i, k)*den(i,k)
      ...
  enddo
enddo
!$OMP END DO
!$OMP END PARALLEL
```

The modified version of Code 5 yielded a maximum speed-up of about 3x whereas Code 6 yielded a maximum speed-up of 62x. Table 2.9 report the results from Code 6.

2.3.2.3 Subroutine Calls Performance Analysis

This experiment measures the performance impact of subroutine calls. Code 7 below contains a subroutine call and a few conditionals.

```
Code 7
!$OMP PARALLEL DEFAULT(shared) PRIVATE(i,j,m,thread_id)
do j=1, je
  thread_id = OMP_GET_THREAD_NUM()

  !$OMP DO SIMD
```

Table 2.9: Performance results from Code 6.

n	T_s	T_p	Overhead	Speed-up	Efficiency
2	1909.236	594.708	-359.910	3.21	160.51
4	1909.292	297.209	-180.114	6.42	160.60
8	1909.105	149.470	-89.17	12.77	159.65
16	1909.222	89.801	-29.52	21.26	132.88
32	1910.795	60.320	0.61	31.68	98.99
40	1910.146	30.584	-17.17	62.45	156.15

```
   do i=1, ie
     do m=1, M_LOOPS
     #if OMPTEST_SUBROUTINE
     call do_work(i,j)
     #else
    3 x work(i,j)
     if (b(i,j,1) .gt. 0.0) then
      3 x work()
     endif
     #endif
     enddo
   enddo
   !$OMP END DO SIMD NOWAIT
   end do
   !$OMP END PARALLEL

 subroutine do_work(i,j)
 !$OMP DECLARE SIMD(do_work)
 integer :: i,j
3 x work()
 if (b(i,j,1) .gt. 0.0) then
  3 x work(i,j)
 endif
 end subroutine do_work
```

Table 2.10 reports the following experimental cases:

- case 1 represents results from Code 7 with OMP PARALLEL + OMP DO SIMD;
- case 2 represents results from Code 7 with OMP PARALLEL + OMP DO SIMD and subroutine; and
- case 3 represents results from Code 7 with OMP PARALLEL + OMP DO SIMD, subroutine and DECLARE SIMD on functions.

Table 2.10 shows the performance results from Code 7. The DECLARE SIMD use in this experiment does not have a significant performance impact. Table 2.10 cases 2 and 3

Table 2.10: Code 7 results.

case	n	T_s	T_p	Speed-up	Efficiency
case 1	64	69	1.46	47.26	73.84
case 2	64	69	1.45	62.09	74.35
case 3	64	90	1.48	60.81	95.02

report additional variations that were tested, as indicated by their captions. These results show significant speed-ups, which indicates that a single-level (no nesting) subroutine and conditionals are not a performance bottleneck, and various combinations of !$OMP PARALLEL and !$OMP SIMD yield significant speed-ups.

2.3.2.4 Nested Conditionals

This experiment focuses on studying the performance impact of conditionals. It compares the performance of Code 9 against a modified versions of Code 6. The modified version of Code 6 includes nested conditionals.

Code 9 : WSM6 loop with masking and no function calls.

```
compute bool_val1, bool_val2, and bool_val3
 !$OMP PARALLEL DEFAULT(shared) PRIVATE(i, k)
 !$OMP DO
 do k = kte, kts, -1
   do i = its, ite
     ...
      temp0 =  1.496e-6 * (t(i,k)*sqrt(t(i,k)))/(t(i,k)+120) &
               /den(i, k)
      temp1 =  8.794e-5*exp(log(p(i,k))*(1.81))/t(i,k)
      work2(i,k) = exp(log((temp0/temp1))* ((.3333333))) &
         /sqrt(temp0)*sqrt(sqrt(den0/den(i,k)))
     ...
     compute result1
       ...
       temp3 = 1.414e3*1.496e-6 * (t(i,k)*sqrt(t(i,k))) &
               /(t(i,k)+120)/den(i, k)
     ...
     compute result2
     final_result = (result1*bool_val1 + result2*bool_val2)&
                    *bool_val0
    enddo
  enddo
  !$OMP END DO
  !$OMP END PARALLEL
```

The modified version of Code 6 obtains a maximum speed-up of about 8x. Using SIMD decreases the speed-up to about 3x. SIMD fails to vectorize because of the nested conditionals. Moving the conditionals outside the loops, as shown in Code 9, to address the bottleneck yielded significant speed-ups and low overheads, as shown in Table 2.11. These experiments indicate that nested conditionals hurt performance. Eliminating branching yields significant improvements. This approach can be used in many of the WSM6 codes that exhibit similar patterns.

2.3.3 Vectorization

2.3.3.1 OMP SIMD

This section analyzes the performance impact of SIMD. In this experiment, a simple compute-only code is considered. It compares the performance of various parallel versions and Code 8 against the serial version. Furthermore, the thread binding is done manually.

```
Code 8 : WSM6 loop with masking and no function calls
!$OMP PARALLEL DEFAULT(none) SHARED(a, b, c, d, je, ie,num_tasks)
!$OMP PRIVATE(i,j,m,its,ite,Thread_id)
thread_id = omp_get_thread_num()
its = 1 + thread_id * num_tasks * VLEN
ite = min(its + num_tasks * VLEN - 1, ie)
do j=1, je
!$OMP SIMD
do i=its, ite
  do m=1, 10
   3 x work(i,j)
  enddo
  if (b(i,j,1) .gt. 0.0) then
    3 x work(i,j)
  endif
enddo
```

Table 2.12 reports the following experimental cases:

- case 1 represents results from Code 8 with OMP DO placed right before the i loop;
- case 2 represents results from Code 8 with only manual implementation of thread

Table 2.11: Performance results from Code 9.

n	T_s	T_p	Overhead	Speed-up	Efficiency
2	2102.487	691.904	-359.34	3.04	151.93
4	2099.805	344.800	-180.15	6.09	152.24
8	2099.784	172.063	-90.41	12.20	154.25
16	2106.650	104.066	-27.60	20.24	126.52
32	2112.524	69.479	3.46	30.40	95.02
40	2100.923	35.199	-17.32	59.69	141.22

binding;

- case 3 represents results from Code 8 withwith OMP DO SIMD at the i loop; and
- case 4 represents results from Code 8 SIMD and manual implementation of thread binding.

Table 2.12 shows the speed-ups when the different directives are placed right before the i loop. OMP PARALLEL + OMP SIMD yields the highest speed-up among the different experiments.

2.4 Modernization of WSM6

2.4.1 Code Overview

The WSM6 supercell test case of WSM6 consists of 27 loops around 10K (i) rows, with three subroutines (slope_wsm6, nislfv_rain_plm, nislfv_rain_plm6) [51]. The last two subroutines contain nontrivial control flow (cycle/goto statements). The other loops are generally memory intensive, with significant branching. Applying the findings of the standalone tests, WSM6 was modified with OpenMP directives as follows:

- OpenMP initialization code in init_microphysics() in mod_microphysics.f90;
- Consistent use of OMP PARALLEL and OMP DO SIMD as presented in case 3 of Table 2.12;
- Minor code modification to remove nested conditionals and function calls as demonstrated in Code 9;
- Use of OMP PARALLEL sections around multiple loops as shown in Code 3 to reduce thread invocation overhead;
- Elimination of false sharing and specification of PRIVATE variables ;

Table 2.12: Code 8 results.

case	n	T_s	T_p	Speed-up	Efficiency
case 1	64	69	2.88	24	37.43
case 2	64	69	0.63	109	171.13
case 3	64	69	0.614	112	175.59
case 4	64	69	0.614	112	175.59

- Merging smaller loops involving the same arrays, to mitigate thrashing; and
- Using C preprocessor macros to enable and measure runtime of parallel, serial or both implementations.

The decision to first pursue OMP DO SIMD, despite worse performance than OMP SIMD with manual indices from Table 2.12, was due to the presence of many temporary arrays in between loops and dependency-heavy subroutines in WSM6. Moreover, the aim is to first explore what naive “low-level" OpenMP parallelization could deliver with minimal reorganization of the code.

2.4.2 WSM6 Results

Tests were conducted on a four-socket (Haswell) Intel Xeon E7-8890 v3 with 3 TB RAM, and Intel Xeon Phi 7210 (“Knights Landing", or KNL) with 16 GB MCDRAM and 96 GB DRAM. The compiler was Intel Parallel Studio 2016, update 3 (build 67), due to issues with Parallel Studio 2017 in other modules within NEPTUNE. Results up to all cores (64 cores on KNL, 72 on Haswell) on these respective systems are shown in Table 2.13 for different values of OMP_NUM_THREADS. We see limited benefit from hyperthreading on either platform. Though not shown in the table, we found that 18 cores of KNL performed 2.12x better than a single-socket equivalent Haswell (with OMP_NUM_THREADS=18). Moreover, using the default maximum number of threads (144 on Haswell, 256 on KNL), KNL performs roughly 2x faster than Haswell core-for-core, which suggests better scalability on KNL than on Haswell. We note, however, that we used default thread affinity settings (i.e., KMP_AFFINITY). Our Brickland-EX Haswell system has a nonconventional memory architecture supporting up to 6 TB RAM, generally exhibiting higher intersocket latencies than comparable Xeon workstations, which could affect performance. Further work may be required to scale specifically on this platform.

2.5 Summary and Discussion

2.5.1 Scalability challenges

Although these results show good performance on KNL, the current implementation does not scale well beyond 36 cores (two-socket equivalent) on Xeon. This scaling problem is perhaps a result of slightly higher thread invocation time on Xeon, but is more likely

Table 2.13: Scalability and speed-up (over serial) on 72-core Haswell-EX and 64-core KNL with different values of OMP_NUM_THREADS.

	Haswell		KNL		KNL vs HSW
n	T_p	Speed-up	T_p	Speed-up	KNL vs HSW
1	0.46	1	1.77	1	0.26
4	0.116	4	0.222	8	0.52
16	0.068	6.9	0.067	26	1
32	0.067	7.9	0.04	44	1.7
64	0.064	18	0.031	57	2.1
128	0.06	19	0.035	50	1.7
256	0.19	6	0.037	47	5

Table 2.14: Wall clock time measurements of individual WSM6 loops, in milliseconds, on 72-core Haswell-EX and 64-core KNL, using all available hardware threads.

	KNL		Haswell	
Loop	T_s	T_p	T_s	T_p
Init loops	2.88×10^{-3}	–	2.92×10^{-3}	–
loop 1	16.8	1.19	6.76	1.58
loop 2	122	1.37	15.6	3.45
loop 5	46.8	1.28	15.9	1.58
loop 7	17.9	1.22	6.56	1.57
slope_wsm6	98.6	1.81	41.5	4.76
loop 8	19.1	1.33	7.77	2.10
rain_plm	260	–	39.0	–
rain_plm	220	–	62	–
loop 9-11	10.3	1.49	11.7	3.12
slope_wsm6	60.7	1.48	13.0	4.51
loop 12-14	176	2.37	36.1	1.86
loop 15-17	2.96	0.859	2.53	0.598
loop 18-19	102	2.26	13.1	2.36
slope_wsm6	59.7	1.77	12.5	4.42
loop 20-21	524	5.32	76.4	5.92
loop 22	246	4.32	119	9.91
loop 23	193	1.95	32.6	6.02
loop 24-26	156	2.99	26.8	3.65
loop 27	4.52	5.85	5.61	6.98
wsm6 total	1860	38.9	440	64.3

due to the higher clock speed of that architecture and worse “base" speed-up of threads compared to the serial version. Although multiple parallel sections scale well on KNL, these standalone experiments with thread overhead suggest that fewer parallel sections and use of manual indexing (i.e., OMP PARALLEL and OMP SIMD instead of OMP DO SIMD directives) are necessary for better Xeon performance.

Table 2.15: Speed-up over serial from Table 2.14.

	KNL		Haswell	
Loop	Speed-up	Efficiency %	Speed-up	Efficiency %
Init loops	–	–	–	–
loop 1	14.06	21.00	4.91	5.94
loop 2	89.10	139.14	4.53	6.28
loop 5	36.44	56.15	10.07	13.97
loop 7	14.67	22.92	4.16	5.80
slope_wsm6	54.56	85.12	8.71	12.08
loop 8	14.44	22.43	3.70	5.13
rain_plm	–	–	–	–
rain_plm	–	–	–	–
loop 9-11	6.90	10.80	3.73	5.20
slope_wsm6	40.87	64.08	2.95	4.00
loop 12-14	74.41	116.03	19.36	26.95
loop 15-17	3.45	5.38	4.22	5.87
loop 18-19	45.11	70.51	5.56	7.70
slope_wsm6	33.69	52.70	2.83	3.93
loop 20-21	98.48	153.90	12.93	17.92
loop 22	57.13	88.97	12.01	16.68
loop 23	99.53	154.64	5.42	7.52
loop 24-26	52.26	81.52	7.31	10.19
loop 27	0.77	1.20	0.80	1.11
wsm6 total	47.81	74.71	6.91	9.50

2.5.2 Remaining bottlenecks

Nonparallelizable and poorly parallelizable subroutines in WSM6 and mod_microphysics remain a bottleneck. Complicated subroutines with dependencies such as nislfv_rain_plm and nislfv_rain_plm6 require extensive rewrites to achieve speed-up; currently only 2x speed-up is achieved for the former. More significantly, horizontal-to-vertical memory copies of arrays in both WSM6 and mod_microphysics in Neptune are difficult to parallelize, achieving at best a 2x speed-up. These require further investigation. In addition to bottlenecks originating from loops with dependencies, the microphysics code as a whole remains bottlenecked by horizontal-to-vertical copying and integration of arrays. To address this problem, one should consider interleaving these copy statements with parallel computation, and re-evaluating how data are warehoused in the calling code. With bottlenecks, the entire microphysics module achieves only a modest 3x improvement over serial on KNL, and 2x on Xeon. Ultimately, we would like to restructure all WSM6 directives, moving from OMP DO SIMD to chunks with OMP SIMD and a single high-level

OMP PARALLEL section, similar to the approach of Michalakes et al. [67]. This approach will require parallelizing all remaining sections and eliminating copies of full arrays within WSM6.

2.5.3 Flat vs. Cache Mode on KNL

The memory modes of the Xeon Phi KNL architecture are of significant interest in many code modernization efforts. In “flat mode", the 16 GB MCDRAM are treated as main memory by the OS; in “cache mode", the MCDRAM serve as a cache for larger pool of DRAM (96 GB on the workstation). In principle, KNL hosts deployed in cache mode incur higher cache miss costs, as memory is pulled from DRAM. WSM6 results showed a negligible difference between flat and cache modes (in fact, an unexpected 1% advantage for cache mode, which is within the 5% margin of error between individual time steps of the microphysics code). The flat and cache configurations merit further investigations, but for the current work we conclude the difference between flat and cache modes is not a major factors in the runtime of WSM6.

2.5.4 Summary

Overall speed-ups achieved compared to serial versions are convincing: 57x over serial version on KNL suggests 5.6% of peak (1024x, 64 cores x 16 SIMD lanes). Although these results correspond only to easily parallelizable loops within WSM6, this study encompasses nontrivial code with branching, subroutines, and incoherent memory access. In all, the WSM6 work is encouraging in that significant speed-up was possible with relatively small changes to code – exactly what is desired in a code portability effort.

We have examined the impact of OpenMP directives on a Fortran-based WSM6 microphysics code in WRF. In standalone experiments, we measured the cost of thread overhead and tested the effectiveness of various directives with and without OMP SIMD. These results suggest that although greater scalability may be possible with high-level OpenMP constructs, parallelization of dependency-free code sections is possible with few modifications to the original code. Moreover, we have found cases in which straightforward low-level OpenMP methodologies may work, delivering satisfactory 50x–100x speed-ups over the serial version. The fact that modern compilers can emit reasonably efficient threaded and SIMD instructions from complex code with branching, subroutines, and unaligned arrays suggests the OpenMP methodology holds promise.

CHAPTER 3

PERFORMANCE OPTIMIZATION APPROACHES IN NWP PHYSICS SCHEMES

This chapter introduces high-level and low-level approaches for shared memory parallelism using thread-local structures of arrays (SOA). The thread local SOA is a layout where a structure local to a thread and the fields inside the structure are arrays. The high-level approach employed here consists of parallelizing large blocks of code at the parent level in the call stack, whereas the low-level approach targets individual instructions. Thread-local SOA and OMP SIMD are employed to accelerate computation in GFS and WSM6 modules by improving data locality and taking advantage of thread and vector parallelism. In addition, a static memory allocation process is used instead of a dynamic memory allocation process to help improve the memory performance of the GFS code. As a result of these optimizations, there has been a significant speed-up over serial versions of the code and a previously optimized version [79]. For instance, the use of SOA coupled with OMP SIMD for vectorization has led to significant speed-up improvements. The optimized versions of WSM6, GFS physics, and GFS radiation run 70, 27, 23 and 26, 18, 30 times faster on KNL and Haswell, respectively. In addition, these optimizations have enabled a speed-up of 23.3 over a prior optimized version of WSM6 [79].

WSM6, introduced in the previous chapter, is a physical parameterization that simulates processes in the atmosphere that cause precipitation in the form of rain, snow, graupel, water vapor, cloud water, and cloud ice. GFS is a weather forecast model developed by the National Center for Environmental Prediction (NCEP). GFS is a coupled model composed of an atmosphere model, an ocean model, a land/soil model, and a sea-ice model. The optimization efforts target GFS physics and GFS radiation, the two most expensive calls within the module driver. Similarly to WSM6, GFS has no dependencies along the horizontal direction, thus making it amendable to performance improvement without the concern of communication.

3.1 Background

There has been significant activity recently on porting and optimizing NWP codes on various new computer architectures. For example, Mielikainen et al. [69] optimized the Goddard microphysics scheme on an Intel Xeon phi 7120P Knights Corner (KNC) [12] by removing temporary variables to reduce the code memory footprint and by refactoring loops for vectorization, leading to a 4.7 speed-up. Furthermore, Mielikainen et al. [70] also optimized the Perdu-Lin microphysics scheme using the same approaches. Again, these approaches resulted in a 4.7 speed-up using vector alignment and SIMD directives. Similarly, Ouermi et al. [79] used a low-level optimization approach based upon OpenMP 4 [17] directives to improve the performance of WSM6 on the KNL. When combined with minor code restructuring to enable and improve locality and vectorization, this approach resulted in a speed-up of three on the whole of WSM6. This speed-up included unoptimized (serial bottleneck) code sections.

In optimizing the Weather Model Radiative Transfer Physics on Intel KNL, Michalakes et al. [68] focused on increasing concurrency, vectorization, and locality. Improving concurrency involved increasing the number of subdomains to be computed by threads. Vectorization and locality were improved by restructuring the loops to compute over smaller tiles and exposing vectorizable loops. This effort led to a threefold speed-up over the original 1.3 speed-up on Xeon Sandybridge.

Data layout plays a key role in performance optimization. The optimal data layout minimizes the memory footprint, reduces cache misses, and allows better usage of vector units. This study uses thread-local structures of arrays (SOA) data layout to improve memory access [43], [107] . The SOA approach and similar approaches have been used to accelerate many scientific applications on various architectures. Henretty et al. [40] used data layout transformation to improve the performance of stencil computations. These optimizations removed alignment conflicts, reduced cache misses, and improved vectorization. Woodward et al. [61], [106] used briquette data structures to accelerate a Piecewise Parabolic Method (PPM) code by reducing memory traffic. A briquette is a small sub-block of a uniform grid. The size of the briquette is chosen in relation to the cache size and vector unit. These data transformations have enabled high performance because they reduce the memory footprint and traffic. In addition, such transformations improve vectorization.

The work presented in this chapter relies on the OpenMP runtime system for task scheduling and OpenMP “pragma" directives for parallelization. Other approaches could be employed. Mencagli et al. [65] used a runtime support to reduce the effective latency of interthread cooperation. This latency reduction is done with a “home-fowarding" mechanism that uses a cache-coherent protocol to reduce cache-to-cache interaction. Buono et al. [9] proposed a light-weight runtime system as an approach to optimize linear algebra routines on Intel KNC [12]. This runtime system focuses on efficient scheduling of tasks from a directed acyclic graph (DAG) that is generated on the fly during execution. Danelutto et al. [18] suggested a pattern-based framework for parallelization. This parallelization approach targets known patterns that can be represented with well-known operations such as map, reduce, scan, etc.

Although this work focuses on the Intel KNL and Haswell architectures, it is important to point out that efforts have been made to port and optimize WRF physics schemes for GPUs [71], [66], [84], [22]. GPU-based optimizations show better performance than Intel KNC and KNL-based optimizations. For instance, Mielikainen et al. [71], using CUDA [75], were able to achieve a speed-up of two orders of magnitude. However, porting to GPUs often requires significant code rewrites. The present work is part of larger effort to develop and optimize a potential US Navy next generation weather code, NEPTUNE. The optimization strategies introduced in this work target Intel KNL and Haswell because the operational version of NEPTUNE is intended to run on Intel micro-architectures instead of GPUs.

3.2 Experimental Setup and Methodology

3.2.1 Strategies for OpenMP Parallelism

3.2.1.1 Task Granularity (High-Level Versus Low-Level OpenMP)

High-level parallelism refers to parallelizing large blocks of code at the parent level in the function call stack, whereas low-level refers to parallelizing smalls blocks of codes at the instruction level (i.e., loops and arithmetic operations). The high-level approach has the advantage of using few individual parallel section, and few modifications within these sections. However, the high-level approach also requires the code blocks to be thread safe and free of serial bottlenecks.

In contrast, the low-level approach has the advantage of permitting parallelism in selectively parallelizable code punctuated by serial sections. If these serial bottlenecks are not easily removed, or if their relative cost is low, this may be a valid approach. Low-level approaches may also be appropriate for codes that require multiple different parallelization approaches (i.e., static versus dynamic scheduling, tasking, etc.) within different logical blocks or subroutines. Whether high-level or low-level parallelism is best depends on the individual code in question. High-level OpenMP is typically more elegant, but requires code that is sufficiently independent to be parallelizable at the parent level in the call stack. A low-level approach requires adding more parallel directives, but allows the original code structure to be used more or less as is. High-level and low-level approaches relate to task granularity, i.e., at which level logic is parallelized within a call stack. The length and the complexity of the logic within each task may have an impact on scheduling and load balancing, as well as on intertask dependency.

3.2.1.2 Data Granularity, Chunks, and SOA

In the physics schemes in NEPTUNE, data granularity refers to the size of arrays or subarrays that are processed by each thread. Coarse-grain data parallelism corresponds to dividing up the input data into the number of worker threads, and fine-grain data parallelism corresponds to further subdividing input data into smaller chunks. The chunk size determines the size of the subdivided data, as shown in Fig. 3.1. For instance, an 2D input array ($im \times jm$) is divided into multiple 2D subarrays of sizes $chunksize \times jm$. Typically input and output data are organized in arrays of structures (AOS) and regular arrays. AOS is a layout where each array element is a structure with fields inside the structure. The SOA data layout is more suitable for vectorization compared to the AOS data layout. WSM6, GFS physics, and GFS radiation use large regular arrays and SOA. These input and output data are transformed into thread-local SOA. A thread-local SOA is an SOA that is private to a particular thread. The beneficial chunk size of the thread-local SOA is determined by the SIMD unit length (8 or 16 in the case of KNL and Haswell), or by the number of cores per block (SM) in a GPU. A more in-depth study of SOA and other data structures can be found in [43], [107], [42]

In choosing the appropriate chunk size for an optimum data granularity, the goal is to keep the data as local as possible to each thread. Ideally, within the L1 and L2 caches, it is advantageous to use thread-local data structures and copy to and from global shared-memory arrays as necessary. The thread-local data are most effective when aligned to SIMD/chunk size and organized in SOA fashion. This data transformation allows the data for each thread to be packed closely in memory, thus reducing cache misses, and requests from L3/MCDRAM (on KNL) and/or main memory.

The input and output data structures in WSM6 and GFS codes are not suitable for performance optimization because both SOA and regular arrays are large and do not fit into cache. In addition, the SOA are dynamically allocated, which requires expensive memory operations. Using thread-local SOA instead of large regular arrays and statically instead of dynamically allocated arrays enables better memory usage and vectorization. Fig. 3.1 shows examples of the data transformation. Arrays A and B represent original input and output data. The top halves of A and B are copied into a thread-local SOA that is private to the thread to which it will be assigned. The same transformation is done for the bottom halves of A and B. When the original input is a large SOA composed of A and B, the transformation would be similar, from large shared memory SOA to thread-local SOA. This thread-local SOA ensures that data required for a calculation are close together in memory, and hence fit into the cache together. Overall, this modification enables memory locality.

3.2.1.3 GFS Physics Code Modifications

Although GFS physics and GFS radiation have similarities with WSM6, additional code transformations were applied to GSF code to achieve reasonable speed-ups. Thread-local SOA, transformation from dynamic to static allocation, and low-level transformation/vectorization, described below, are the key changes implemented in GFS physics and GFS radiation to enable better performance and are now described in turn.

- Data reorganization with thread-local SOA: A thread-local SOA transformation is applied to the input and output arrays as described in Fig. 3.1. This transformation makes it possible to construct a thread-local SOA that is local to the thread to which it

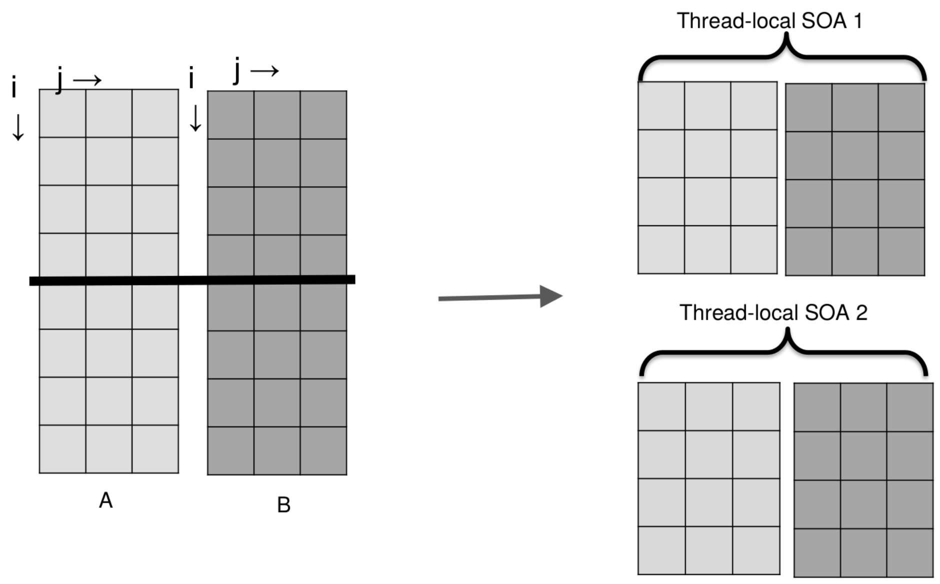


Fig. 3.1: Transformation from AOS to SOA. The 2D arrays A and B are transformed into two thread-local SOA. The top and bottom parts are put next to each other as shown on the right. The chunk size shown in blue determines how to split and A and B. If the chunk size is chosen to be two, A and B would be split into four parts, which would give four thread-local SOA.

is assigned and small enough to fit in cache. This transformation requires copying the original input and output data into the new thread-local SOA before passing it to the GFS driver function calls. In the work by Ouermi et al. [81], the data reorganization transformed regular arrays to thread-local SOA. Here, the data converted from large SOA and regular arrays to thread-local SOA.

- Dynamic to static allocation: With static allocation, the arrays sizes are known at compile time whereas in the dynamic case the arrays sizes are not known a priori. The original GFS code employs dynamic allocation for the input and output arrays in the GFS driver, which does not guarantee contiguous data. However, each array in the SOA will be contiguously allocated, but the different allocations may be far apart in memory. Thus, accessing dynamically allocated arrays is often more expensive than accessing statically allocated arrays. Instead of using the original data or SOA that are dynamically allocated, the original inputs and outputs are copied to statically allocated thread-local SOA and then passed to the function calls in the GFS driver.
- Vectorization and low-level code transformations: The GFS physics and GFS radiation do not have many serial bottlenecks that require major code transformation at a

low-level as in WSM6 with niflv_rain_plm6 and niflv_rain_plm. Auto-vectorization often fails to vectorize large body loops, or relatively complex code. Given that there are not many serial bottlenecks, the OpenMP directive OMP SIMD is used at the lower level in the physics parameterization codes to improve vectorization. This directive is applied to the innermost loop, the i-loop, which has no dependencies. In the case of WM6, as shown by Ouermi et al. [81], major code transformation was required in some cases to enable better vectorization.

3.2.2 Experimental Setup

The methodology used here follows Ouermi et al. [79], [81] to investigate various optimization strategies. This methodology consists of constructing standalone experiments to study the different approaches for parallelism in a more flexible and controlled environment. The findings from the standalone experiments inform the optimization decisions in the modules of interest, such as WSM6, GFS physics, and GFS radiation.

The experiments presented in this chapter use the Intel Knights Landing (KNL) [50] and Xeon CPU E-7-8890 (Haswell). As a reminder, the Intel Knights Landing (KNL) [50] architecture consists of 36 tiles interconnected with a 2D mesh, MCDRAM of 16GB high bandwidth memory on one socket. The KNL architecture has a clock frequency of 1.3 GHz, which is lower than the 2.5 GHz of Haswell. The Knights Landing tile is the basic unit that is replicated across the entire chip. This tile consists of two cores, each connected to two vector processing units (VPUs). Both cores share a 1 MB L2 cache. Two AVX-512 vector units process eight double-precision lanes each; a single core can execute two 512-bit vector multiply-add instructions per clock cycle. The Intel Xeon CPU E-7-8890 (Haswell) is composed of four sockets and four Non Uniform Memory Access (NUMA) nodes. Each node is made of 18 cores with 2 threads per core and clock frequency of 2.5 Ghz frequency.

3.3 Standalone Experiments

3.3.1 Synthetic Codes

These experiments analyze the thread-local SOA performance with different array sizes and dimensions in order to find a suitable structure for the physics schemes. The thread-local SOA in Code 1 use 1D arrays whereas those in Code 2 use 2D arrays. In Code 1, the k-loop is vectorized whereas in Code 2 the vectorization is along the i-loop. The access pattern is more involved in Code 1 compared to Code 2 because of the 1D versus 2D data layout. The performance results from the data transpose approach, as shown in Fig. 3.2, and the GFS and WSM6 codes take 2D ($im \times jm$) and 3D ($im \times jm \times km$) arrays where $im > 800$ and $jm < 40$. For a long rectangular data matrix $(im \times km)$ as shown in Fig. 3.2, thread parallelism across the k loop is limited by the number of iterations, i.e., km. In this case, $km <$ 40, which corresponds to less than 40 threads of the 256 threads on KNL. Transposing the data matrix from $im \times km$ to $km \times im$ allows for better thread parallelism while maintaining a good memory access pattern, as illustrated in Fig. 3.2. This transformation does not have an impact on computation correctness because both the standalone experiment codes and target physics codes have no dependencies along the horizontal direction (i-loop).

```
Code 1

!$OMP PARALLEL DEFAULT(shared)
!$OMP PRIVATE(its ,ite ,ice ,
              tsoa,thread_id,c)
!$OMP DO
do c=1,ite
  do j=1,je
    tsoa%a(j) = a(c,j)
    tsoa%b(j) = b(c,j)
    tsoa%d(j) = d(c,j)
    tsoa%e(j) = e(c,j)
  enddo
  call work(tsoa%a,tsoa%b,
            tsoa%d,tsoa%e,1,ice)
  do j=1,je
    a(c,j) = tsoa%a(j)
    b(c,j) = tsoa%b(j)
    d(c,j) = tsoa%d(j)
    e(c,j) = tsoa%e(j)
  enddo
enddo
!$OMP END DO
!$OMP END PARALLEL

subroutine work(a, b, c, d)
imlicit none
real,intent(inout):: a(:),b(:)
real,intent(inout):: c(:),d(:)
integer:: j
!$OMP SIMD
do j=2,je-1
  a(j) = 0.1+c(j)/d(j)
  b(j) = (0.2+c(j-1)-c(j))
      /(c(j)-c(j-1)+0.5)
enddo
end subroutine work
```

```
Code 2

!$OMP PARALLEL DEFAULT(shared)
!$OMP PRIVATE(its ,ite ,ice ,
              tsoa,thread_id,c)
!$OMP DO
do c=1,ite
  its = 1+ (c-1)*CHUNK
  ite = min(its+CHUNK-1, ie)
  ice = ite-its+1
  do j=1,je
    tsoa%a(1:ice,j) = a(its:ite,j)
    tsoa%b(1:ice,j) = b(its:ite,j)
    tsoa%d(1:ice,j) = d(its:tte,j)
    tsoa%e(1:ice,j) = e(its:ite,j)
  enddo
  call work(tsoa%a,tsoa%b,
        tsoa%d,tsoa%e,1,ice)
  do j=1,je
    a(its:ite,j) = tsoa%a(1:ice,j)
    b(its:ite,j) = tsoa%b(1:ice,j)
    d(its:ite,j) = tsoa%d(1:ice,j)
    e(its:ite,j) = tsoa%e(1:ice,j)
  enddo
enddo
!$OMP END DO
!$OMP END PARALLEL

subroutine work(a, b, c, d)
imlicit none
real, intent(inout):: a(:,:),b
    (:,:)
real, intent(inout):: c(:,:),d
    (:,:)
integer, intent(in):: is,ie
integer:: i,j
do j=2,je-1
  !$OMP SIMD
  do i=is,ie
    a(i,j) = 0.1+c(i,j)/d(i,j)
    b(i,j) = (0.2+c(i,j-1)-c(i,j))
            /(c(i,j)-c(i,j-1)+0.5)
  enddo
enddo
end subroutine work
```

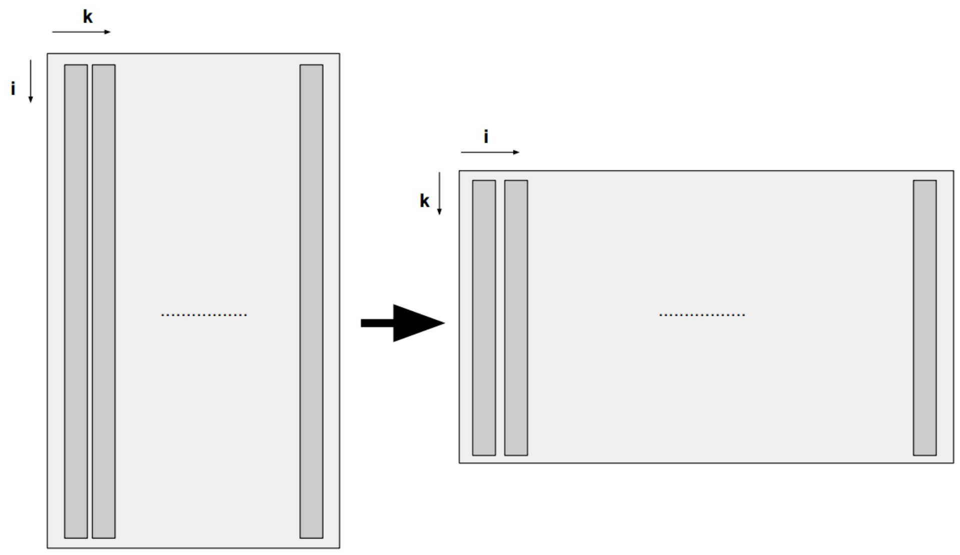
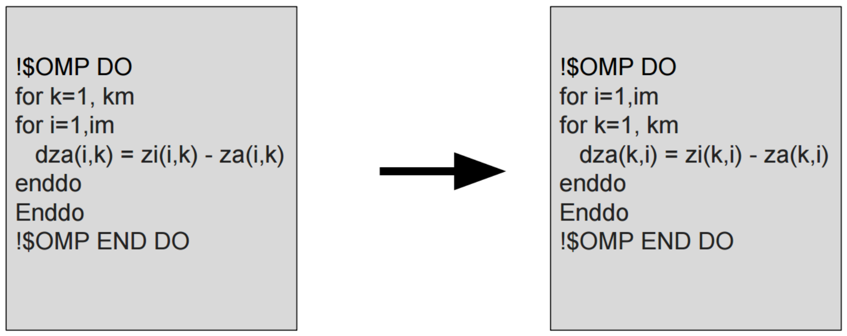


Fig. 3.2: Transpose representation that shows transposition of a 2D ($im \times jm$) array, where $im > 800$ and $jm < 40$. Given that Fortran is column major, k is the outer loop before the transposition, shown on the left. The outer loop becomes i after the transposition as shown on the right. This transformation increases thread parallelism on the outer loop.

Fig. 3.3 shows a code example of the transposition. Following the column major ordering in Fortran, the i-loop becomes the outer loop with $im = 10586$ after transformation. Furthermore, there are no dependencies along the i index, which allows parallelism in index i to be exploited. Table 3.1 shows performance results from SOA with 1D arrays, transposed data matrices, and unmodified original data. The SOA approach yields significant speed-ups with a maximum of about 34. The data transpose approach performs the best in this particular experiment, with a maximum speed-up of about 41. The length of the arrays in the SOA is 48. This small array length translates to a small amount of work for the innermost



```
!$OMP DO
for k=1, km
for i=1,im
   dza(i,k) = zi(i,k) - za(i,k)
enddo
Enddo
!$OMP END DO
```

```
!$OMP DO
for i=1,im
for k=1, km
   dza(k,i) = zi(k,i) - za(k,i)
enddo
Enddo
!$OMP END DO
```

Fig. 3.3: Code transformation with transpose. This shows how the transposition is implemented with a simple code. The loops and the indices are swapped.

loop in Code 1. In this experiment, the peak performance is observed at 128 threads with two threads per cores. In hyper-threading, each core resources are shared between the hyper-threads. The instructions from hyper-threads flow through the same pipeline, which can help improve core utilization as observed in Table 3.1. However, sharing resources between hyper-threads may lead to performance decrease, as observed with 256 threads. In addition, after 128 threads the work is not enough to enable further improvement in speed-up.

Table 3.2 shows performance results similar to those in Table 3.1 with an increased problem size given by ke = 768. The arrays in the SOA are 16 times larger that those used in previous experiments. In both cases, these results indicate that the transpose approach for data organization yields better results. After 64 threads, each core uses hyper-threading, with two to four threads per core, which a given core, divides the resources between the hyper-threads causing the performance to decrease.

Table 3.3 shows the performance results from using thread-local SOA with 2D arrays, transposed data matrices, and unmodified original data. In this experiment, the OpenMP chunk size is set to 8. In contrast to the previous experiments, these results show that the thread-local SOA approach yields higher speed-ups than the other methods for data organization. The maximum speed-up observed is 103 at 32 cores. After 32 threads, there is not enough work per thread to enable performance scalability.

The results from Table 3.1 – 3.3 indicate that the size and structure of the arrays in the thread-local SOA play an important role in the performance. Vectorizing along the k-loop, in the 1D case, has a more involved access pattern than vectorizing along the i-loop, in the 2D case. In addition, there are no dependencies along the i-loop, which allows for trivial vectorization. Furthermore, the L2 cache is about 16 times the size of the input data in each SOA. Thus the thread-local SOA fit in the L2 cache, which allows for fast memory access. When the thread-local SOA does not fit in the L2 cache, as shown in Table 3.4, the speed-ups are significantly lower than the ones observed in Table 3.3. In Table 3.4, the peak performance for thread-local SOA is observed at about 16 threads. This is lower than the previous cases because of the high rate of cache misses.

Table 3.1: Results from Code 1 compared to transpose approach and original code. The maximum speed-ups for transpose and thread-local SOA are at 128 threads. At 128 threads, each core uses two hyper-threads per core. The hyper-threads share the same instruction pipeline, which helps improve utilization of cores. Sharing resources can contribute to reducing core utilization, as observed at 256 threads. Hyper-threading performance is dependent on how the shared resources are managed between hyper-threads.

Threads	Time (ms)			Speed-up		
	Orig.	Transp.	SOA	Orig.	Transp.	SOA
1	2.06	3.3 6	3.33	1	0.61	0.62
2	1.59	1.97	1.74	1.30	1.05	1.18
4	0.91	1.44	0.84	2.26	1.43	2.45
8	0.67	0.5	0.41	3.07	4.12	5.02
16	0.55	0.26	0.18	3.75	7.92	11.44
32	0.54	0.17	0.15	3.81	12.12	13.73
64	0.72	0.05	0.11	2.86	41.20	18.73
128	0.87	0.05	0.06	2.37	41.20	34.33
256	1.35	0.1	0.49	1.53	20.60	4.20

Table 3.2: Results from Code 1 compared to transpose approach and original code with large array sizes. In this experiment, the maximum performance occurs at 64 threads. After 64 threads, hyper-threading is used and the resource per core is divide up between the hyper-threads. This causes the performance to slow down.

Threads	Time (ms)			Speed-up		
	Orig.	Transp.	SOA	Orig.	Transp.	SOA
1	33.82	29.53	75.45	1.00	1.15	0.45
2	26.98	19.44	45.7	1.25	1.74	0.74
4	15.54	13.47	23.37	2.18	2.51	1.45
8	10.9	5.09	7.44	3.10	6.64	4.55
16	8.86	2.98	5.96	3.82	11.35	5.67
32	8.93	2.61	1.72	3.79	12.96	19.66
64	10.97	0.95	1.39	3.08	35.60	24.33
128	16.14	1.17	5.93	2.10	28.91	5.70
256	22.27	2.17	9.57	1.52	15.59	3.53

Table 3.4: Results from Code 2 compared to transpose approach and original code with large arrays. The peak performance is observed at 16 threads. In this case, the array thread-local SOA do not fit in L2 cache. This leads to lower performance than the cases where the thread-local SOA fit in cache.

Threads	Time (ms)			Speed-up		
	Orig.	Transp.	SOA	Orig.	Transp.	SOA
1	264.71	194.94	159.98	1.00	1.36	1.65
2	119.93	120.69	113.15	2.21	2.19	2.34
4	98.89	61.57	57.08	2.68	4.30	4.64
8	54.17	25.57	34.25	4.89	10.35	7.73
16	30.11	16.3	22.83	8.79	16.24	11.59
32	16.87	13.51	34.23	15.69	19.59	7.73
64	13.81	13.15	29.72	19.17	20.13	8.91
128	15.74	6.56	38.25	16.82	40.35	6.92
256	23.33	13.24	45.51	11.35	19.99	5.82

Table 3.3: Results from Code 2 compared to transpose approach and original code. The best performance is observed at 64 threads for the thread-local SOA. At 128 and 256 threads, each core uses about two and four threads per core. The core resources, such as L1 cache, are shared between the hyper-threads, which causes the performance to slow down for large core counts.

Threads	Time (ms)			Speed-up		
	Orig.	Transp.	SOA	Orig.	Transp.	SOA
1	2.06	3.36	1.99	1.00	0.61	1.04
2	1.59	1.97	1.07	1.30	1.05	1.93
4	0.91	1.44	0.53	2.26	1.43	3.89
8	0.67	0.5	0.14	3.07	4.12	14.71
16	0.55	0.26	0.07	3.75	7.92	29.43
32	0.54	0.17	0.02	3.81	12.12	103.00
64	0.72	0.05	0.06	2.86	41.20	34.33
128	0.87	0.05	0.27	2.37	41.20	7.63
256	1.35	0.1	0.04	1.53	20.60	51.50

Fig. 3.4 shows the performance results from choosing different lengths for the index i. All the chunk sizes considered yield higher speed-ups than using transpose approach. The best performance is observed when using a chunk size of 32. The chunk size of 32 provides enough work to make better use of the SIMD units. The choice of the chunk size is application dependent.

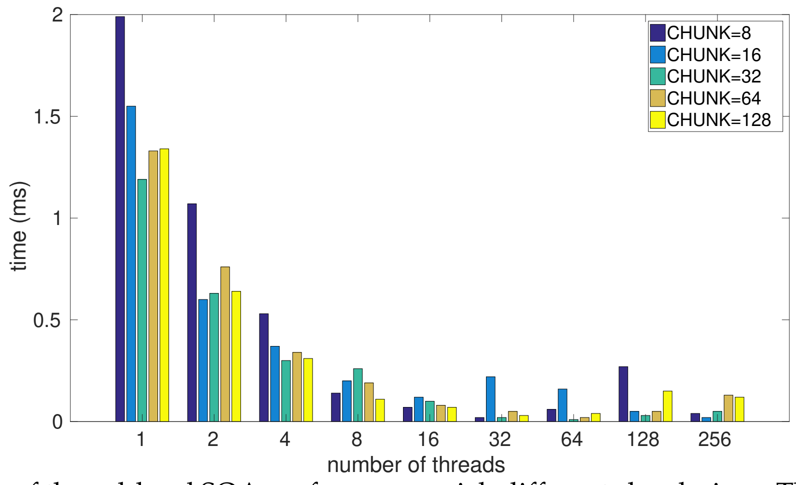


Fig. 3.4: Plots of thread-local SOA performance wieh different chunk sizes. The bars indicate the runtime of the optimized standalone Code 2 with different size thread-local SOA which determined by the choice of chunk. The lowest runtime occurs at 64 with $chunk = 32$. This indicates that $chunk = 32$ provide enough work per thread and one thread per core enables a better usage of core resources compare to two and four threads per core.

3.3.2 Rain Routines

The WSM6 module contains semi-Lagrangian routines [52], nisflv_rain_plm6, and nisflv_rain_plm6 for simulating falling hydrometeors. These semi-Lagrangian routines, an alternative to a traditional Eulerian scheme, use forward advection to calculate the path of the falling hydrometeors. Initially, nisflv_rain_plm6, and nisflv_rain_plm6 used Fortran keywords cycle, goto, and exit. With these keywords, the termination criteria are not known a priori, which prevents parallelism. This limitation was resolved by substituting the keywords with carefully engineered logic that performs the same computation. The exits were replaced by masking, the gotos by loops coupled with conditionals and cycle by conditionals. After removing these serial bottlenecks, the thread-local SOA and transpose approaches from Code 2 are applied to the rain routines. As in Code 2, the rain routines have no dependencies along the i-loop, and the thread-local SOA with a chunk size of 32 now fits into the L2 cache. The results in Fig. 3.5 and Table 3.5 for the optimized rain routine with $chunk = 32$ demonstrate that using thread-local SOA produces larger speed-ups than transposing the input data. In this case, the thread-local SOA are chosen to fit in cache and designed for contiguous memory access to improve performance. In contrast, transposing the input data increases parallelism at the thread level but does not improve memory performance. The optimized, thread-local SOA, version of the rain routine runs 50 times faster than the original serial version, and two faster than the transpose version.

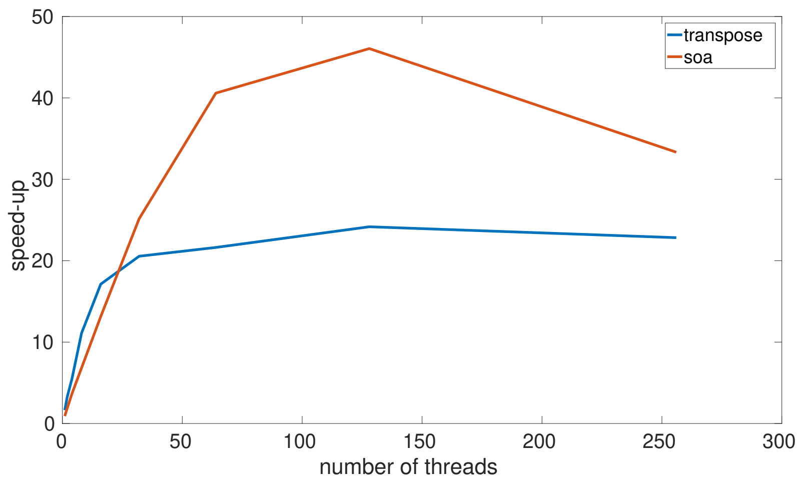


Fig. 3.5: Transpose vs SOA speed-ups on nisflv_rain_plm6. This thread scalability plot reaffirms that using thread-local SOA scales better than transposing the input data.

Table 3.5: Thread-local SOA and Transpose approach applied to nisfl_rain_plm6. This shows runtimes of transpose and thread-local SOA on a subroutine in WSM6.

Threads	Transpose (ms)	SOA (ms)
1	250	450
2	127	220
4	74	112
8	37	60
16	24	31.2
32	20	16.3
64	19	10.1
128	17	8.9
256	18	12.3

3.4 Optimization Results with WSM6 and GFS-Physics

3.4.1 WSM6

In addition to the transformations in nisflv_rain_plm6 and nisflv_rain_plm6 routines, the OMP SIMD directive is applied at the lower level to the innermost loops instead of relying on the Intel compiler auto vectorization. Thread parallelism is implemented at the parent level in the WSM6 module. The bar plots in Fig. 3.6 and 3.7 do not show significant differences in runtime for various thread-local SOA sizes on KNL and Haswell. These figures indicate that the different chunk sizes used in this experiment achieve about the same performance improvement with the best speed-up of about 26 when using static scheduling.

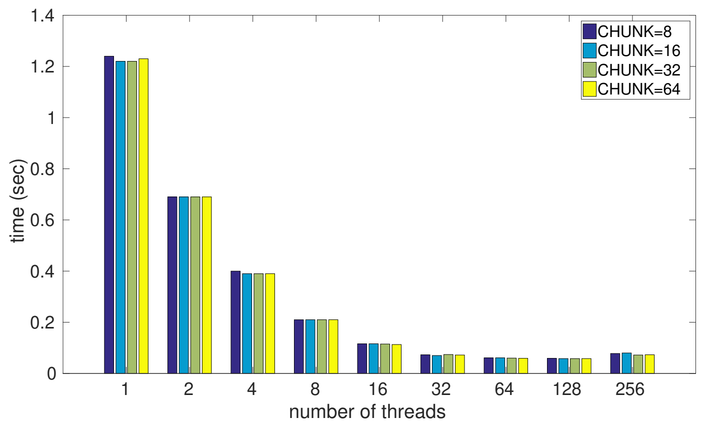


Fig. 3.6: WSM6 runtime with various thread-local SOA sizes and static scheduling on KNL. The bars shows that the runtimes decreases exponentially as the number of threads increases regardless of the chunk sizes. Each chunk provides enough work for thread and vector parallelism. The lowest runtime occurs at 64 threads and plateaus after that.

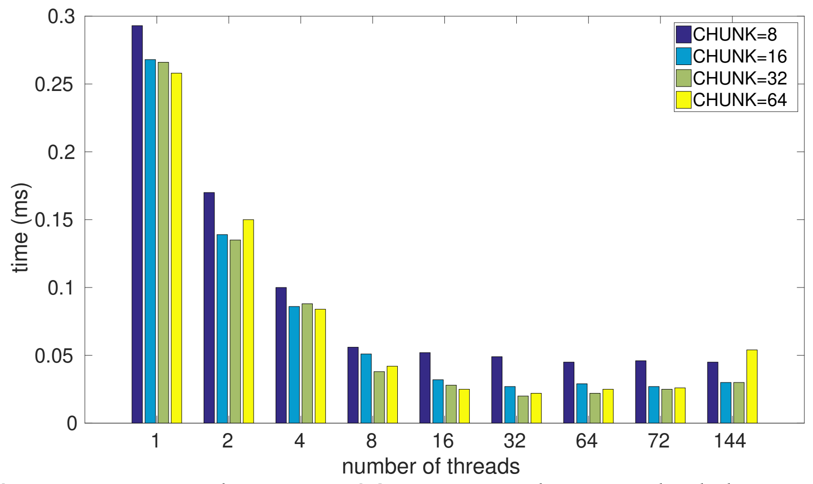


Fig. 3.7: WSM6 runtime with various SOA sizes and static scheduling on Haswell. The bars show that the runtimes decrease exponentially as the number of thread increase up to about 32 threads. The best runtime is observed at 64 threads with $chunk = 32$. The performance plateaus after the 64 threads. This indicates that hyper-threading does not improve performance in WSM6. In addition, after 64 threads the amount of work per thread is not large enough to enable scalability.

Fig. 3.8 and 3.9 compare static versus dynamic scheduling performance on KNL and Haswell, respectively. In both systems, dynamic scheduling, performs better than static scheduling. The dynamic scheduler helps load balance the work between the threads. Because of the conditionals and complexity within physics routines, the work distributed between the threads may be unbalanced, causing some threads to run longer than necessary. With dynamic scheduling, an internal work queue is used to give a block of iterations to each thread. When a thread finishes its current task, it retrieves the next ready block for the top of the queue. This helps reduce the wait time observed in the static scheduling case. In the case of KNL, the performance can be further improved by enabling the flat configuration. In the flat configuration, the high band width memory (HBM) MCDRAM is used as a physical address instead of cache. This flat configuration in WSM6 makes better use of the HBM compare to cache configuration. Fig. 3.10 compares flat versus cache performance on KNL. The flat KNL configuration provides better performance than the cache configuration by a factor of 1.6. Overall, the optimized version of WSM6 runs 70 times faster and 26 times faster on KNL and Haswell, respectively. Haswell performs better than KNL by a factor of by a factor of 1.3. In addition, the optimized version of WSM6 on KNL runs 23.3 faster than the optimized version presented in [79].

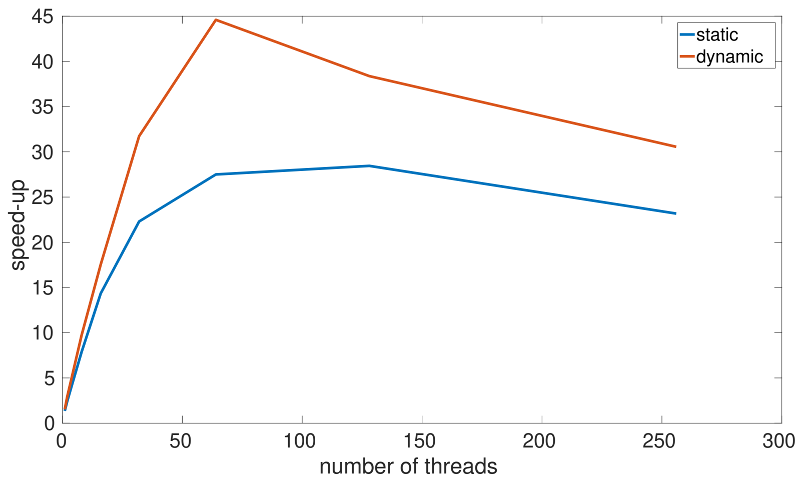


Fig. 3.8: WSM6 speed-ups on KNL. This shows scalability plots of WSM6 with static and dynamic scheduling. The chunk size is chosen to be 32 in this case. The performance scales up to 64 threads and then decreases. Hyper-threading is used at 128 and 256 threads. In hyper-threading, the core resources are divided between hyper-threads, and this may limit the performance, as seen in this case.

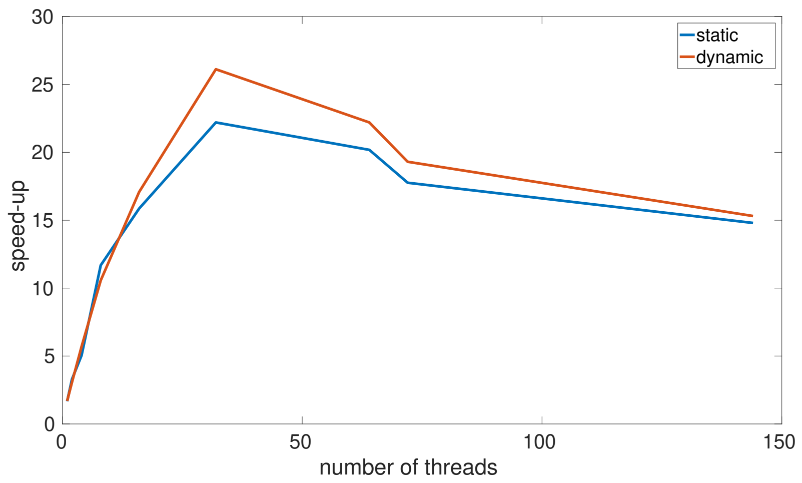


Fig. 3.9: WSM6 speed-ups with static and dynamic scheduling on Haswell. The best performance occurs at 32 threads. After 16 threads, performance is limited by the NUMA affect because OpenMP is not suitable for parallelism across NUMA nodes. The performance could be improved by using MPI.

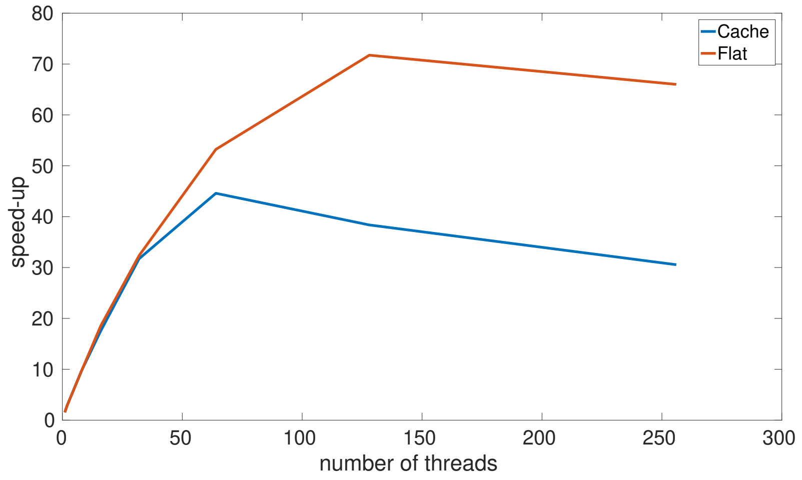


Fig. 3.10: WSM6 speed-ups on KNL with flat configuration. In this scalability plot of WSM6 with cache and flat configuration, dynamic scheduling is used for both and the chunk size is set to 32. The maximum speed-up is observed at 64 threads in the case of the cache configuration and 128 threads in the case flat configuration. In the case of the flat mode, hyper-threading with two hyper-threads per core improved performance. Although the resources per core are shared between hyper-threads, in this case the set instructions in the shared instruction pipeline enable a better utilization of core resources.

Table 3.6: SOA approach applied to WSM6 with flat and cache modes.

Threads	cache (ms)	flat (ms)
1	1079.3	1084.32
2	570.51	574.92
4	325.86	324.91
8	171.67	167.61
16	93.3	90.32
32	53.66	50.21
64	35.4	31.66
128	45.39	23.45
256	65.59	24.2

3.4.2 GFS Physics Results

GFS physics does not have many serial bottlenecks that require major code transformations as in the case of WSM6 with niflv_rain_plm6 and niflv_rain_plm. Thread parallelism is applied at a high level in the GFS driver using thread-local SOA. OMP SIMD directives are instrumented at a lower level, the innermost loops, to enable better vectorization. In addition, static allocation is used for the thread-local SOA instead of dynamic allocation as in the original input data.

Fig. 3.11 and 3.12 show runtime performance on KNL and Haswell, respectively, with different chunk sizes. In the case of KNL, the runtime decreases exponentially, indicating good scalability. As shown in Fig. 3.12, the runtime decreases up to about 16 threads. The first 16 threads are running in one NUMA node. After 16 threads, more NUMA nodes are used. Shared memory parallelism is not suitable for parallelism across NUMA nodes. This limitation is addressed by using four MPI ranks, one for each node. Fig. 3.13 indicates that coupling the four MPI ranks with shared memory parallelism led to significant improvement on runtimes past the 16 threads.

Fig. 3.14–3.16 compare static and dynamic scheduling scalability. In Fig. 3.15, the speed-up increases up to 16 threads and decreases rapidly after the 16 threads because of difficulties shared memory parallelism across the NUMA nodes. In both Fig. 3.14 and 3.16 static, scheduling performs better than dynamic scheduling. The work load between threads is sufficiently balanced that using a dynamic scheduler does not yield any improvement. In the case of KNL, the flat configuration improves the speed-up by a factor of 1.04 compared to the cache configuration. The optimized version of GFS physics runs about 2.4 times faster on Haswell compared to KNL, which corresponds to speed-ups of 27 and 18 on KNL and Haswell, respectively, over the original serial versions.

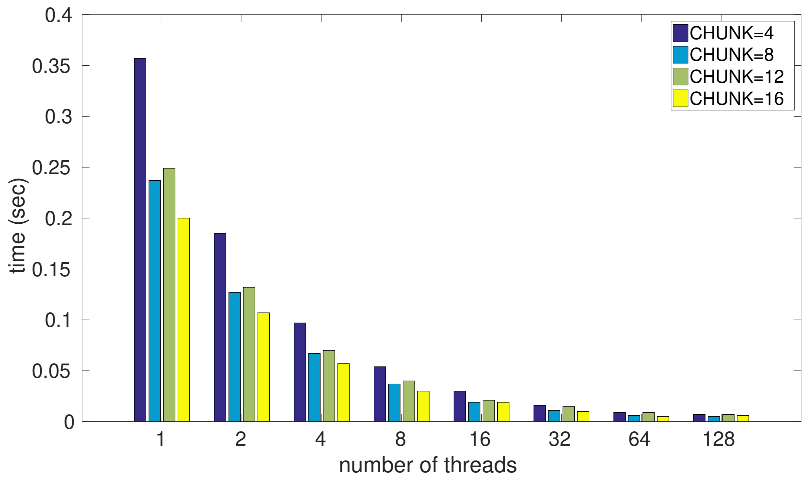


Fig. 3.11: GFS physics runtime with various thread-local SOA sizes on KNL. This plot shows the runtimes of different thread-local SOA to help guide the choice of chunk size. Static scheduling is used in this experiment. The maximum speed-up occurs at 128 threads with *chunk* $= 8$. The maximum number of loop iterations is 108. Thus, using 256 threads is largely more than necessary given there are only 108 loop iterations.

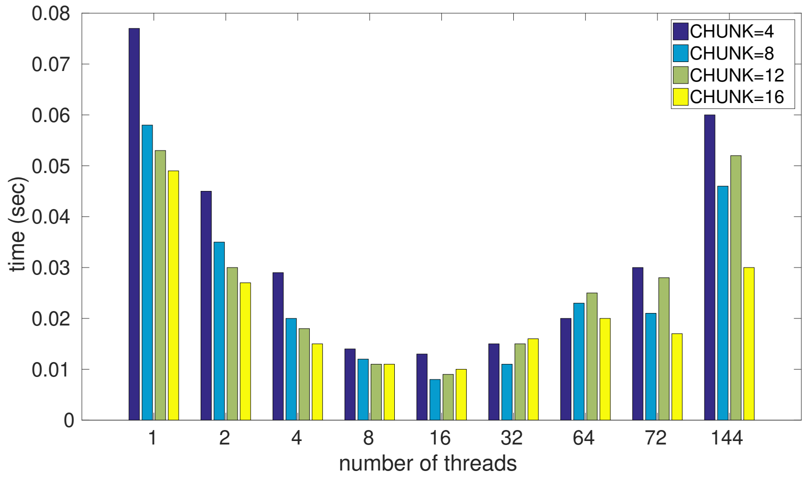


Fig. 3.12: GFS physics runtime with various SOA sizes on Haswell. This plot shows runtimes of different thread-local SOA to help determine the appropriate choice for the chunk size. MPI was not used. OpenMP is used for shared parallelism across NUMA nodes. The default static scheduler is used in this experiment. The lowest runtime occurs a 16 threads. After 16 threads, the NUMA effect start limiting performance. This can be addressed by using MPI for parallelism across NUMA nodes.

3.4.3 GFS radiation

As in GFS physics, GFS radiation is optimized at a high-level with thread-local SOA to improve thread parallelism and at a low-level with OMP SIMD to improve utilization

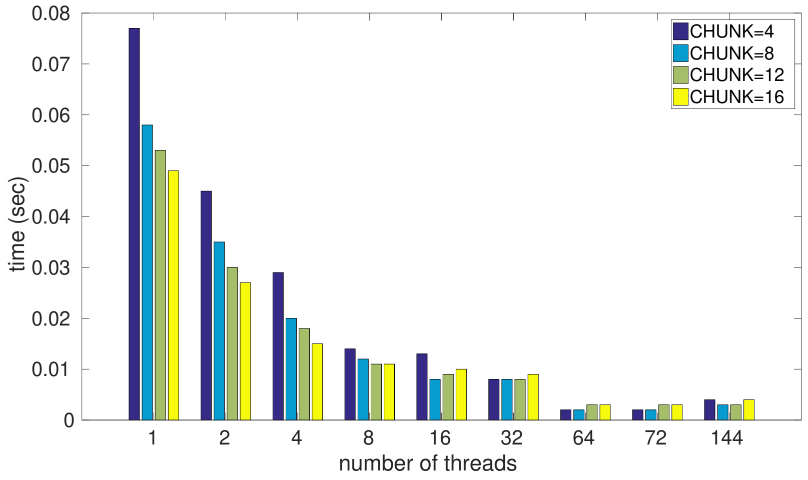


Fig. 3.13: GFS physics runtime with various SOA sizes on Haswell with MPI. This plots shows runtimes of different thread-local SOA to help choose on the appropriate chose for the chunk size. MPI was used for parallelism across NUMA nodes and OpenMP for shared parallelism within NUMA nodes. The default static scheduler is used in this experiment. The performance scales up to 72 threads. Hyper-threading does not help improve speed-ups.

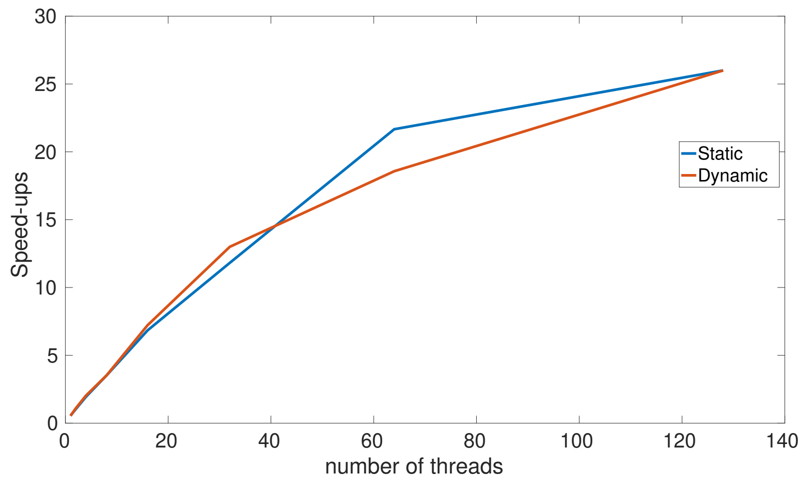


Fig. 3.14: GFS physics speed-ups on KNL. These plots show thread scalability performance with static and dynamic scheduling on KNL. The performance scales up to 128 threads. The uses a maximum of 128 threads because there is 108 iteration. Using 256 threads would be oversubscribing. In this case, hyper-threading enables better performance.

of SIMD units. Static instead of dynamic allocation is used as well to improve memory accesses. Similarly to WSM6 and GFS physics, the bar plots in Fig. 3.18 - 3.20 are used to determine the appropriate thread-local SOA size to choose for optimization. Fig. 3.19 indicates the limitations of using OpenMP for parallelism across NUMA nodes. Fig. 3.18 and 3.20 show that the chunk sizes of 8 and 16 yield the lowest runtime on both KNL and

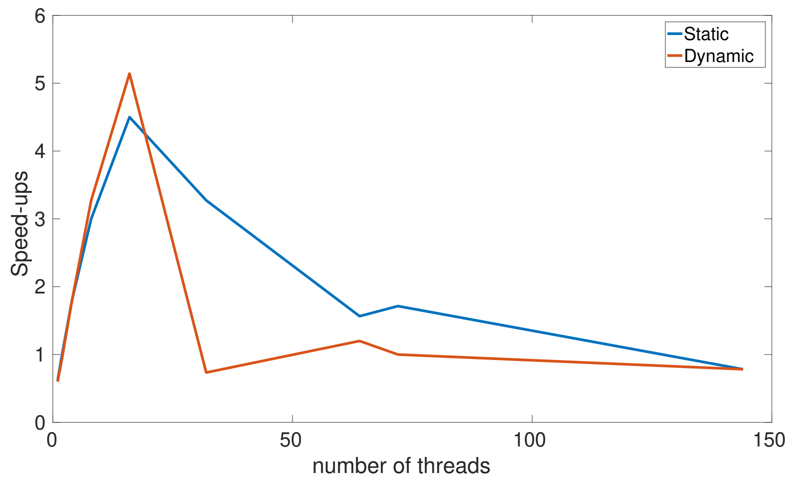


Fig. 3.15: GFS physics speed-ups on Haswell. These plots show thread scalability performance with static and dynamic scheduling on Haswell. OpenMP is used for parallelism within and across NUMA nodes. The performance decreases after 16 threads because of NUMA effects. Using OpenMP fr parallelism across NUMA nodes does not improve speed-ups.

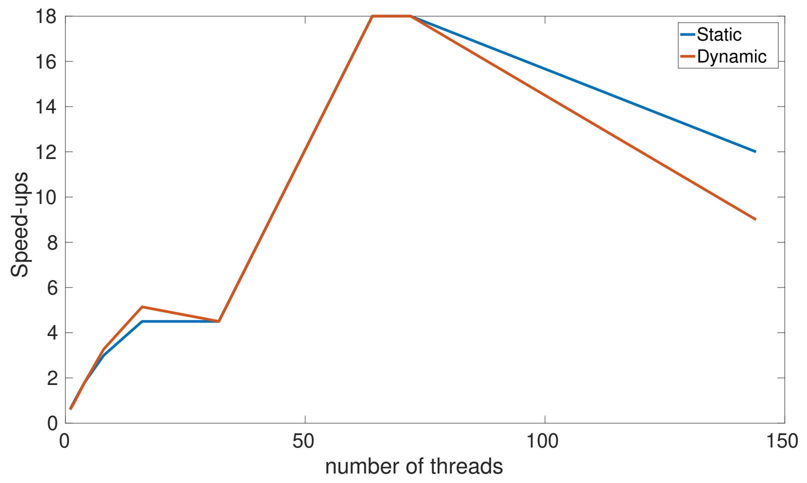


Fig. 3.16: GFS physics speed-ups on Haswell with MPI across nodes. These plots show thread scalability performance with static and dynamic scheduling on Haswell. MPI and OpenMP are used for parallelism across and within NUMA nodes, respectively. This optimization scales up to 72 cores. Hyper-threading does not improve the utilization of core resources.

Haswell.

Fig. 3.21 – 3.23 compare static and dynamic performance on KNL and Haswell. Similarly to the previous case with GFS codes, not using MPI for parallelism across NUMA nodes does not scale as shown in Fig. 3.22. On both KNL and Haswell, dynamic scheduling performs better than static scheduling. The use of dynamically assigned work loads to threads reduces the threads wait time compared to statically distributing work between the

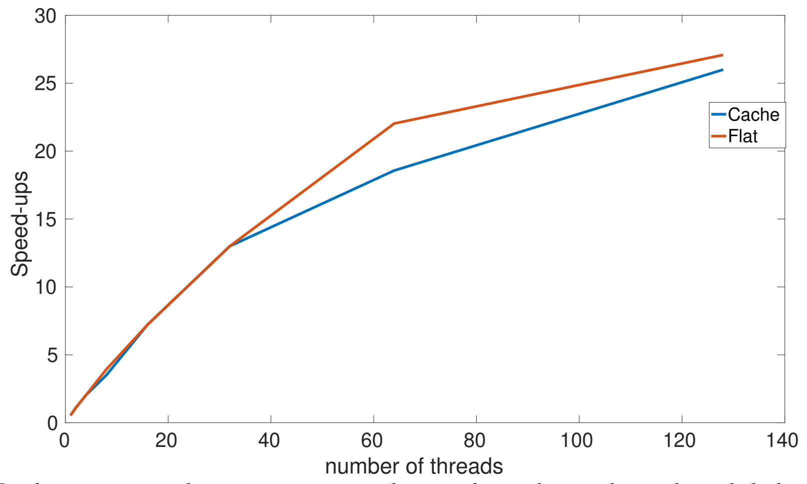


Fig. 3.17: GFS physics speed-ups on KNL. These plots show thread scalability performance with flat and cache configurations on KNL. Dynamic scheduling is used. Both cache and flat configurations scale up to 128 threads.

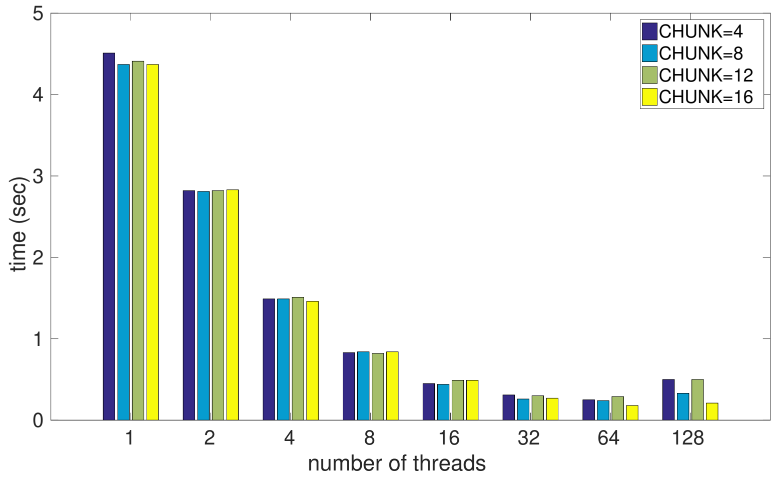


Fig. 3.18: GFS radiation runtimes with various SOA sizes on KNL. This plot shows runtimes of different thread-local SOA to help determine the appropriate choice for the chunk size. Static scheduling is used in this experiment. The best speed-up is observed at 64 threads. This indicates that one thread per core allows for better utilization of core resources compared to two threads per core.

threads. Further performance improvement is observed when using the flat configuration in the case of KNL by a factor of 1.05, as shown in Fig. 3.24.

The optimized version of GFS physics runs 23 and 30 times faster on KNL and Haswell, respectively, over the serial times on KNL and Haswell. The runtime on Haswell is 6.5

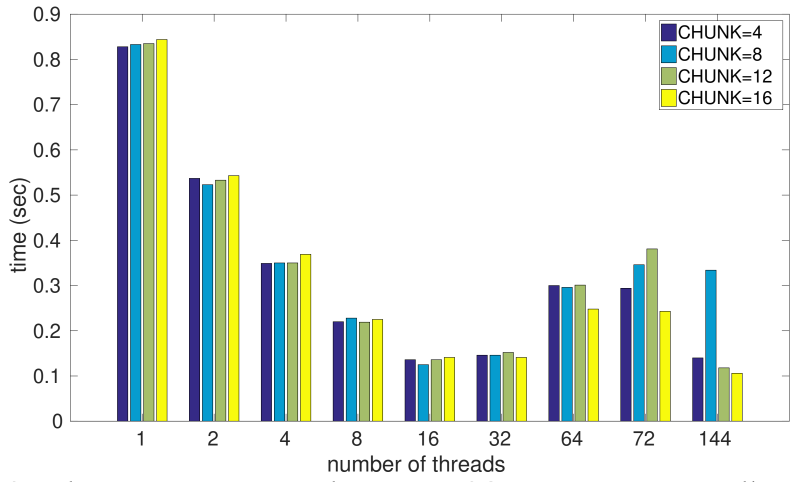


Fig. 3.19: GFS radiation runtimes with various SOA sizes on Haswell. This plot shows runtimes of different thread-local SOA to help determine the appropriate choice for the chunk size. MPI was not used. OpenMP is used for shared parallelism across NUMA nodes. The default static scheduler is used in this experiment. The best performance is observed at 16 threads. After 16 threads, using OpenMP for parallelism across NUMA nodes limits performance. OpenMP is designed for shared memory parallelism.

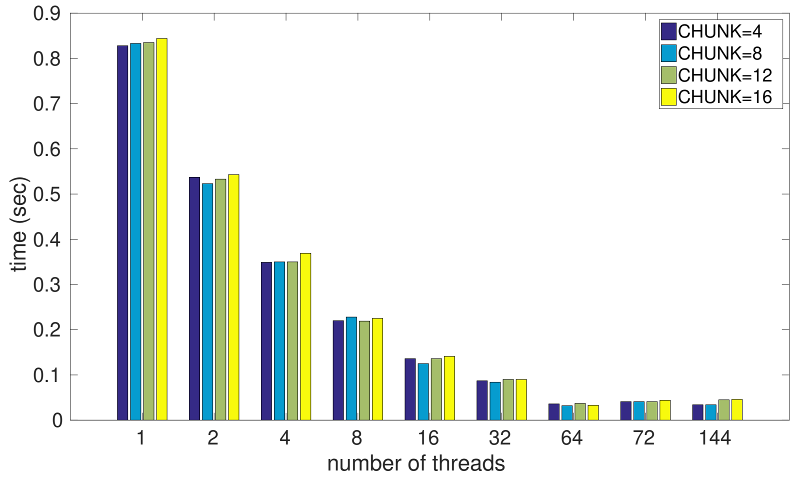


Fig. 3.20: GFS radiation runtimes with various SOA sizes on Haswell. This plot shows runtimes of different thread-local SOA to help determine the appropriate choice for the chunk size. MPI was used for parallelism across NUMA nodes and OpenMP for shared parallelism within NUMA nodes. The default static scheduler is used in this experiment. This experiment scales up to 64 threads. Using hyper-threads does not improve performance.

faster than the runtime on KNL.

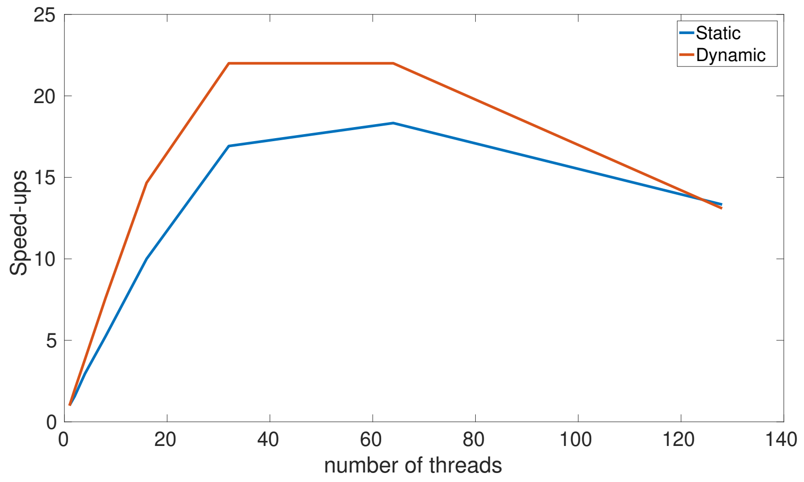


Fig. 3.21: GFS radiation speed-ups on KNL. These plots show thread scalability performance with static and dynamic scheduling on KNL. The maximum performance is observed at about 64 threads. The performance does not change much between 32 and 64 threads because there is not enough work per thread to improve scalability. After 64 threads, the performance decreases because the hyper-threading does not help increase performance.

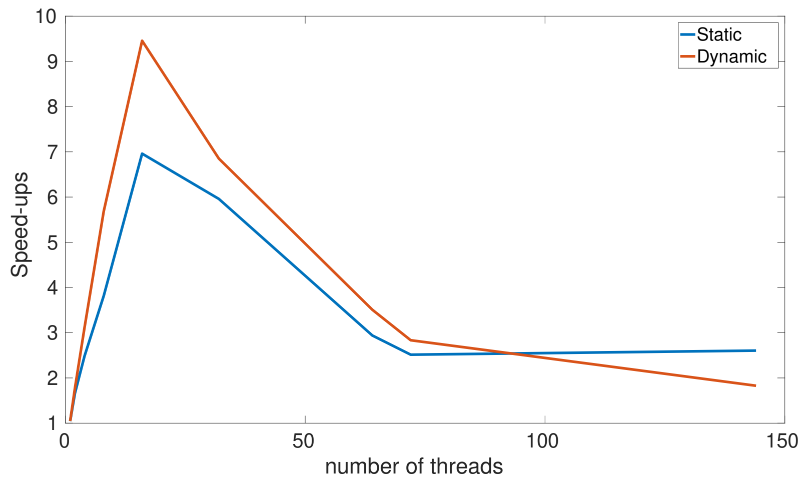


Fig. 3.22: GFS radiation speed-ups on Haswell. These plots show thread scalability performance with static and dynamic scheduling on Haswell. OpenMP is used for parallelism within and across NUMA nodes. The best performance is observed at 16 threads. After 16 threads, the performance decreases because OpenMP is not suitable for parallelism across NUMA nodes.

3.5 Summary and Discussion

The results from the standalone experiments in Section 3.3 demonstrate that the thread-local SOA approach is suitable for optimizing the physics schemes in NEPTUNE. These standalone experiments are instrumental in identifying the modifications necessary to

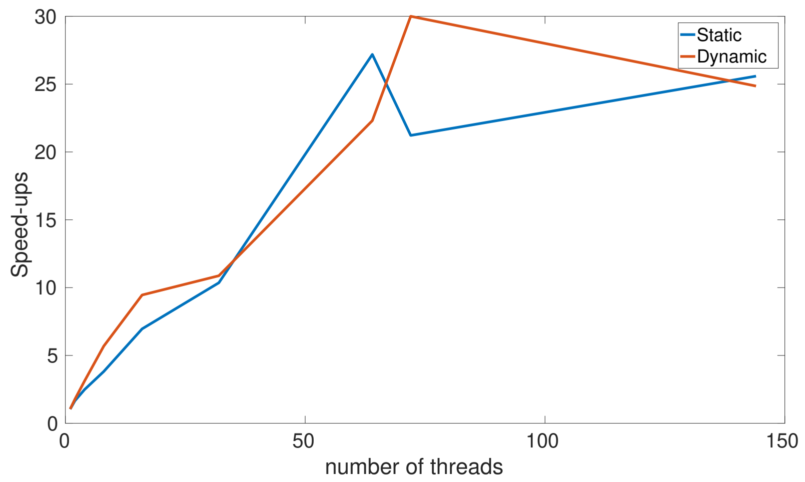


Fig. 3.23: GFS radiation speed-ups on Haswell with MPI across nodes. These plots show thread scalability performance with static and dynamic scheduling on Haswell. MPI and OpenMP are used for parallelism across and within NUMA nodes, respectively. This optimized code scales up to 72 threads. After the 72 threads the performance decreases. This indicates that one thread per core enables a better utilization of the core's resources than two threads per core.

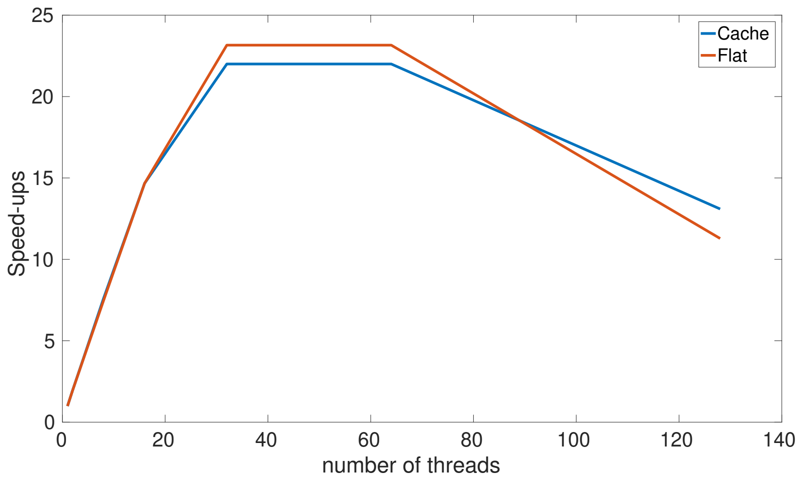


Fig. 3.24: GFS radiation speed-ups on KNL. These plots show thread scalability performance with flat and cache configurations on KNL. Both flat and cache configurations scale up to 64 threads. The performance decreases after 64 threads because hyper-threads do not improve utilization of core's resources. Because the test case fit in MCDRAM, the flat configuration enables a slightly better memory usage, which is translated into better performance.

optimize WSM6, GFS physics and GFS radiation on KNL and Haswell. This study exploits the flexibility and simplicity of the standalone experiments to prototype and test the different optimization strategies, which are not easily and trivially testable in NEPTUNE.

The transformation of the input and output data into thread-local SOA is the main approach used in optimizing the WSM6 and GFS codes. The size of the thread-local SOA is chosen to fit in the L2 cache. Each thread-local SOA is composed of the inputs and outputs required to calculate the physics for a few columns. This data transformation reduces memory traffic and increase data locality. In the transpose approach, the data might be far apart in memory and too large to fit in the L2 cache. This causes cache misses, which limits performance. In addition, applying the transpose to the entire physics routines requires significant code modification compared to the thread-local SOA approach.

The OpenMP directive OMP SIMD is used to improve vector parallelism at the low level. In cases similar to the rain routines, significant low-level code modifications are required to enable vectorization. Given that there are dependencies along the vertical direction, the OMP SIMD directive is applied along the horizontal direction (*i* loop). In the thread-local SOA, the i-loop corresponding to the chunk size is chosen to be a multiple of the SIMD unit length.

The original serial version of WSM6, GFS physics, and GFS physics ran for 1.65 sec, 0.130

sec, 4.40 sec on KNL and 0.444 sec, 0.036 sec, and 0.870 sec on Haswell. These runtimes are about 3.7, 3.6, and 5.05 times faster on Haswell compared to KNL for serial codes. The original codes rely on auto vectorization, which does not work well with a complex and large body of code. Haswell has lower runtimes because it has a higher clock frequency and a turbo boost. Table 3.7 shows a summary of performance improvement from original codes to optimized thread-local SOA on both KNL and Haswell.

The optimized version of WSM6 yields a speed-up of 70 and 26 on KNL and Haswell, respectively, over the serial times. In the case of Haswell, the maximum performance is observed at about 32 cores compared to 64 cores on KNL. Haswell performs better than KNL because it has higher clock speed and transactional synchronization extensions (TSX-NI) technology to improve threading.

The performance of WSM6 on Haswell could be improved by designing the code to run with 4 MPI ranks for parallelism across NUMA nodes. On KNL it could be further improved by better using the SIMD units.

The optimized version of GFS physics runs about 2.4 faster on Haswell compared to KNL. GFS physics scales up to 72 cores on Haswell and 64 cores on KNL. The large SIMD units on KNL are not sufficient to outperform Haswell, which has more cores and a higher clock frequency than KNL. After optimization, GFS physics runs 27 and 18 times faster on KNL and Haswell, respectively, over the serial times.

The optimized GFS radiation runs 23 times faster on KNL and 30 times faster on Haswell with respect to their serial times. In this case, Haswell performs about 6.5 better than KNL. As in the GFS physics optimization, the GFS radiation scales up to 64 cores on KNL and 72

Table 3.7: Performance summary. This table shows the best performance results for the different physics codes used on KNL and Haswell. On KNL the chunk size is set to 8 and on Haswell it is set 32.

physics schemes		WSM6	GFS physics	GFS radiation
KNL	best time (ms)	23.0	4.8	190.0
	speed-up	70	27	23
	threads	64	128	64
	configuration	dynamic+flat	static+flat	dynamic+flat
Haswell	best time (ms)	17.0	2.0	29.0
	speed-up	26	18	30
	threads	32	72	72
	configuration	dynamic	static	dynamic

cores on Haswell.

The test cases used in this study have about $10K$ iterations for WSM6 and 800 iterations for GFS codes. These test cases are not large enough to provide sufficient work to each thread and scale well to 64 cores on KNL and 72 cores on Haswell.

With regard to peak performance, some of the challenges faced by physics codes are illustrated by Code 2 in Section 5. In this case, there are only nine flops in the inner loop. This is typical of some of the loops in WSM6. As a result, with array dimensions of 10592 and 39, there are only 3.7M flops. A loop time of 0.02ms gives a flop rate of 185 GFLOPs, which is about 6.6% of peak and is not unexpected for loops that have low flop counts.

All the tables and plots show a performance decrease after 128 threads for KNL and 72 threads for Haswell. This corresponds to two or four thread per core. In the KNL and Haswell, all active threads in a given core flow through the same pipeline, and thus they share resources such as instruction cache and instruction queue. The increase in the number of threads per cores leads to the division of the shared resources among threads, and to an increase in memory access conflicts. This competition for resources indicates why a performance decrease is observed after 128 threads and 72 threads on KNL and Haswell, respectively.

This work has demonstrated the efficiency of high-level optimization approach using thread-local SOA paired with low-level optimization technique using OMP SIMD directive. As presented in the results section, these optimization approaches enable a better utilization of the KNL and Haswell resources by improving locality, memory allocation, and vectorization. The use of thread-local SOA and static allocation enables better memory traffic by increasing locality and decreasing cache misses. The use OMP SIMD directives coupled with SOA chunk sizes, set to be multiples of the SIMD length, enables a better utilization of SIMD units in KNL and Haswell. Overall, the various optimizations achieved a speed-ups of 70, 27, 23 on KNL, and 26, 18, and 30 on Haswell over the original serial version of WSM6, GFS physics, GFS Radiation, respectively. In addition, the results indicated that WSM6, GFS physics, GFS radiation run 1.3, 2.4 and 6.5 faster on on Haswell compared to KNL because the Haswell system used here has more cores and a higher clock frequency than KNL. As mentioned in the discussion, peak performance is still relatively challenging to achieve given the complexity of the physics schemes.

CHAPTER 4

HIGH-ORDER DATA-BOUNDED AND POSITIVITY-PRESERVING INTERPOLATION

4.1 Introduction

A number of key scientific computing applications that are based upon high-order methods over tensor-product grid constructions, such as numerical weather prediction (NWP) and combustion simulations, require *property-preserving* interpolation. In the aforementioned application areas, property preservation often manifests itself as a requirement for either data boundedness or positivity preservation. The particular application motivating this work is the Navy Environmental Prediction System Utilizing a Nonhydrostatic Engine (NEPTUNE). NEPTUNE is a next-generation global NWP system being developed at the Naval Research Laboratory (NRL) and the Naval Postgraduate School (NPS) [55]. NEPTUNE makes use of the Nonhydrostatic Unified Model of the Atmosphere (NUMA) [34] three-dimensional spectral element dynamical core, but currently uses physics routines that were developed assuming uniform grid spacing on the elements. At least two options are available for combining these two NWP building blocks: either (1) evaluate the physics routines at the (nonuniformly spaced) quadrature points on the spectral element with acknowledgment that a modeling “crime" has been committed; or (2) interpolate between the grid (quadrature points) on which the dynamics is calculated to a grid on which the physics is calculated (and back), and hence incur an interpolation error. Since there is a long-standing history of using the validated physics routines designed for use on uniformly spaced grids, there is a strong incentive to apply the second option. However, interpolating density or other key physical quantities without accounting for property preservation may lead to negative values that are nonphysical and result in inaccurate representations and/or interpretations of the physical data. For example, Skamrock et al. [98] demonstrated that not preserving positivity may lead to a positive bias in a predicted physical quantity of interest (e.g., prediction of moisture). The second option mentioned above of moving information from nonuniform to uniform and back via ENO-type interpolation schemes, explored in [78] in the context of high-order methods for numerical weather prediction, is the main motivation for this work.

Property-preserving interpolation is straightforward when used in the context of low-order numerical simulation methods. High order property-preserving interpolation is, however, nontrivial, especially when the interpolation points are not uniformly spaced. In this chapter, we demonstrate that it is possible to adaptively construct high-order interpolation methods over unevenly spaced tensor product grids in a way that ensures either data boundedness or positivity preservation (within user-supplied bounds). The algorithm we have developed comes with theoretical estimates, presented herein, that provide sufficient conditions for data boundedness and positivity preservation.

4.1.1 Previous Work

In this section, we provide an overview of various numerical approaches to data boundedness and positivity preservation. This overview is not meant to be exhaustive, but instead to summarize the various ways by which researchers have attempted to tackle this challenging problem. Introduced by Harten et al. [39], Essentially Non-Oscillatory (ENO) schemes were developed to solve problems with sharp gradients and discontinuities while achieving high-order accuracy in both smooth and nonsmooth regions. As with many finite-difference-based methods, the backbone of these schemes is interpolation methods. In the context of this chapter, which is to propose ENO-like interpolation schemes that are property preserving, we briefly review ENO methods. In the context of finite volume schemes, Fjordholm et al. [29] demonstrated that ENO schemes are stable, in the sense that the jump of the reconstructed value at each cell interface has the same sign as the jump in the underlying cell average. Building on the work in [101] and [29], Fjordholm et al. [28] developed a high-order entropy stable ENO scheme for conservation laws. This approach consists of using entropy conservative flux based on [101], adding a numerical diffusion to obtain a stable scheme, and obtaining the high-order accuracy via ENO reconstruction.

Harten [37], [38] developed an ENO scheme for subcell resolution in the cases where a discontinuity lies inside a given cell. Weighted Essentially Non-Oscillatory (WENO) schemes were later proposed by Liu et al. [63] to address some of the shortcomings of the ENO schemes. Shu [95] provided a comprehensive overview of different applications and problems in which ENO and WENO schemes are used. Shen et al. [92] proposed an adaptive mesh refinement method (AMR) based on WENO schemes for hyperbolic conservation laws. In this approach, high-order WENO interpolation is used for the prolongation. A generalization of the AMR-WENO in [105] was used to solve a multidimension detonation problem.

Another body of literature sometimes considered around property-preserving methods is computer-aided design and visualization. Although different from the finite difference (stencil) methods that we seek, we briefly review this literature. In this literature, “shape" preservation is often used to describe the preservation of properties like monotonicity and convexity, and may include positivity and data boundedness [15] and [13]. We only briefly review this literature as the additional smoothness constraints at the stencil points enforced by these methods introduce a level of complexity not needed for our application domain. Our focus is finite difference ENO-type schemes. Perhaps the most widely used approach for preserving monotonicity in many applications is PCHIP by Fritch and Carlson [33], who derived necessary and sufficient conditions for monotone cubic interpolation, and provided an algorithm for building a piecewise cubic approximation from data. This algorithm calculates the values of the first derivatives at the nodes based on the necessary and sufficient conditions. Lux et al. [64] proposed a monotone quintic spline (MQS) algorithm that relies on the results of Heß and Schmidt [90] and Ulrich and Watson [104]. This method is dependent on the value of the first and second derivatives at the node. The algorithm uses the sufficient conditions from [90] to check for monotonicity. When the conditions are not met, the method in [104] is used to modify the values of the first and second derivatives to ensure monotonicity. The work of Dougherty et al. [21] extends these ideas to preserving convexity and concavity and also to quintic splines.

A second area in which one often finds the development of methods for property preservation is numerical methods for partial differential equations (PDEs). Various methods have been developed to enable, for example, positivity-preserving approximations. To preserve positivity in discontinuous Galerkin (dG) schemes, Zhang et al. [110], [113], [111] introduced a linear rescaling of polynomials that ensures that the evaluation of the polynomial at the quadrature points remains positive. In addition, the linear rescaling of the polynomial conserves mass. Light et al. [60] developed a similar approach with a more involved linear polynomial rescaling that preserves positivity at the quadrature nodes and conserves mass. The polynomial rescaling does not address the case of interpolating between different meshes, which is the primary focus of this work. Harten et al. [39] developed an Essentially Non-Oscillatory (ENO) piecewise polynomial reconstruction that enables interpolation between different meshes. The ENO method adaptively chooses stencil points for the interpolation and helps remove Gibbs-like effects but does not guarantee positivity. As previously mentioned, extensions of these ideas to a Weighted ENO (WENO) combination of these schemes have been proposed by Zhang et al. [112] and others. Finally, Zala et al. [108], [109] developed a nonlinear filtering operator for property-preservation by casting it as an optimization problem in which the desired “structures" (properties) are encoded as constraints.

The data-bounded interpolation (DBI) method of Berzins [5] builds on three ideas from these ENO and WENO algorithms in the area of the numerical solution of advection equations: adaptively selecting stencils as in the ENO methods to reduce oscillations [39]; altering the polynomial approximation so that any discontinuities in higher derivatives are removed [38]; and altering the polynomial degree and/or terms so that the ratio of successive divided differences in the series is strictly limited to enforce the boundedness of the interpolation [5]. The work in [3] extends the earlier proof to 1D unevenly spaced points where, in addition to the interval points, all the remaining points used to build the interpolant are to the right or left of the interval of interest. In addition, the work in [3] recognizes that switching off data boundedness when extrema are present is important for maintaining accuracy. Positivity is important in interpolation cases in which extrema lie between data points and where the data-bounded interpolant will “clip" the function, resulting in a loss of accuracy. A novel feature of the approach addresses the fact that preserving positivity alone may still produce undesirable oscillations that lead to an inaccurate representation and/or interpretation of the underlying data. These oscillations are removed here by imposing strict user-supplied bounds on the positive interpolants as a way of limiting oscillations and correspondingly improving accuracy.

This work extends the ideas in [5] by addressing data boundedness and positivity (within user-supplied bounds) in the same framework and by allowing meshes of unevenly spaced points. The DBI method presented in this chapter introduces more relaxed conditions for data-boundedness which give greater accuracy than the conditions used in [5]. Thus, these new proofs provide the previously missing theoretical underpinning for complex interpolation cases such as those like the NWP case described above. The new approach used here both generalizes the DBI method to unevenly spaced structured meshes and extends the approach to preserve positivity (positivity-preserving interpolation (PPI)) rather than the more restrictive data-bounded approach in [5] and [3].

4.2 Background

The approach introduced in this work relies on the Newton polynomial [56, 103] representation to build interpolants that are positive or bounded by the data values. The ability to adaptively select the divided differences or the stencil as in ENO methods [39] is central to the data-bounded and positivity-preserving interpolation approaches presented in this work.

Consider a 1D mesh defined as follows:

$$\mathcal{M} = \{x_{i-J}, \cdots, x_i, x_{i+1}, \cdots, x_{i+L}\}, \tag{4.1}$$

where $x_{i-J} < \cdots < x_i < x_{i+1} < \cdots < x_{i+L}$, and $\{u_{i-J}, \cdots, u_{i+L}\}$ is the set of data values associated with the mesh points. In the definition of the mesh $\mathcal{M}$, the subscripts J, L, $i, \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$, and x_k, $u_k \in \mathbb{R}$ for $i - J \leq k \leq i + L$. For the given mesh $\mathcal{M}$, the Newton divided differences are recursively defined as follows:

$$\begin{cases} U[x_i] = u_i \\ U[x_i, \cdots, x_{i+j}] = \frac{U[x_{i+1}, \cdots, x_{i+j}] - U[x_i, \cdots, x_{i+j-1}]}{x_{i+j} - x_i}. \end{cases} \tag{4.2}$$

The ENO procedure starts by setting the initial stencil $\mathcal{V}_0$:

$$\mathcal{V}_0 = \{x_i, x_{i+1}\} = \{x_0^l, x_0^r\}. \tag{4.3}$$

The stencil $\mathcal{V}_0$ is expanded by successively appending a point to the right or left of $\mathcal{V}_j$ to form $\mathcal{V}_{j+1}$. The point appended is selected by picking the smallest divided difference at each step.

Given $\mathcal{V}_j$, let x_j^l and x_j^r be the leftmost and rightmost stencil points, respectively. In addition, let x_p and x_q be the stencil points immediately to the left and right of $\mathcal{V}_j$. The stencil is expanded from $\mathcal{V}_j$ to $\mathcal{V}_{j+1}$ based on the following rules:

- if $|U[x_p, x_j^l, \cdots, x_j^r]| < |U[x_j^l, \cdots, x_j^r, x_q]|$ then
  $\mathcal{V}_{j+1} = \{x_p, \mathcal{V}_j\}$ with $x_{j+1}^l = x_p$ and $x_{j+1}^r = x_j^r$.
- otherwise
  $\mathcal{V}_{j+1} = \{\mathcal{V}_j, x_q\}$ with $x_{j+1}^l = x_j^l$ and $x_{j+1}^r = x_q$.

Let

$$I_i = [x_i, x_{i+1}], \quad \text{for } 0 \leq i \leq n-1. \tag{4.4}$$

Once the final stencil $\mathcal{V}_{n-1}$ is obtained, the interpolant of degree n defined on I_i can be written as

$$U_n(x) = u_i + U[x_0^l, x_0^r]\pi_{0,i}(x) + U[x_1^l, \cdots, x_1^r]\pi_{1,i}(x) + \cdots + U[x_{n-1}^l, \cdots, x_{n-1}^r]\pi_{n-1,i}(x), \tag{4.5}$$

where $\pi_{0,i}(x) = (x - x_i), \pi_{1,i}(x) = (x - x_i)(x - x_1^e), \cdots$ are the Newton basis functions. x_j^e is the point added to expand the stencil $\mathcal{V}_{j-2}$ to $\mathcal{V}_{j-1}$ and can be explicitly expressed as

$$\begin{cases} x_0^e = x_i, \\ x_1^e = x_{i+1}, \\ x_j^e = \mathcal{V}_{j-1} \setminus \mathcal{V}_{j-2}, \quad 2 \leq j \leq n-1. \end{cases} \tag{4.6}$$

The first step in developing the DBI and PPI methods consists of reorganizing the terms in the polynomial $U_n(x)$ defined in Equation (6.3) to expose the features used to enforce data boundedness and positivity. The reorganization begins by defining λ_j as follows:

$$\lambda_j = \begin{cases} 1, \quad j = 0 \\ \frac{U[x_j^l, \cdots, x_j^r]}{U[x_{j-1}^l, \cdots, x_{j-1}^r]}(x_j^r - x_j^l), \quad 1 \leq j \leq n-1. \end{cases} \tag{4.7}$$

Expressing $U_n(x)$ in terms of λ_j, for $j > 0$ gives

$$\begin{aligned} U_n(x) = u_i + (u_{i+1} - u_i)\frac{x - x_0^e}{x_0^r - x_0^l}\Bigg(1 + \frac{(x - x_1^e)}{(x_1^r - x_1^l)}\lambda_1 \\ \Bigg(1 + \frac{(x - x_2^e)}{(x_2^r - x_2^l)}\lambda_2\Bigg(\cdots\lambda_{n-2}\Bigg(1 + \frac{(x - x_{n-1}^e)}{(x_{n-1}^r - x_{n-1}^l)}\lambda_{n-1}\Bigg)\cdots\Bigg). \end{aligned} \tag{4.8}$$

For $x \in I_i$, s, t_j, and d_j are defined as follows:

$$0 \leq s = \frac{x - x_i}{x_{i+1} - x_i} = \frac{x - x_0^e}{x_0^r - x_0^l} \leq 1, \tag{4.9}$$

$$t_j = -\frac{x_i - x_j^e}{x_0^r - x_0^l}, \text{ and} \tag{4.10}$$

$$0 \le d_j = \frac{x_j^r - x_j^l}{x_0^r - x_0^l}. \tag{4.11}$$

s and d_j are defined such that $s \in [0,1]$ and $d_j \ge 0$. Expressing $\frac{x - x_j^e}{x_j^r - x_j^l}$ in terms of s, t_j, and d_j gives

$$\frac{x - x_j^e}{x_j^r - x_j^l} = \frac{\frac{x - x_i}{x_0^r - x_0^l} + \frac{x_i - x_j^e}{x_0^r - x_0^l}}{\frac{x_j^r - x_j^l}{x_0^r - x_0^l}} = \frac{s - t_j}{d_j}. \tag{4.12}$$

Using the results from Equation (4.12), the polynomial $U_n(x)$ as expressed in Equation (4.8) can be written as

$$U_n(x) = u_i + (u_{i+1} - u_i)S_n(x) \tag{4.13}$$

with $S_n(x)$ defined as

$$S_n(x) = s\left(1 + \frac{(s-1)}{d_1}\lambda_1\left(1 + \frac{(s-t_2)}{d_2}\lambda_2\left(\cdots\left(1 + \frac{(s-t_{n-1})}{d_{n-1}}\lambda_{n-1}\right)\cdots\right)\right)\right). \tag{4.14}$$

For future use below, $S_n(x)$ can be compactly represented by introducing δ_j defined as

$$\begin{cases} \delta_n = 1 \\ \delta_j = 1 + \frac{s-t_j}{d_j}\lambda_j\delta_{j+1} \quad 2 \le j \le n-1 \\ \delta_1 = s + \frac{s(s-1)}{d_1}\delta_2 = S_n(x). \end{cases} \tag{4.15}$$

Together, $U_n(x)$ and $S_n(x)$ in Equations (6.6) and (6.7) are reorganizations needed to construct the DBI and PPI algorithm. The general approach is to first bound the quadratic term in $S_n(x)$ and then to increase the order to cubic, quartic, and higher order polynomials. This iterative procedure is used to define computational bounds on the values of $\bar{\lambda}_j = \prod_{k=0}^{j} \lambda_k$. $\bar{\lambda}_j$ can be explicitly written as

$$\bar{\lambda}_j = \lambda_j \bar{\lambda}_{j-1} = \prod_{k=1}^{j} \lambda_k = \begin{cases} 1 \quad j = 0, \\ \frac{U[x_j^l, \cdots, x_j^r]}{U[x_0^l, x_0^r]} \prod_{k=1}^{j}(x_k^r - x_k^l), \quad 1 \le j \le n-1. \end{cases} \tag{4.16}$$

4.3 Data-Bounded Interpolation

The DBI method builds on three ideas from algorithms in the area of the numerical solution of advection equations: adaptively selecting stencils as in the ENO methods to reduce oscillations [39]; altering the polynomial approximation so that any discontinuities in higher derivatives are removed [38]; and altering the polynomial degree and/or terms so that the ratio of successive divided differences in the series is strictly limited to enforce the boundedness of the interpolation [5]. In the DBI method introduced here, more relaxed bounds on $\bar{\lambda}_j$ defined in Equation (6.11) are derived, which gives greater accuracy than those in [5]. The work in [5] requires that the absolute values of $\bar{\lambda}_j$ decrease as more terms are added ($|\bar{\lambda}_j| > |\bar{\lambda}_{j+1}|$) and $|\bar{\lambda}_j| < 1$, which are more restrictive than the bounds in Equation (4.27). For a given set of mesh points and the data values associated with those mesh points, we approximate the data with a $\mathbf{C}^0$ continuous function that is built by fitting a polynomial in each subinterval I_i. The fitted polynomial is constructed in such a way that it is bounded by u_i and u_{i+1}. Given that this work concerns itself with locally fitting a polynomial in the interval I_i, let us assume, for the remaining parts of this chapter, that $x \in I_i$ and that building the interpolant always starts with the stencil $\mathcal{V}_0 = \{x_i, x_{i+1}\}$.

Let $U^l(x)$ be the limited polynomial defined as in Equation (6.6) and bounded by u_i and u_{i+1}. For the polynomial $U^l(x)$ to be bounded by u_i and u_{i+1}, it follows that for $x \in I_i$

$$0 \leq S_n(x) \leq 1, \tag{4.17}$$

with $S_n(x)$ defined in Equation (6.7). The reconstruction procedure begins by considering the linear and quadratic terms from $S_n(x)$ in Equation (6.7), and imposing the following bounds:

$$0 \leq s\big(1 + \frac{s-1}{d_1}\bar{\lambda}_1\big) \leq 1. \tag{4.18}$$

As $s \in [0, 1]$ and isolating $\bar{\lambda}_1$ in Equation (4.18) gives

$$-\frac{d_1}{s} \leq \bar{\lambda}_1 \leq \frac{d_1}{1-s}, \text{ and} \tag{4.19}$$

$$-d_1 \leq \bar{\lambda}_1 \leq d_1. \tag{4.20}$$

The bounds from Equation (4.20) are extended to bound the cubic form by requiring that what multiplies $\bar{\lambda}_1$ must fit into the inequality in Equation (4.20). Thus, for the cubic case Equation (4.20) becomes

$$-d_1 \leq \bar{\lambda}_1\big(1 + \frac{(s-t_2)}{d_2}\lambda_2\big) \leq d_1. \tag{4.21}$$

Subtracting $\bar{\lambda}_1$ from this inequality gives

$$-d_1 - \bar{\lambda}_1 \leq \frac{(s-t_2)}{d_2}\bar{\lambda}_2 \leq d_1 - \bar{\lambda}_1. \tag{4.22}$$

In the case when t_2 is negative, $s - t_2$ has a maximum value at $s = 1$ and a minimum value at $s = 0$. $\bar{\lambda}_2$ is then bounded by

$$\frac{d_2}{(1-t_2)}(-d_1 - \bar{\lambda}_1) \leq \bar{\lambda}_2 \leq (d_1 - \bar{\lambda}_1)\frac{d_2}{(1-t_2)}. \tag{4.23}$$

When t_2 positive, $\frac{1}{1-t_2}$ is substituted by $\frac{1}{-t_2}$ and the inequalities $\leq$ with $\geq$ and vice versa are swapped. In the quartic case, we require that

$$\frac{d_2}{1-t_2}(-d_1 - \bar{\lambda}_1) \leq \bar{\lambda}_2\left(1 + \frac{(s-t_3)}{d_3}\lambda_3\right) \leq \frac{d_2}{1-t_2}(d_1 - \bar{\lambda}_1). \tag{4.24}$$

If we assume that t_3 is negative

$$\frac{d_3}{1-t_3}\left(\frac{d_2}{1-t_2}(-d_1 - \bar{\lambda}_1) - \bar{\lambda}_2\right) \leq \bar{\lambda}_3 \leq \frac{d_3}{1-t_3}\left(\frac{d_2}{1-t_2}(d_1 - \bar{\lambda}_1) - \bar{\lambda}_2\right). \tag{4.25}$$

This reconstruction procedure can be continued to higher orders provided that care is taken to correctly manage the impact of the signs of t_j. For the boundary and nearby boundary intervals, fewer choices are available, and the final stencil is biased toward the interior of the domain because there are no points to choose from beyond the boundaries. In the process of constructing $\mathcal{V}_{n-1}$, when the left or right boundary are reached, the remaining mesh points are obtained from the side that is toward the interior of the domain.

For a more formal and complete expression of this recursive procedure, the bounds on $\bar{\lambda}_j$ can be defined as follows:

$$B_j^- = \begin{cases} -d_1 & j = 0 \\ (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{1-t_j}, & t_j \in (-\infty, 0] \quad j > 1 \\ (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{-t_j}, & t_j \in (0, +\infty) \quad j > 1, \end{cases} \tag{4.26a}$$

and

$$B_j^+ = \begin{cases} d_1, & j = 1 \\ (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{1-t_j}, & t_j \in (-\infty, 0] \quad j > 1 \\ (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{-t_j}, & t_j \in (0, +\infty) \quad j > 1. \end{cases} \tag{4.26b}$$

The sign of t_j is incorporated into the definitions of B_j^- and B_j^+ in Equations (4.26a) and (4.26b), respectively. The sufficient conditions for data boundedness such as Equations (4.20), (4.23) and (4.25) can now be written as

$$B_j^- \leq \bar{\lambda}_j \leq B_j^+, \text{ for } j \geq 0. \tag{4.27}$$

Lemma 4.3.1. *Let us assume that for $x \in I_i$, B_j^- and B_j^+ are defined as in Equations (4.26b) and (4.26a), respectively. In addition, let δ_j be defined as in Equation (4.15). If for $x \in I_i$, B_j^- is negative, B_j^+ is positive, and $B_j^- \leq \bar{\lambda}_j \delta_{j+1} \leq B_j^+$, then*

$$B_{j-1}^- \leq \bar{\lambda}_{j-1}\delta_j \leq B_{j-1}^+.$$

Proof. The proof is split into two cases that take into consideration the different possible values of t_j, and in each case we consider the left and right side of the inequality separately.

(I) $t_j \in (-\infty, 0]$

Let us start with the left side of the inequality (i.e., $B_{j-1}^- \leq \bar{\lambda}_{j-1}\delta_j$). Noting that $\frac{1-t_j}{s-t_j} \geq 1$ for $s \in [0,1]$, and using $B_j^- \leq 0$ and $B_j^- \leq \bar{\lambda}_j \delta_{j+1}$, we have

$$\begin{aligned} (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{s-t_j} =& \frac{1-t_j}{s-t_j} B_j^- \\ \leq& B_j^- \\ \leq& \bar{\lambda}_j \delta_{j+1}. \end{aligned} \tag{4.28}$$

Isolating B_{j-1}^- in Equation (4.28) and using Equations (4.15) and (6.11) lead to

$$\begin{aligned} B_{j-1}^- \leq& \bar{\lambda}_{j-1} + \frac{s-t_j}{d_j}\bar{\lambda}_j \delta_{j+1} \\ \leq& \bar{\lambda}_{j-1}\left(1 + \frac{s-t_j}{d_j}\lambda_j \delta_{j+1}\right) \\ =& \bar{\lambda}_{j-1}\delta_j. \end{aligned} \tag{4.29}$$

Now, let us focus on the right side of the inequality (i.e., $B_{j-1}^+ \geq \bar{\lambda}_{j-1}\delta_j$) Again, observing that $\frac{1-t_j}{s-t_j} \geq 1$ for $s \in [0,1]$ and using $B_j^+ \geq 0$ and $B_j^+ \geq \bar{\lambda}_j \delta_{j+1}$ yields

$$\begin{aligned} (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{s-t_j} =& \frac{1-t_j}{s-t_j} B_j^+ \\ \geq& B_j^+ \\ \geq& \bar{\lambda}_j \delta_{j+1}. \end{aligned} \tag{4.30}$$

Isolating B_{j-1}^+ in Equation (4.30) yields

$$\begin{aligned} B_{j-1}^+ \geq& \bar{\lambda}_{j-1} + \frac{s-t_j}{d_j}\bar{\lambda}_j \delta_{j+1} \\ \geq& \bar{\lambda}_{j-1}\left(1 + \frac{s-t_j}{d_j}\lambda_j \delta_{j+1}\right) \\ =& \bar{\lambda}_{j-1}\delta_j. \end{aligned} \tag{4.31}$$

(II) $t_j \in (0, +\infty)$

Let us consider the left side of the inequality (i.e., $B^-_{j-1} \leq \bar{\lambda}_{j-1}\delta_j$). Multiplying B^-_j by $\frac{-t_j}{s-t_j}$ yields

$$(B^+_{j-1} - \bar{\lambda}_{j-1})\frac{d_j}{s-t_j} = \frac{-t_j}{s-t_j}B^-_j. \tag{4.32}$$

Given that $B^-_j \leq 0$ and $B^-_j \leq \bar{\lambda}_j\delta_{j+1}$, and noting that $\frac{-t_j}{s-t_j} \geq 1$ for $s \in [0,1]$, the right side of Equation (4.32) can be bounded by B^-_j to give

$$\begin{aligned}(B^+_{j-1} - \bar{\lambda}_{j-1})\frac{d_j}{s-t_j} &\leq B^-_j \\ &\leq \bar{\lambda}_j\delta_{j+1}.\end{aligned} \tag{4.33}$$

Isolating B^+_{j-1} in Equation (4.33) leads to

$$\begin{aligned}B^+_{j-1} &\geq \bar{\lambda}_{j-1} + \frac{s-t_j}{d_j}\bar{\lambda}_j\delta_{j+1} \\ &\geq \bar{\lambda}_{j-1}\left(1 + \frac{s-t_j}{d_j}\lambda_j\delta_{j+1}\right) \\ &= \bar{\lambda}_{j-1}\delta_j.\end{aligned} \tag{4.34}$$

For the right side of the inequality (i.e. $B^-_{j-1} \leq \bar{\lambda}_{j-1}\delta_j$), $\frac{-t_j}{s-t_j} \geq 1$ for $s \in [0,1]$, and using $B^-_j \leq 0$ and $B^-_j \leq \bar{\lambda}_j\delta_{j+1}$ yields

$$\begin{aligned}(B^-_{j-1} - \bar{\lambda}_{j-1})\frac{d_j}{s-t_j} &= \frac{-t_j}{s-t_j}B^+_j \\ &\geq B^-_j \\ &\geq \bar{\lambda}_j\delta_{j+1}.\end{aligned} \tag{4.35}$$

Isolating B^-_{j-1} in Equation (4.35) yields

$$\begin{aligned}B^-_{j-1} &\leq \bar{\lambda}_{j-1} + \frac{s-t_j}{d_j}\bar{\lambda}_j\delta_{j+1} \\ &\leq \bar{\lambda}_{j-1}\left(1 + \frac{s-t_j}{d_j}\lambda_j\delta_{j+1}\right) \\ &= \bar{\lambda}_{j-1}\delta_j.\end{aligned} \tag{4.36}$$

The results from Equations (4.29), (4.31), (4.29), and (4.31) can be summarized as

$$B^-_{j-1} \leq \bar{\lambda}_{j-1}\delta_j \leq B^+_{j-1}.$$

$\square$

Theorem 4.3.2. *Assuming that for $x \in I_i$, the polynomial $S_n(x)$ of degree n is built starting from the stencil $\mathcal{V}_0 = \{x_i, x_{i+1}\}$, and then by successively appending mesh points from the left and/or right of the interval I_i to obtain the final stencil $\mathcal{V}_{n-1}$. The construction of $\mathcal{V}_{n-1}$ does not require the points to be added in a symmetric fashion alternating from left to right. If for $x \in I_i$, B_j^- defined in Equation (4.26a) is negative, B_j^+ defined in Equation (4.26b) is positive, and $B_j^- \leq \bar{\lambda}_j \leq B_j^+$ then for $x \in I_i$*

$$0 \leq S_N(x) \leq 1.$$

Proof. This proof builds on the results from Lemma 4.3.1 and starts by using $B_j^- \leq \bar{\lambda}_j \leq B_j^+$ to bound $\bar{\lambda}_{n-1}$ as follows:

$$B_{n-1}^- \leq \bar{\lambda}_{n-1} \leq B_{n-1}^+. \tag{4.37}$$

By Lemma 4.3.1, Equation (4.37) then leads to

$$B_{n-2}^- \leq \bar{\lambda}_{n-2}\delta_{n-1} \leq B_{n-2}^+. \tag{4.38}$$

Successively using the results from Lemma 4.3.1 to bound $\bar{\lambda}_{n-2}\delta_{n-1}$, $\bar{\lambda}_{n-3}\delta_{n-2}$, $\cdots$, $\bar{\lambda}_1\delta_2$, yields

$$B_1^- \leq \bar{\lambda}_1\delta_2 \leq B_1^+, \tag{4.39}$$

where δ_j is defined in Equation (4.15). The results from Equation (4.39) may now be used to derive the target bounds (i.e., $0 \leq S_N(x) \leq 1$). Considering the left side of Equation (4.39) (i.e., $B_1^- \leq \bar{\lambda}_1\delta_2$), and noting that $\frac{(s-1)}{s(s-1)} \geq 1$, gives

$$\begin{aligned} -\frac{(s-1)}{s(s-1)}d_1 =& B_1^- \frac{(s-1)}{s(s-1)} \\ \leq& B_1^- \\ \leq& \bar{\lambda}_1\delta_2. \end{aligned} \tag{4.40}$$

Isolating δ_1 from Equation (4.40) gives

$$1 \geq s + \frac{s(1-s)}{d_1}\bar{\lambda}_1\delta_2 = \delta_1 = S_n(x). \tag{4.41}$$

Considering the right side of Equation (4.39) (i.e. $B_1^+ \geq \bar{\lambda}_1\delta_2$), and noting that $\frac{(-s)}{s(s-1)} \geq 1$, gives

$$\begin{aligned} \frac{(-s)}{s(s-1)}d_1 =& B_1^+ \frac{(-s)}{s(s-1)} \\ \geq& B_1^+ \\ \geq& \bar{\lambda}_1\delta_2. \end{aligned} \tag{4.42}$$

Isolating δ_1 from Equation (4.42) gives

$$0 \leq s + \frac{s(1-s)}{d_1}\bar{\lambda}_1\delta_2 = \delta_1 = S_n(x). \tag{4.43}$$

The proof concludes by combining the results from Equations (4.41) and (4.43) to obtain

$$0 \leq s + \frac{s(1-s)}{d_1}\bar{\lambda}_1\delta_2 = \delta_1 = S_n(x) \leq 1. \tag{4.44}$$

□

4.4 Constrained Positivity-Preserving Interpolation

In many cases, it is sufficient to preserve positivity through interpolation and not to enforce the stricter requirement of data boundedness. As mentioned in the introduction, the case of unknown extrema between data points is an important example. Let $U^p(x)$ be a positive polynomial of degree n defined over the interval I_i as in Equation (6.6). For $x \in I_i$, the polynomial $U^p(x)$ is allowed to grow beyond u_i and u_{i+1} but must remain positive. For the polynomial to be positive, one requires that

$$U^p(x) \geq 0. \tag{4.45}$$

However, in practice, enforcing positivity alone may still result in large oscillations and in extrema that degrade the approximation. We observe this behavior because enforcing positivity alone does not restrict how much the polynomial is allowed to grow beyond the data values. In addition to enforcing positivity, it is important to remove the undesirable oscillations and extrema as much as possible. Let us define u_{min} and u_{max} as

$$u_{min} = \mathbf{min}(u_i, u_{i+1}) - \Delta_{min}, \tag{4.46}$$

and

$$u_{max} = \mathbf{max}(u_i, u_{i+1}) + \Delta_{max}, \tag{4.47}$$

where Δ_{min} and Δ_{max} are user-defined parameters used to bound the positive polynomial $U^p(x)$. To allow the polynomial to grow beyond the data values but not produce extrema that are too large, we bound $U^p(x)$ as follows:

$$u_{min} \leq U^p(x) = u_i + (u_{i+1} - u_i)S_n(x) \leq u_{max}. \tag{4.48}$$

The interpolant $U^p(x)$ is now positive and bounded by u_{min} and u_{max}. Equation (6.12) is equivalent to bounding $S_n(x)$ as follows:

$$m_\ell \leq S_n(x) \leq m_r, \tag{4.49}$$

where the factors m_ℓ and m_r are expressed as

(I) : $u_{i+1} > u_i$

$$m_\ell = \mathbf{min}\left(0, \frac{u_{min} - u_i}{u_{i+1} - u_i}\right), \text{ and } m_r = \mathbf{max}\left(1, \frac{u_{max} - u_i}{u_{i+1} - u_i}\right) \tag{4.50}$$

(II) : $u_{i+1} < u_i$

$$m_\ell = \mathbf{min}\left(0, \frac{u_{max} - u_i}{u_{i+1} - u_i}\right), \text{ and } m_r = \mathbf{max}\left(1, \frac{u_{min} - u_i}{u_{i+1} - u_i}\right). \tag{4.51}$$

We note that if we set $\Delta_{min} = 0$ and $\Delta_{max} = 0$, we recover Equation (4.45).

The PPI method is constructed by relaxing the bounds imposed on $\bar{\lambda}_1$ as follows:

$$\Big(-4(m_r - 1) - 1\Big) d_1 \leq \bar{\lambda}_1 \leq \Big(-4m_\ell + 1\Big) d_1. \tag{4.52}$$

Let us demonstrate how the PPI method is constructed in the case of a quadratic interpolant. Starting from the DBI results in the Theorem 4.3.2, it follows that

$$0 \leq s + \frac{s(s-1)}{d_1}\bar{\lambda}_1 \leq 1. \tag{4.53}$$

Relaxing the left and right bounds in Equation (4.53) by m_ℓ and m_r, respectively, leads to

$$m_\ell \leq s + \frac{s(s-1)}{d_1}\bar{\lambda}_1 \leq m_r. \tag{4.54}$$

Isolating $\bar{\lambda}_1$ from Equation (4.54) leads to

$$\frac{m_r - s}{s(s-1)} d_1 \leq \bar{\lambda}_1 \leq \frac{m_\ell - s}{s(s-1)} d_1. \tag{4.55}$$

Equation (4.55) can be reorganized to obtain

$$\left(\frac{m_r - 1}{s(s-1)} + \frac{1-s}{s(s-1)}\right) d_1 \leq \bar{\lambda}_1 \leq \left(\frac{m_\ell}{s(s-1)} - \frac{s}{s(s-1)}\right) d_1 \tag{4.56}$$

and then

$$\left(\frac{m_r - 1}{s(s-1)} - \frac{1}{s}\right) d_1 \leq \bar{\lambda}_1 \leq \left(\frac{m_\ell}{s(s-1)} - \frac{1}{(s-1)}\right) d_1. \tag{4.57}$$

Noting that $\frac{1}{s(s-1)} \leq -4$, $\frac{1}{s} \geq 1$, and $\frac{1}{s-1} \leq -1$, we obtain

$$\Big(-4(m_r - 1) - 1\Big) d_1 \leq \bar{\lambda}_1 \leq \Big(-4m_\ell + 1\Big) d_1. \tag{4.58}$$

Once the bounds on $\bar{\lambda}_1$ and the quadratic interpolant are determined, the extension to cubic, quartic, and higher order interpolants follows the same reconstruction procedure used in the DBI method and outlined from Equation (4.21) to (4.25). As in the case of the DBI method, fewer choices are available for $\mathcal{V}_{n-1}$ at the boundary and nearby boundary intervals because there are no points to choose from beyond the boundaries. When a boundary is reached during the process of constructing the stencil $\mathcal{V}_{n-1}$, the remaining mesh points are picked from the side that is toward the interior of the domain. The final stencils at the boundary and nearby the boundary intervals are biased toward the interior of the domain. The recursive expression for the bounds on $\bar{\lambda}_j$ for the PPI method becomes

$$B_j^- = \begin{cases} (-4(m_r-1)-1)d_1 \quad j=1 \\ (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{1-t_j}, \text{ if } t_j \in (-\infty, 0] \quad j>1 \\ (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{-t_j}, \text{ if } t_j \in (0,1)\cup(1,+\infty) \quad j>1, \end{cases} \tag{4.59a}$$

and

$$B_j^+ = \begin{cases} (-4m_\ell+1)d_1, \quad j=1 \\ (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{1-t_j}, \text{ if } t_j \in (-\infty, 0] \quad j>1 \\ (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{-t_j}, \text{ if } t_j \in (0,+\infty) \quad j>1. \end{cases} \tag{4.59b}$$

The difference between the DBI and PPI methods is highlighted in how the bounds B_1^- and B_1^+ are calculated. More precisely, B_1^- and B_1^+ are defined as $-d_1$ and d_1 for the DBI method, whereas for the PPI method, they are defined as $(-4(m_r-1)-1)d_1$ and $(-4m_\ell+1)d_1$, respectively. In addition, the DBI method can be recovered from the PPI methods by setting $m_\ell = 0$ and $m_r = 1$. For example, in the case of the right boundary Equations (4.20) and (4.58) can be written as

$$-d_1 \le \bar{\lambda}_1 = \frac{U[x_{N-2}, x_{N-1}, x_N]}{U[x_{N-1}, x_N]}(x_N - x_{N-1}) \le d_1, \text{ and} \tag{4.60}$$

$$\Big(-4(m_r-1)-1\Big)d_1 \le \bar{\lambda}_1 = \frac{U[x_{N-2}, x_{N-1}, x_N]}{U[x_{N-1}, x_N]}(x_N - x_{N-1}) \le \Big(-4m_\ell+1\Big)d_1, \tag{4.61}$$

where x_N is the mesh point at the right boundary, $m_\ell \le 0$, $m_r \ge 1$, and

$$d_1 = \frac{x_N - x_{N-2}}{x_N - x_{N-1}}. \tag{4.62}$$

From Equations (6.17) and (6.18), $m_r = 18.94$ and $m_\ell = -18.94$ for the right boundary of the Runge example in Fig. 4.1 below. Equations (4.60) and (4.61) show the bounds on $\bar{\lambda}_1$ for data boundedness and positivity, respectively. Given that $(-4(m_r-1)-1) \le 0$ and $(-4m_\ell+1) \ge 1$, the bounds for positivity are more relaxed than data boundedness. Thus, enabling the use of higher degree polynomials for the PPI method than for the DBI method.

Theorem 4.4.1. *Let us assume that for $x \in I_i$, the polynomials $U_n(x)$ and $S_n(x)$ of degree n are defined as in Equations (6.6) and (6.7), respectively. Both polynomials are built starting from the stencil $\mathcal{V}_0 = \{x_i, x_{i+1}\}$, and then by successively appending mesh points from the left and/or right of the interval I_i to obtain the final stencil $\mathcal{V}_{n-1}$. The construction of $\mathcal{V}_{n-1}$ does not require the points to be added in a symmetric fashion alternating from left to right. If for $x \in I_i$, B_j^- defined in Equation (4.26a) is negative, B_j^+ defined in Equation (4.26b) is positive, and $B_j^- \leq \bar{\lambda}_j \leq B_j^+$ then for $x \in I_i$*

$$m_\ell \leq S_n(x) \leq m_r,$$

where m_ℓ and m_r are provided in Equations (6.17) and (6.18).

Proof. As in Theorem 4.3.2, the proof begins by using the results from Lemma 4.3.1 and the expression $B_j^- \leq \bar{\lambda}_j \leq B_j^+$ to bound $\bar{\lambda}_{n-2}\delta_{n-1}, \bar{\lambda}_{n-3}\delta_{n-2}, \cdots, \bar{\lambda}_1\delta_2$ and so to obtain the result

$$B_1^- \leq \bar{\lambda}_1\delta_2 \leq B_1^+. \tag{4.63}$$

Equation (4.63) is then used to derive the target bounds. Starting with the left side of the inequality (i.e., $B_1^- \leq \bar{\lambda}_1\delta_2$) and noting that $\frac{1}{s(s-1)} \leq -4$ and $-\frac{1}{s} \leq -1$, yields

$$\begin{aligned}
\frac{m_r - s}{s(s-1)}d_1 =& \left(\frac{m_r - 1}{s(s-1)} + \frac{1-s}{s(s-1)}\right)d_1 \\
=& \left(\frac{m_r - 1}{s(s-1)} - \frac{1}{s}\right)d_1 \\
\leq& \Big(-4(m_r - 1) - 1\Big)d_1 \\
=& B_1^- \\
\leq& \bar{\lambda}_1\delta_2.
\end{aligned} \tag{4.64}$$

Isolating m_r, leads to the desired result

$$m_r \geq s + \frac{s(s-1)}{d_1}\bar{\lambda}_1\delta_2 = \delta_1 = S_n(x). \tag{4.65}$$

Now, addressing the right side of the inequality (i.e. $B_1^+ \geq \bar{\lambda}_1\delta_2$) and noting that $\frac{1}{s(s-1)} \leq -4$ and $-\frac{1}{s-1} \geq 1$, gives

$$
\begin{aligned}
\frac{m_\ell - s}{s(s-1)} d_1 =& \left(\frac{m_\ell}{s(s-1)} - \frac{s}{s(s-1)} \right) d_1 \\
=& \left(\frac{m_\ell}{s(s-1)} - \frac{1}{(s-1)} \right) d_1 \\
\geq& \left(-4m_\ell + 1 \right) d_1 \\
=& B_1^+ \\
\geq& \lambda_1 \delta_2 .
\end{aligned}
\tag{4.66}
$$

Isolating m_ℓ leads to the desired bound

$$
m_\ell \leq s + \frac{s(s-1)}{d_1} \bar{\lambda}_1 \delta_2 = \delta_1 = S_n(x). \tag{4.67}
$$

The proof is concluded by combining Equations (4.65) and (4.67) to obtain

$$
m_\ell \leq s + \frac{s(s-1)}{d_1} \bar{\lambda}_1 \delta_2 = \delta_1 = S_n(x) \leq m_r. \tag{4.68}
$$

□

At the boundary intervals, both the DBI and PPI methods construct the interpolants using a left- or right-biased stencil. For the left boundary, the final stencil is built by successively appending mesh points from the right side of the interval I_i. In the same way, the final stencil for the right boundary interval is obtained by successively appending the mesh points from the left side. For the nearby boundary intervals, the stencil points selection process could reach the boundary before completing the final stencil. In such a case, the remaining points are selected from the right if the left boundary is reached and from the left is the right boundary is reached.

4.4.1 Hidden Local Extrema

The interval I_i may contain a hidden extremum when two of three divided differences $U[x_{i-1}, x_i]$, $U[x_{i+1}, x_i]$ and $U[x_{i+1}, x_{i+2}]$ of the neighboring intervals are of opposite signs. In this case, the PCHIP and DBI algorithms truncate the extremum whereas the relaxed nature of the PPI algorithm allows for a better approximation of the extremum. In [3], when an extremum is detected, the ENO approach is used to construct the interpolant. The ENO approach may fail to recover the extremum or result in oscillations that violate the requirements for positivity and reduce the accuracy. The data-bounded method in [3] is much more restrictive and does not address positivity. These limitations can be addressed by using a bounded positive interpolant.

To simplify the notation, let us define σ_{i-1}, σ_i and σ_{i+1} such that

$$\sigma_{i-1} = U[x_{i-1}, x_i],\ \sigma_i = U[x_{i+1}, x_i],\ \text{and}\ \sigma_{i+1} = U[x_{i+1}, x_{i+2}]. \tag{4.69}$$

As in [3] and [91], we assume that there exists an extremum in I_i if

$$\sigma_{i-1}\sigma_{i+1} < 0,\ \text{or}\ \sigma_{i-1}\sigma_i < 0. \tag{4.70}$$

To address the cases with and without extremum, we choose the parameters Δ_{min} and Δ_{max} according to

$$\Delta_{min} = \begin{cases} \left|\epsilon_1 \mathbf{min}(u_i, u_{i+1})\right| & \text{if } \sigma_{i-1}\sigma_{i+1} < 0 \text{ and } \sigma_{i-1} < 0 \\ & \text{or } \sigma_{i-1}\sigma_{i+1} \geq 0 \text{ and } \sigma_{i-1}\sigma_i < 0 \\ \epsilon_0 \left|\mathbf{min}(u_i, u_{i+1})\right| & \text{otherwise,} \end{cases} \tag{4.71}$$

and

$$\Delta_{max} = \begin{cases} \epsilon_1 \left|\mathbf{max}(u_i, u_{i+1})\right| & \text{if } \sigma_{i-1}\sigma_{i+1} < 0 \text{ and } \sigma_{i-1} > 0 \\ & \text{or } \sigma_{i-1}\sigma_{i+1} \geq 0 \text{ and } \sigma_{i-1}\sigma_i < 0 \\ \epsilon_0 \left|\mathbf{max}(u_i, u_{i+1})\right| & \text{otherwise.} \end{cases} \tag{4.72}$$

The positive parameters ϵ_0 and ϵ_1 are introduced to adjust Δ_{min} and Δ_{max} when no extremum is detected. In Equation (6.14), the interval I_i has a local maximum if $\sigma_{i-1}\sigma_{i+1} < 0$ and $\sigma_{i-1} < 0$. Correspondingly, in Equation (6.15), the interval I_i has a local minimum if $\sigma_{i-1}\sigma_{i+1} < 0$ and $\sigma_{i-1} > 0$. In both Equations (6.14) and 6.15, the type of extremum is ambiguous if $\sigma_{i-1}\sigma_{i+1}$, and $\sigma_{i-1}\sigma_i < 0$. When an extremum is identified, Δ_{min} and/or Δ_{max} are chosen to be sufficiently large to allow the interpolant $U^p(x)$ to grow beyond the data as needed to approximate the extremum without violating the requirement for positivity. In the case where no extremum is identified, the parameter ϵ_0 is used to adjust Δ_{min} and/or Δ_{max} to be sufficiently large to allow higher degree interpolants compared to the DBI method, but sufficiently small to not allow for large oscillations that will degradate the accuracy of the approximation.

In Fig. 4.1, we approximate the Runge function with $N = 17$ LGL points and different values of ϵ_0, and the target polynomial degree is set to $d = 16$ for each interval. For $\epsilon_0 > 0.01$, the PPI method leads to oscillations, whereas for $\epsilon_0 \leq 0.01$ the oscillations are removed. Similar oscillations are seen when using high-order Chebyshev polynomials. The

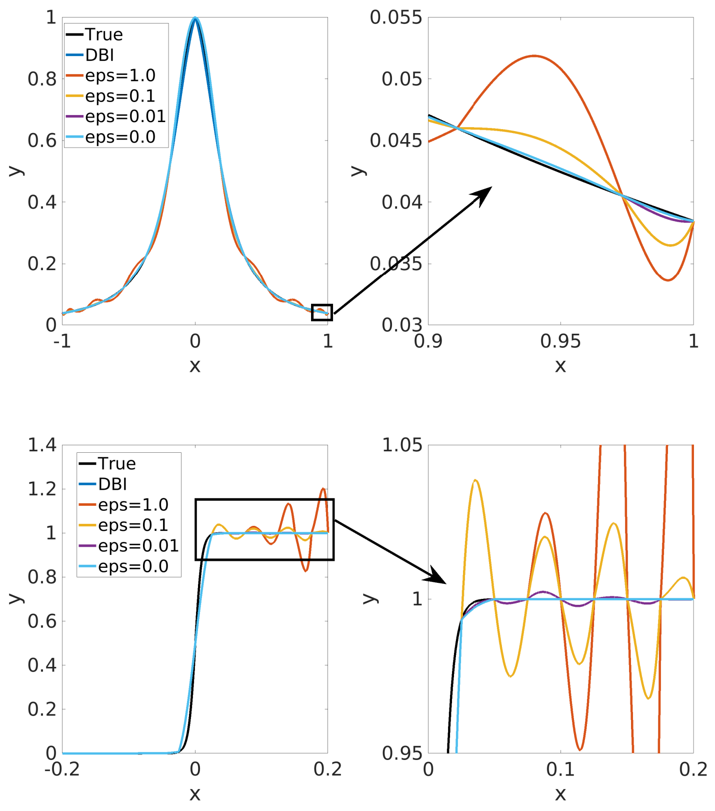


Fig. 4.1: The top row shows an approximation of $f_1(x)$ from $N = 17$ LGL points using DBI and PPI with different values of ϵ_0. The bottom row shows an approximation of $f_2(x)$ from $N = 17$ uniformly spaced points using DBI and PPI with different values of ϵ_0. The values of ϵ_1 is set to 1.0. The target polynomial degree is set to $d = 16$ for both $f_1(x)$ and $f_2(x)$.

cutoff for the positive parameter ϵ_0 depends on the underlying function and the input data. For the Runge example with $N = 17$ uniformly spaced points, the spurious oscillations are removed for $\epsilon_0 \leq 0.05$. With the same Runge example with $N = 129$ and $d = 16$, the unconstrained approximation does not produce oscillations and ϵ_0 can be set to any value in $[0, 1]$. In the case of the smoothed Heaviside examples, setting $\epsilon_0 = 0.05$ with $N = 17$ uniformly spaced points lead to large oscillations that degrade the approximations. However, for $\epsilon_0 \leq 0.01$ with $N = 17$, the oscillations are significantly reduced, and the approximation improved, as shown in the bottom part of Fig. 4.1. Setting $\epsilon_0 = 0.0$ will completely eliminate the oscillations. Overall, using $\epsilon_0 \leq 0.01$ is sufficient to remove or significantly reduce the oscillations and improve the approximation. For an interval I_i with no extremum, as ϵ_0 approaches zero and both Δ_{min} and Δ_{max} get smaller, the approximation method becomes closer to the DBI approach. As for the DBI approach, the PPI method may become restrictive for higher degree polynomial interpolants as ϵ_0 approaches zero. This approach is also further explored for a variety of practical applications [78].

The right part of Fig. 4.1 shows the interpolants used at the right boundaries in both the Runge and smoothed Heaviside examples. At the right boundary of the Runge example, the stencil $\{x_{N-12} \cdots x_N\}$ is used to build the data-bounded interpolant and the stencil $\{x_{N-16}, \cdots, x_N\}$ is used for the positive interpolant with $\epsilon_0 = 1$. As the positive parameter ϵ_0 gets smaller the upper and lower bounds for the interpolant gets tighter and converges to the DBI bounds. The stencil used for both the DBI and PPI are the same for $\epsilon_0 \leq 0.01$. At the boundary intervals, the PPI method allows for higher degree interpolants compared to the DBI method. However, these higher degree interpolants while positive may introduce oscillations that can be removed using the parameter ϵ_0. For $u_i = u_{i+1}$, m_ℓ, m_r and $U_n(x)$ as written in Equations (6.17), (6.18) and (6.6) are not defined. The PPI algorithm addresses this limitation by rewriting $U_n(x)$ as

$$U_n(x) = u_i + U[x_1^l, \cdots, x_1^r](x_{i+1} - x_i)(x_1^r - x_1^l)S_n(x), \tag{4.73}$$

where $S_n(x)$ is expressed as follows:

$$S_n(x) = \sum_{j=1}^{n-1} \bar{s}_j. \tag{4.74}$$

The summation starts at $j = 1$ because the linear term $\frac{u_{i+1}-u_i}{x_{i+1}-x_i}(x - x_i) = 0$. Let

$$w = U[x_1^l, \cdots, x_1^r](x_{i+1} - x_i)(x_1^r - x_1^l). \tag{4.75}$$

$\bar{\lambda}_j$ in this context is defined as

$$\bar{\lambda}_j = \frac{U[x_j^l, \cdots, x_j^r]}{w} \prod_{k=0}^{j} (x_k^r - x_k^l). \tag{4.76}$$

For $u_i = u_{i+1}$, the parameters m_ℓ and m_r are then defined according to

(I) : $U[x_1^l, \cdots, x_1^r] > 0$

$$m_\ell = \mathbf{min}\left(0, \frac{u_{min} - u_i}{w}\right), \text{ and } m_r = \mathbf{max}\left(1, \frac{u_{max} - u_i}{w}\right) \tag{4.77}$$

(II) : $U[x_1^l, \cdots, x_1^r] < 0$

$$m_\ell = \mathbf{min}\left(0, \frac{u_{max} - u_i}{w}\right), \text{ and } m_r = \mathbf{max}\left(1, \frac{u_{min} - u_i}{w}\right). \tag{4.78}$$

For $U[x_i, x_{i+1}] = U[x_1^l, \cdots, x_1^r] = 0$, the data u_{i-1}, u_i, u_{i+1}, and u_{i+2} have the same value ($u_{i-1} = u_i = u_{i+1} = u_{i+2}$). In this case, the algorithm approximates the function in the interval I_i with a linear interpolant. For both cases $U[x_1^l, \cdots, x_1^r] < 0$ and $U[x_1^l, \cdots, x_1^r] > 0$, B_j^+ and B_j^- remain defined as previously in Equations (6.19b) and (6.19a). Lemma 4.3.1 and Theorem 4.4.1 still hold and remain unchanged.

Fig. 4.2 shows an example with $u_i = u_{i+1}$ and a hidden local extremum at $x = 0$. In Fig. 4.2, we approximate the Runge function $f_1(x)$ using the PCHIP, DBI, and PPI methods from 16 uniformly spaced data points. The PPI method is able to better capture the peak compared to the DBI and PCHIP methods.

4.4.2 Algorithm

The ENO reconstruction can result in a stencil that is biased to the left or right. Rogerson et al. [85] demonstrated that a biased ENO stencil may lead to some stability issues when used to solve hyperbolic equations, and a refined resolution may lead to even larger errors. To address this limitation, Shu [93] developed a modified ENO reconstruction that uses a bias coefficient to target a preferred final stencil. Furthermore, a left- and right-biased stencil may fail to recover hidden local extrema. For instance, if $U[x_{i-1}, x_i] > 0$, $U[x_i, x_{i+1}] < 0$, and $U[x_{i+1}, x_{i+2}] > 0$, the interval I_i has an extremum. In such a case, if the points in the final stencil are all to the right or left of x_i, the interpolant may fail to recover the extremum. The points x_{i-1} and x_{i+2} are important for identifying and reconstructing a hidden local

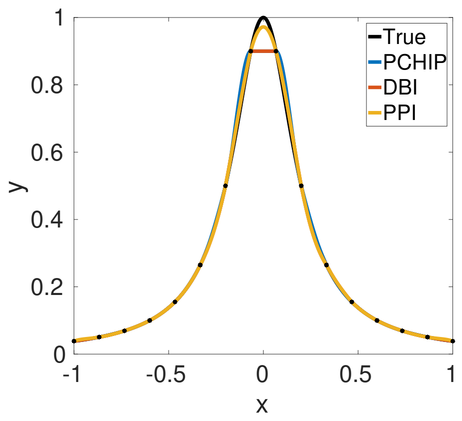


Fig. 4.2: Approximation of $f_1(x)$ with $N = 16$ points using PCHIP, DBI, and PPI. The interpolants from DBI and PPI are in $\mathcal{P}_8$, where 8 is the target polynomial degree. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively.

extremum. However, the right-biased stencils does not include x_{i-1}, and the left-biased stencil does not include x_{i+2}. To resolve these issues due to biased stencils, the algorithm introduced here favors a symmetric stencil over the ENO stencil in addition to enforcing the requirements for data boundedness or positivity preservation. A symmetric stencil centered around x_i includes x_{i-1} and x_{i+2} and better approximates a hidden local extremum compared to a biased stencil.

Before we present the algorithm for the DBI and PPI method, let us define $\bar{\lambda}^-_{j+1}$ and $\bar{\lambda}^+_{j+1}$. At any given step j, the next point inserted into $\mathcal{V}_j$ can be to the right or left. $\bar{\lambda}^-_{j+1}$ and $\bar{\lambda}^+_{j+1}$ correspond to the case where the stencil inserted is to the left and right, respectively.

$$\begin{cases} \bar{\lambda}^-_{j+1} = \bar{\lambda}_{j+1} & \text{with } \mathcal{V}_{j+1} = \{x_p\} \cup \mathcal{V}_j \\ \bar{\lambda}^+_{j+1} = \bar{\lambda}_{j+1} & \text{with } \mathcal{V}_{j+1} = \mathcal{V}_j \cup \{x_q\}. \end{cases} \tag{4.79}$$

As a reminder, x_p and x_q are the mesh points immediately to the left and right of $\mathcal{V}_j$. Given $\mathcal{V}_j$, let μ^l_j be the number of points to the left of x_i and μ^r_j the number of points to the right. Below we introduce an algorithm for DBI and PPI based on the procedures introduced above.

Input: $\{x_i\}_{i=0}^n$, $\{u_i\}_{i=0}^n$, $\{\tilde{x}_i\}_{i=0}^{\tilde{n}}$, ϵ_0 and d. **Output:** $\{\tilde{u}_i\}_{i=0}^{\tilde{n}}$.

1. Select an interval $[x_i, x_{i+1}]$. Let $\mathcal{V}_0 = \{x_i, x_{i+1}\} = \{x^l_0, x^r_0\}$.

2. If $\sigma_{i-1}\sigma_{i+1} < 0$ or $\sigma_{i-1}\sigma_i < 0$, then the interval I_i has a hidden local extremum. For the boundary intervals, we assume that the divided differences to the left and right have the same sign.

3. Compute u_{min} and u_{max} using Equations (4.46) and (4.47).

4. Compute m_r and m_ℓ based on Equations (50) and (51) or Equations (72) and (73). For DBI, set $m_r = 1$ and $m_\ell = 0$.

5. Given a stencil $\mathcal{V}_j$,

   - if $B^-_{j+1} \le \bar{\lambda}^+_{j+1} \le B^+_{j+1}$ and $B^-_{j+1} \le \bar{\lambda}^-_{j+1} \le B^+_{j+1}$
     - if $\mu^l_j < \mu^r_j$ then insert a new stencil point to the left;
     - else if $\mu^l_j > \mu^r_j$ then insert a new stencil point to the right;

  - else insert a new stencil point to the right if $|\bar{\lambda}^l_{j+1}| \geq |\bar{\lambda}^r_{j+1}|$, otherwise insert a new point to left;

- else if $B^-_{j+1} \leq \bar{\lambda}^-_{j+1} \leq B^+_{j+1}$, then insert a new stencil point to the left;
- else if $B^-_{j+1} \leq \bar{\lambda}^+_{j+1} \leq B^+_{j+1}$, then insert a new stencil point to the right;

6. This process (Step 3) iterates until the halting criterion that the ratio of divided differences lies outside the required bounds stated above or the stencil has $d+1$ points, with d being the target degree for the interpolant.

7. Evaluate the final interpolant $U^l(x)$ (for DBI) or $U^p(x)$ (for PPI) at the output points $\tilde{x}_i$ that are in I_i.

8. Repeat Steps 1–7 for each interval in the input 1D mesh.

At the left and right boundary intervals, there are no mesh points beyond the boundaries to calculate σ_{i-1} and σ_{i+1}, respectively. At both boundaries, σ_{i-1} is set to σ_{i+1} ($\sigma_{i-1} = \sigma_{i+1}$) to ensure that no new extrema are introduced. At the boundary and nearby boundary intervals, the algorithm allows for hidden local extrema to be recovered. For example, if the right boundary interval has a hidden extremum $\sigma_{i-1}\sigma_i < 0$ (from Step 2) then the algorithm will relax the bounds on the interpolant and allow for the extremum to be recovered.

4.5 Numerical Experiments

In this section, we present both numerical experiments that demonstrate the properties of our proposed methods. These experimental studies use the PCHIP, DBI, and PPI methods. The test functions used here are taken from test problems 1, 2, 7, and 10 in [77]. A full suite of test problems has been undertaken by the authors in [77]. In that study, nine test problems are used with both uniform and nonuniform Legendre-Gauss-Lobatto (LGL) meshes. The Legendre-Gauss-Lobatto mesh consists of uniform elements with eight LGL quadrature nodes [36] inside each element. The integrals in the L^2-norm calculation are approximated using the trapezoid rule with 10^4 uniformly spaced points. The parameter ϵ_0 is set to 0.01 to allow the interpolant in each interval to grow beyond the data in a bounded way.

For various problems, including all the examples below, a standard Lagrange interpolant leads to large oscillations and negative values. The ENO and WENO methods reduce the oscillations, but they do not address the issue of preserving data boundedness or positivity. The DBI and PPI methods resolve both issues. The numerical experiments compare the DBI and PPI methods against the widely used PCHIP method, and show approximation errors using the algorithm described in Section 4.4.2.

4.5.1 1D Example: Runge Function

Our first example uses the Runge [23] function, defined as follows:

$$f_1(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1]. \tag{4.80}$$

Approximating the Runge function via a standard global polynomial using the set of points provided for the experiment leads to large oscillations and negative values.

Tables 4.1 and 4.2 show L^2-errors and convergence rates when approximating the Runge function $f_1(x)$ using the uniform and LGL meshes. For the approximations in Table 4.1, we use the PCHIP, DBI, and PPI methods with a target polynomial degree $d = 3$; whereas in Table 4.2, we use the DBI and PPI methods with the target polynomial degree varying from $d = 1$ to $d = 16$. The results in Table 4.1 show that the DBI and PPI methods lead to smaller errors and larger convergence rates compared to PCHIP for N larger than 17 in both the uniform and LGL mesh examples. For $N = 17$, the PCHIP approach leads to smaller errors. For higher polynomial degrees, the PPI method gives better results compared to the DBI and PCHIP, as demonstrated in Table 4.2. These results demonstrate that the PPI method is a suitable approach for interpolating data from one mesh to another when the underlying function is similar to the Runge function.

Table 4.1: L^2-errors and rates of convergence when using the PCHIP, DBI, and PPI methods to approximate the function $f_1(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N represents the number of input points used to build the approximation. The approximation functions for the DBI and PPI methods are cubic interpolants.

N	PCHIP	Rate	DBI	Rate	PPI	Rate
			Uniform Mesh			
17	7.15E-03	–	1.01E-02	–	1.01E-02	–
33	1.91E-03	1.99	1.21E-03	3.20	1.59E-03	2.78
65	3.70E-04	2.42	9.64E-05	3.73	1.12E-04	3.92
129	6.79E-05	2.47	6.29E-06	3.98	6.29E-06	4.20
257	1.22E-05	2.49	3.94E-07	4.02	3.94E-07	4.02
			LGL Mesh			
17	4.75E-03	–	8.36E-03	–	8.38E-03	–
33	1.30E-03	1.96	1.84E-03	2.28	1.84E-03	2.28
65	2.86E-04	2.23	2.05E-04	3.24	2.05E-04	3.24
129	5.81E-05	2.32	1.17E-05	4.17	1.17E-05	4.17
257	1.15E-05	2.35	1.04E-06	3.51	1.04E-06	3.51

For $N = 17$ in this example, the higher order terms added when going from $\mathcal{P}_8$ to $\mathcal{P}_{16}$ increase the L^2-error norms. These results indicate that resolution for $N = 17$ is not sufficient to see polynomial convergence when going from $\mathcal{P}_8$ to $\mathcal{P}_{16}$. The L^2-errors norms decrease with larger values of N.

Fig. 4.3 shows the errors found when approximating the Runge function $f_1(x)$ with PCHIP, DBI, and PPI. The top and bottom plots in Fig. 4.3 show the absolute errors when approximating the Runge example using $N = 33$ and $N = 129$ uniformly spaced points, respectively. The target polynomial degree is set to $d = 8$ for both the DBI and PPI methods and $\epsilon_0 = 0.01$. The errors around the middle of the domain dominate the overall error. The relaxed nature of the PPI method allows for higher degree interpolants compared to the DBI and PCHIP, which leads to better approximations, as shown in the bottom plots in Fig. 4.3.

4.5.2 1D Example: Smoothed Heaviside Function

This 1D example uses an analytic approximation of the Heaviside function defined as

$$f_2(x) = \frac{1}{1+e^{-2kx}}, \quad k = 100, \text{and } x \in [-0.2, 0.2]. \tag{4.81}$$

A polynomial approximation of $f_2(x)$ is challenging because of the large solution gradient around $x = 0$. Attempts to use a standard polynomial approximation for this function result

Table 4.2: L^2-errors and rates of convergence when using the DBI and PPI methods to approximate the function $f_1(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N represents the number of input points used to build the approximation. The interpolants are in $\mathcal{P}_j$, where j is the target polynomial degree.

	Uniform Mesh				LGL Mesh			
N	DBI		PPI		DBI		PPI	
	L^2-error	Rate	L^2-error	Rate	L^2-error	Rate	L^2-error	Rate
				$\mathcal{P}_1$				
17	2.16E-02	–	2.16E-02	–	1.69E-02	–	1.69E-02	–
33	6.02E-03	1.92	6.02E-03	1.92	5.84E-03	1.60	5.84E-03	1.60
65	1.52E-03	2.03	1.52E-03	2.03	1.66E-03	1.86	1.66E-03	1.86
129	3.82E-04	2.02	3.82E-04	2.02	5.80E-04	1.53	5.80E-04	1.53
257	9.56E-05	2.01	9.56E-05	2.01	1.52E-04	1.94	1.52E-04	1.94
				$\mathcal{P}_4$				
17	8.34E-03	–	7.02E-03	–	6.55E-03	–	6.54E-03	–
33	5.91E-04	3.99	5.91E-04	3.73	7.62E-04	3.24	7.62E-04	3.24
65	4.26E-05	3.88	2.39E-05	4.73	5.30E-05	3.93	5.29E-05	3.94
129	2.68E-06	4.03	8.00E-07	4.95	3.44E-06	3.99	3.44E-06	3.99
257	8.63E-08	4.99	2.55E-08	5.00	8.88E-08	5.31	8.87E-08	5.31
				$\mathcal{P}_8$				
17	4.61E-03	–	3.11E-03	–	3.49E-03	–	4.40E-03	–
33	4.43E-04	3.53	1.51E-04	4.56	1.76E-04	4.50	1.76E-04	4.85
65	3.67E-05	3.67	1.05E-06	7.33	3.25E-06	5.89	3.01E-06	6.00
129	2.56E-06	3.88	3.10E-09	8.50	5.64E-08	5.91	8.82E-09	8.51
257	8.24E-08	4.99	6.80E-12	8.88	3.51E-09	4.03	3.96E-11	7.84
				$\mathcal{P}_{16}$				
17	4.34E-03	–	3.44E-03	–	4.89E-03	–	5.01E-03	–
33	4.21E-04	3.52	4.85E-05	6.43	1.18E-04	5.62	1.17E-04	5.67
65	3.67E-05	3.60	5.92E-08	9.89	1.22E-06	6.75	9.40E-08	10.51
129	2.56E-06	3.88	4.21E-12	13.94	5.57E-08	4.50	1.02E-11	13.32
257	8.24E-08	4.99	2.18E-16	14.32	3.51E-09	4.01	5.04E-16	14.38

in oscillations and negative values.

Tables 4.3 and 4.4 show L^2-errors and convergence rates when approximating the smoothed Heaviside function $f_2(x)$ using the uniform and LGL meshes. Table 4.4 shows that for a target polynomial of degree $d = 3$, the errors for PCHIP, DBI, and PPI are comparable. When the target degree increases from $d = 1$ to $d = 16$, the errors for the DBI and PPI methods decrease, as shown in Table 4.4. Overall, the errors from the DBI and PPI methods are comparable with DBI yielding slightly smaller errors than PPI. The uniform mesh leads to better approximation results compared to the LGL mesh. These results demonstrate that the DBI and PPI methods are both suitable for mapping data between different meshes

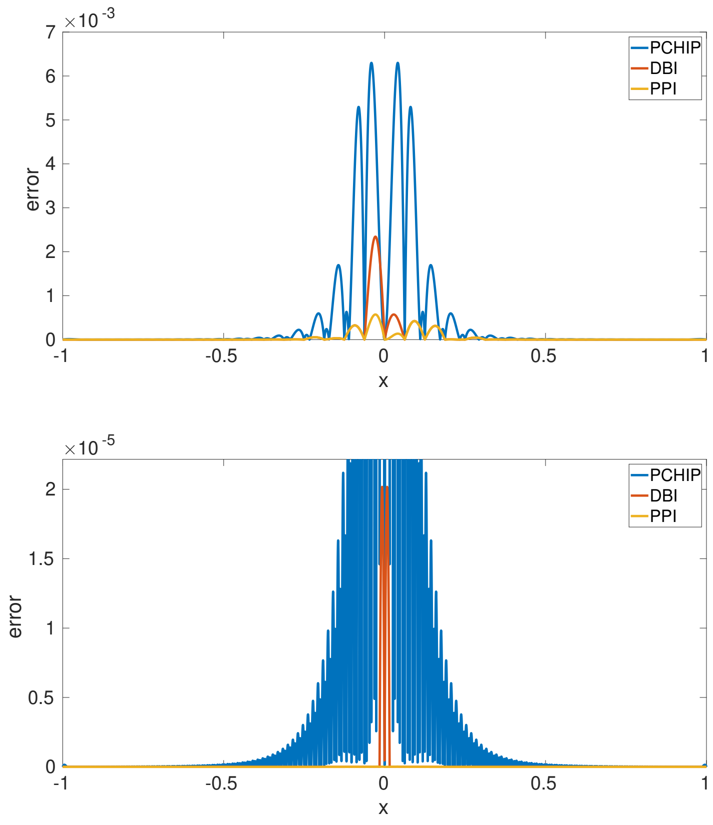


Fig. 4.3: Error plots when approximating $f_1(x)$. The top and bottom error plots are obtained from approximating $f_1(x)$ with $N = 33$ and $N = 129$ uniformly spaced points, respectively. The target polynomial degree is set to $d = 8$ and $\epsilon_0 = 0.01$.

when the underlying function is similar to the smoothed Heaviside function.

Fig. 4.4 provides examples of error plots for approximating the smoothed Heaviside function $f_2(x)$ with PCHIP, DBI, and PPI. The top and bottom plots in Fig. 4.4 show the absolute error when approximating the smoothed Heaviside function $f_2(x)$ using $N = 33$ and $N = 129$ uniformly spaced points, respectively. The global error is dominated by the errors in the region with the steep gradient around $x = 0$. The errors from DBI and PPI are identical for $N = 129$ because the stencil selected by both methods are the same around the region with the steep gradients. Away from the steep gradient, the DBI and PPI methods use different stencils, but the errors in those regions are negligible compared to the errors around $x = 0$.

Table 4.3: L^2-errors and rates of convergence when using the PCHIP, BDI, and PPI methods to approximate the function $f_2(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N represents the number of input points used to build the approximation. The approximation functions for the DBI and PPI methods are cubic interpolants.

N	PCHIP	Rate	DBI	Rate	PPI	Rate
	Uniform Mesh					
17	2.02E-02	–	1.97E-02	–	1.97E-02	–
33	3.38E-03	2.70	3.53E-03	2.59	3.54E-03	2.59
65	3.59E-04	3.31	5.00E-04	2.88	5.00E-04	2.89
129	4.21E-05	3.13	4.51E-05	3.51	4.51E-05	3.51
257	5.12E-06	3.06	3.01E-06	3.93	3.01E-06	3.93
	LGL Mesh					
17	3.65E-03	–	5.38E-03	–	5.38E-03	–
33	1.45E-03	1.39	1.55E-03	1.88	1.56E-03	1.86
65	4.07E-04	1.87	6.49E-04	1.28	6.49E-04	1.30
129	8.85E-05	2.23	9.77E-05	2.76	9.77E-05	2.76
257	1.38E-05	2.70	9.06E-06	3.45	9.06E-06	3.45

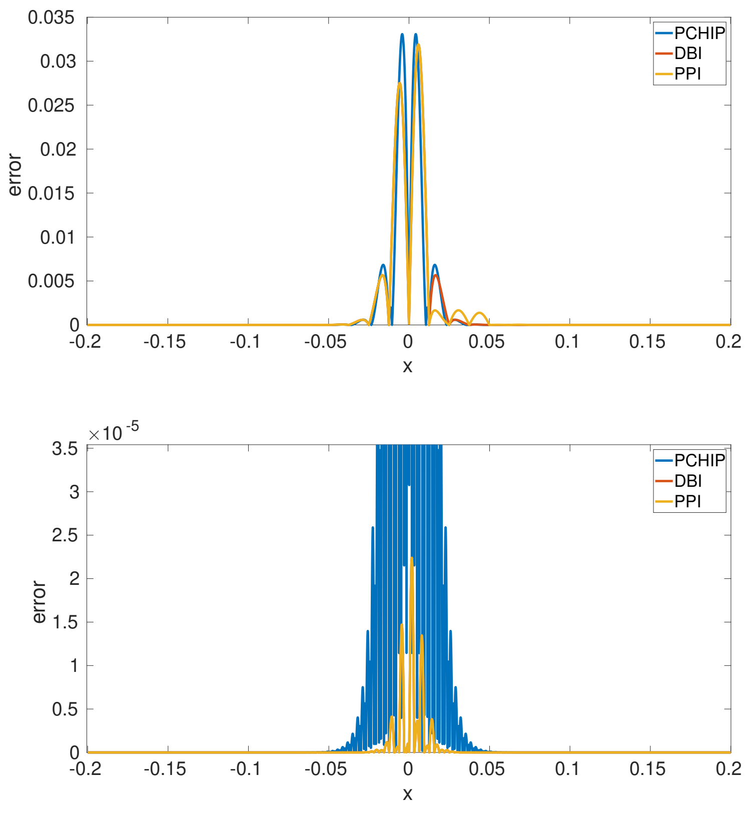


Fig. 4.4: Error plots when approximating $f_2(x)$. The top and bottom error plots are obtained from approximating $f_1(x)$ with $N = 33$ and $N = 129$ uniformly spaced points, respectively. The target polynomial degree is set to $d = 8$, $\epsilon_0 = 0.01$, and $\epsilon_1 = 1.0$.

Table 4.4: L^2-errors and rates of convergence when using the DBI and PPI methods to approximate the function $f_2(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N represents the number of input points used to build the approximation. The interpolants are in $\mathcal{P}_j$, where j is the target polynomial degree.

	Uniform Mesh				LGL Mesh			
N	DBI		PPI		DBI		PPI	
	L^2-error	Rate	L^2-error	Rate	L^2-error	Rate	L^2-error	Rate
	$\mathcal{P}_1$							
17	2.89E-02	–	2.89E-02	–	8.58E-03	–	8.58E-03	–
33	7.69E-03	1.99	7.69E-03	1.99	5.24E-03	0.74	5.24E-03	0.74
65	1.80E-03	2.14	1.80E-03	2.14	2.20E-03	1.28	2.20E-03	1.28
129	4.58E-04	2.00	4.58E-04	2.00	8.08E-04	1.47	8.08E-04	1.47
257	1.15E-04	2.00	1.15E-04	2.00	2.01E-04	2.01	2.01E-04	2.01
	$\mathcal{P}_4$							
17	2.23E-02	–	2.23E-02	–	5.24E-03	–	5.24E-03	–
33	4.09E-03	2.56	4.10E-03	2.56	1.10E-03	2.36	1.11E-03	2.34
65	3.05E-04	3.83	3.05E-04	3.84	3.06E-04	1.88	3.07E-04	1.89
129	1.35E-05	4.55	1.35E-05	4.55	3.32E-05	3.24	3.32E-05	3.24
257	4.71E-07	4.87	4.71E-07	4.87	1.17E-06	4.85	1.17E-06	4.85
	$\mathcal{P}_8$							
17	2.08E-02	–	2.08E-02	–	4.87E-03	–	4.68E-03	–
33	3.36E-03	2.75	3.33E-03	2.76	8.71E-04	2.59	7.84E-04	2.69
65	1.38E-04	4.70	1.38E-04	4.69	7.57E-05	3.60	1.24E-04	2.72
129	1.22E-06	6.90	1.22E-06	6.90	2.17E-06	5.19	2.17E-06	5.90
257	4.44E-09	8.15	4.44E-09	8.15	1.95E-08	6.83	1.95E-08	6.83
	$\mathcal{P}_{16}$							
17	2.00E-02	–	2.00E-02	–	4.83E-03	–	4.64E-03	–
33	2.93E-03	2.90	2.91E-03	2.91	7.38E-04	2.83	7.27E-04	2.80
65	9.17E-05	5.11	9.17E-05	5.10	7.60E-05	3.35	9.41E-05	3.02
129	1.70E-07	9.17	1.70E-07	9.17	2.88E-07	8.14	2.88E-07	8.45
257	2.64E-11	12.73	2.64E-11	12.73	5.39E-11	12.45	5.39E-11	12.45

4.5.3 Hidden Local Extrema Examples

This numerical study demonstrates the ability of the PPI method to recover hidden extrema. The study uses the Runge functions $f_1(x)$ with a uniform mesh. The uniformly spaced mesh points are constructed such that the extremum at $x = 0$ lies inside of an interval. Tables 4.5 shows L^2-error norms and convergence rates when approximating $f_1(x)$ from Equations (4.80). Overall, the PPI method achieves high-order accuracy when approximating the Runge functions from data with and without hidden extrema. The results from both tables show that the PPI method leads to smaller errors and larger convergence rates compared to the DBI method. The DBI approach uses a bounded interpolant that

Table 4.5: L^2-errors and rates of convergence when using the DBI and PPI methods to approximate the function $f_1(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0. The uniform mesh used to build the approximation is constructed with N points . The interpolants are in $\mathcal{P}_j$, where j is the target polynomial degree.

N	DBI		PPI	
	L^2-error	Rate	L^2-error	Rate
	$\mathcal{P}_1$			
16	2.81E-02	–	2.81E-02	–
32	6.41E-03	2.13	6.41E-03	2.13
64	1.57E-03	2.03	1.57E-03	2.03
128	3.88E-04	2.02	3.88E-04	2.02
256	9.63E-05	2.01	9.63E-05	2.01
	$\mathcal{P}_4$			
16	2.81E-02	–	1.37E-02	–
32	4.72E-03	2.57	6.85E-04	4.32
64	8.14E-04	2.54	2.57E-05	4.73
128	1.42E-04	2.52	8.32E-07	4.95
256	2.49E-05	2.51	2.60E-08	5.00
	$\mathcal{P}_8$			
16	2.74E-02	–	1.07E-02	–
32	4.69E-03	2.55	2.06E-04	5.70
64	8.14E-04	2.53	1.19E-06	7.43
128	1.42E-04	2.52	3.32E-09	8.49
256	2.49E-05	2.51	7.04E-12	8.88
	$\mathcal{P}_{16}$			
16	2.75E-02	–	1.02E-02	–
32	4.69E-03	2.55	1.43E-04	6.16
64	8.14E-04	2.53	7.18E-08	10.96
128	1.42E-04	2.52	4.74E-12	13.89
256	2.49E-05	2.51	2.77E-16	14.06

fails to represent the extremum at $x = 0$, whereas the relaxed nature of the PPI approach allows for a more accurate representation of the extremum. In the case of DBI, as the target polynomial degree increases from $\mathcal{P}_4$ to $\mathcal{P}_{16}$, the errors and convergence rates do not improve because the global error is dominated by the local error in the interval with the hidden extremum. The DBI approach achieves only an $O(h^{2.5})$ accuracy as opposed to the PPI method, that achieves the same high accuracy regardless of whether or not the extremal values are data points. These results highlight the advantage of the PPI method over the DBI method for recovering hidden extrema from data.

4.6 Summary and Discussion

In this chapter, we present both an algorithm and theoretical foundations for sufficient conditions to ensure data boundedness and positivity on any set of mesh points via a Newton polynomial formulation. The one-dimensional PPI and DBI methods analyzed herein are building blocks that have been extended to multidimensional PPI and DBI methods using tensor-products. This extension consists of successively applying the one-dimensional PPI or DBI method on each dimension to generate the multidimensional results.

The DBI method imposes restrictions on the ratio of divided differences to ensure that the interpolants are bounded by the input data. The proof of the DBI approach presents new challenges because the configuration of mesh points may not exhibit a regular structure. The PPI method starts from the DBI method and relaxes the bounds on the ratio of divided differences, thereby allowing the interpolants to grow beyond the data as needed while remaining positive. The positive interpolant is further bounded by the parameters u_{min} and u_{max} to remove undesirable oscillations that may potentially degrade the approximation. The proofs of both the DBI and PPI approaches rely on the results from Lemma 4.3.1, which consist of using the definition of B_j^+, B_j^- to arrive at the bounds $B_{j-1}^- \leq \bar{\lambda}_{j-1}\delta_j \leq B_{j-1}^+$. The proofs from Theorems 4.3.2 and 4.4.1 use Lemma 4.3.1 to show that $0 \leq S_n(x) \leq 1$ for the DBI method and $m_\ell \leq S_n(x) \leq m_r$ for the PPI method.

Note that one observation we have made is that the PPI method uses higher order interpolants compared to the DBI method. Relaxing the bounds on the ratio of divided differences increases the range of polynomial degrees that meet the desired requirement. The numerical results, in Tables 4.1-4.4, indicate that the DBI or PPI methods provided herein are appropriate for ensuring data boundedness or positivity preservation, and both methods converge as the interpolant degree and resolution increase. Fig. 4.1 demonstrates that enforcing positivity alone may not be sufficient to remove large oscillations. We resolve this issue by bounding the positive polynomial with u_{min} and u_{max}, which are determined based on user-supplied values, such as $\epsilon_0 = 0.01$ for the numerical examples in Section 5.5. In addition, Fig. 4.2 demonstrates that for an interval I_i where there exists a local extremum, the PCHIP and DBI methods truncate the extremum whereas the PPI method leads to a better approximation of the extremum. The different results demonstrated that the PPI method is able to produce high-order accurate approximations in examples with and without a hidden extremum.

CHAPTER 5

NUMERICAL TESTING OF THE POSITIVITY-PRESERVING AND DATA-BOUNDED INTERPOLATION

Introduction

This chapter is concerned with the numerical testing of new interpolation algorithm that has been introduced in Chapter 4 for positivity preservation when mapping solution values between structured meshes. The theoretical basis for the algorithm builds on the data-bounded work of Berzins [5] to develop a new data-bounded and positivity-preserving methods for both evenly- and unevenly-spaced structure meshes. The new data-bounded interpolation (DBI) method in introduced in the previous chapter relaxes conditions for data boundedness, which gives greater accuracy than the conditions used in [5]. Ouermi et al. [82] further extended the DBI method to give a new positivity-preserving interpolation (PPI) method. The application of these new methods to numerical weather prediction examples is described in [78]. In this chapter, a number of possible alternative interpolation schemes are introduced. A representative sample of such methods is compared against the new approaches on a number of different test functions, including smooth, C^0, discontinuous, and steep-gradient functions. The comparison undertaken focuses on how accurately the different methods are able to represent this underlying set of test functions. In addition, a representative weather model problem is considered. Overall, it will be shown that the new methods are well suited for function approximation and mapping data values between meshes for numerical weather examples. The generality of this approach suggests that these methods also have application to other problems for which preserving positivity is important.

5.1 Examples of Existing Interpolation Methods

This section highlights several approaches that have been developed to address the need for data-bounded, positivity-preserving, and shape-preserving interpolation. Although this selection of methods is not all-inclusive, it is intended to illustrate the main types of polynomial-based approaches.

5.1.1 Cubic Splines

In computer-aided design (CAD), graphics and visualization, significant contributions have been made to develop and advance shape-preserving methods. Many of the approaches for shape-preservation are based on cubic splines. In [89] and [90] Schmidt and Heß introduced positive interpolation methods using rational quadratic and cubic splines respectively. Necessary and sufficient conditions for positivity are provided for both the rational quadratic and cubic interpolants. These conditions impose some restrictions on the values of the first derivatives at each node. As in [41], both approaches lead to multiple solutions, and the one with the minimal curvature is selected. The work in [54], [53], and [47] presented positivity-preserving interpolation methods that rely on rational cubic splines. The C^2 continuity in [54] is obtained by solving a tridiagonal system of linear equations. All three methods introduce free parameters that are used to derive and enforce conditions for positivity. Butt and Brodlie [10] provide a method for constructing C^1 cubic Hermite splines. This method is dependent on the availability of values of first derivatives at the nodes, which may not be available in practice. Positivity is enforced by imposing a bound on the values of the derivatives. In the case where bounds on the derivatives are not met, one or two knots are inserted to ensure that the constructed spline is positive. Perhaps the most widely used approach for preserving monotonicity in many applications is PCHIP by Fritch and Carlson [33] who derived necessary and sufficient conditions for monotone cubic interpolation, and provided an algorithm for building a piecewise cubic approximation from data. This algorithm calculates the values of the first derivatives at the nodes based on the necessary and sufficient conditions.

5.1.2 Quartic and Quintic Splines

Although many shape-preserving interpolation methods are cubic or lower order, a number of approaches target higher order interpolants, with an emphasis on quartic or quintic polynomial approximations. The work in [26] and [25] presents geometric or visual continuity G^1 and G^2 continuous shape-preserving interpolation using Pythagorean-Hodograph quintic splines curves. This approach uses Bernstein basis functions and a parametric representation of the interpolant in each interval. A sufficient condition for shape preservation is constructed based on free angular parameters that influence the shape of the curve in each interval. The appropriate angular parameters are selected based on the cubic-cubic (CC) criterion introduced in [24]. The G^2 case requires a tridiagonal solve and use of a Newton-Raphson iteration, which potentially affects the computational performance.

Hussain et al. [46] and Hussain et al. [48] introduced C^2 rational quintic interpolation interpolation approaches that preserve positivity. These rational quintic functions are constructed with free parameters that are used to enforce positivity. Both methods require the approximation of values of first and second derivatives at the nodes if these derivatives are not available. In addition, the rational quintic interpolation methods in [46] and [48] have a $O(h^3)$ order of accuracy.

Heß and Schmidt [41] developed interpolation schemes that preserve positivity and monotonicity using C^2 quartic and quintic splines. Positivity and monotonicity are achieved by imposing some restrictions on the values of the first and second derivatives at each node. This approach leads to a potentially infinite number of solutions that meet the required conditions. Of these solutions, the solution with minimal curvature is selected using global minimization. The global nature of the minimization makes the algorithm challenging to parallelize and may have an impact on computational performance. MQS [64] is an example of a monotonic quintic spline method that was developed by Lux et al. [64] who built on the work of Heß and Schmidt [90] and Ulrich and Watson [104]. This algorithm uses the sufficient conditions from [90] to check for monotonicity and the work in [104] to adjust values of the first and second derivatives to ensure monotonicity. This method requires the values of the first and second derivatives at the nodes, which may not be available in practice. In this report, the first and second derivatives are approximated using a fourth-order finite difference stencil based on [30].

5.1.3 SPS and B-Spline Higher Order Splines

Costantini [15], [14] developed a C^1 and C^2 shape-preserving spline (SPS) interpolation method using Berstein-Bezier polynomials of an arbitrary degree. The desired shape property is obtained by imposing restrictions on the value of the first derivatives at the nodes. The Bezier coefficients for each spline are derived from a linear function. For a given interval, the coefficients of the Berstein-Bezier polynomial interpolant are selected from a linear function. The first derivatives at the nodes are calculated such that the sufficient conditions for shape preservation given in [15] are met. The approximation of the first derivatives at the node is third-order accurate. In addition, Theorem 9 of [13] shows that the spline method presented in [15] [14] has an error of $O(h^4)$. More details on the construction of the splines, an algorithm and a software package for the SPS method can be found in [15] [14]. In addition to the positivity-preserving approaches, conventional B-splines [19] are also used here. Although the B-spline approach does not preserve positivity, many of the approaches mentioned in this work are based on B-splines, and so the use of unmodified B-splines provides an accuracy check on the other spline methods.

5.1.4 DBI and PPI Methods

The numerical solution of partial differential equations (PDEs), particularly hyperbolic equations, is an another area in which various methods have been developed to enable data-bounded and positivity-preserving approximations. In order to preserve positivity in discontinuous Galerkin (dG) schemes, Zhang et al. [110], [113], [111] and Light et al. [60] introduced a linear rescaling of polynomials that ensures that the evaluation of the polynomial at the quadrature points is positive. In addition, this linear rescaling of the polynomial conserves mass. The polynomial rescaling, however, does not address the case of interpolating between different meshes, which is the primary focus of this work. Harten et al. [39] developed an essentially nonoscillatory (ENO) piece-wise polynomial reconstruction that is suitable for interpolating between different meshes. ENO methods adaptively build an interpolant based on Newton divided differences and can help remove Gibbs-like effects but do not guarantee positivity. A weighted combination of ENO schemes, (WENO) has been used by Zhang et al. [112] and many others.

A DBI method was developed by Berzins using evenly spaced meshes from ENO methods [5]. This method was extended by the authors in [82], [78] to work for both evenly and unevenly spaced meshes and, more importantly, to the PPI method. Ouermi et al. [82] relaxed the conditions for data boundedness which gives greater accuracy compared to the conditions used in [5]. Both the improved DBI and the new PPI methods are used in this chapter. The PPI method further extends the DBI method by relaxing the bounds on the ratio of divided differences and so allows the interpolant to grow beyond the data, while still remaining positive. For a given interval, the DBI and PPI methods successively select stencil points until the required bounds are violated or $d+1$ points are selected, with d being the target degree of the interpolant. In addition to enforcing data boundedness and positivity, the algorithm in [78] uses a user-supplied parameter st to guide the stencil construction procedure. When adding the next point to both the right or left of the current stencil meets the requirements for data boundedness or positivity, the algorithm makes the selection based on the three cases below.

- If $st = 1$, the algorithm chooses the point with the smallest divided difference, as in the ENO stencil.
- If $st = 2$, the point to the left of the current stencil is selected if the number of points to the left of x_i is smaller than the number of points to right. Similarly, the point to the right is selected if the number of points to the right of x_i is smaller than the number of points to the left. When both the number of points to right and left are the same, the algorithm chooses the point with the smallest ratio of divided differences.
- If $st = 3$, the algorithm chooses the point that is closest to the starting interval I_i.

Enforcing positivity alone may still lead to undesirables oscillations. To address this limitation, the algorithm provides the parameters ϵ_0 and ϵ_1 that are used to impose an upper and lower bound for each interpolant. For each interval I_i, the bounds are constructed using the parameters ϵ_0 and ϵ_1, and the data values u_i and u_{i+1}. Both the DBI and PPI methods and the algorithm details are described in [78] with numerical examples pertaining to NWP.

5.2 Comparison Methodology

5.2.1 Compared Methods

The numerical experiments in this report use the PCHIP [33], MQS [64], SPS [15], [14], B-splines [19], the improved DBI [5], and the new PPI methods [82]. These methods are available as follows:

PCHIP: The version of the PCHIP algorithm used in this report is implemented in Fortran 90 and can be found at `https://people.sc.fsu.edu/~jburkardt/f_src/pchip/pchip.html`.

MQS: The method of Lux et al. [64] is an example of a method for monotonic quintic splines. The algorithm is implemented in Python3 and can be found `https://github.com/tchlux/papers/tree/master/%5B2019-11%5D_HPC_(quintic_spline)`.

SPS: Costantini [15], [14] introduced a high-order shape-preserving (monotonicity- and convexity-preserving) Spline (SPS) method using Berstein-Bezier polynomials of arbitrary degree. The SPS method is implemented in the BVSPIS software package in Fortran 77 and is available from ACM as Algorithm 770 [14] `https://dl.acm.org/action/downloadSupplement?doi=10.1145%2F264029.264059&file=770.gz&download=true`.

B-splines: PPPACK, a Fortran 90 library that evaluates piecewise polynomial functions, including cubic splines. The original FORTRAN77 library is by Carl de Boor [19]. The package is available from `https://people.sc.fsu.edu/~jburkardt/f_src/pppack/pppack.html`.

HPPIS: The DBI and PPI methods have been developed based on the theory and algorithm in [82], [78]. The software and implementation details can be found in [78].

5.2.2 Comparison Criteria

The three steps outlined below are used to compare the different methods when used to approximate smooth and nonsmooth functions. The errors are measured in a discrete approximation to the L^2-error norm.

- The first step consists of demonstrating that the various schemes preserve positivity for each of the test functions used. In addition, this step is used to show that a standard polynomial interpolation method does not guarantee positivity.
- The second step experimentally investigates the convergence of the various schemes

when using smooth functions. This step tests the ability of the different methods to accurately represent smooth functions as the resolution increases. For the shape-preserving spline (SPS) [15], [14], DBI and PPI methods, we also investigate the approximation accuracy obtained with varying interpolant polynomial degrees.

- The third step focuses on the ability of the different methods to represent a set of challenging test functions with large gradients and/or discontinuities. This step represents situations often encountered in computational science problems, such as mapping between physics and dynamics meshes in NEPTUNE.

5.3 Positivity-Preserving Interpolants

Preserving positivity while maintaining accuracy is perhaps the key property needed when mapping from one mesh to another in NEPTUNE and similar applications. This section compares the PCHIP, MQS, SPS, DBI, and PPI against a standard interpolation method using five examples. The standard polynomial interpolation approach (STD) uses the points in each element to build a standard Lagrange interpolant for that element. In each of the examples, the different interpolants are constructed using:

- a uniform mesh that is constructed using uniformly spaced points. In this mesh, all the elements have the same size and the nodes are uniformly spaced inside each element.

In the figures presented in this section, the black and red plots represent the underlying function and its approximation using the different interpolation methods. Both the DBI and PPI methods use a mesh point selection method that favors a symmetric stencil about x_i by setting $st = 1$ with $\epsilon_0 = 0.01$ and $\epsilon_1 = 1.0$. The results in this section demonstrate that the PCHIP, MQS, DBI, SPS, and PPI methods preserve positivity, whereas the standard interpolation methods lead to oscillations and fail to preserve positivity.

5.3.1 Example I $f_1(x)$

This example uses the famous Runge function [23] defined as follows:

$$f_1(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1]. \tag{5.1}$$

Fig. 5.1 shows the different polynomial approximations for this function using 17 uniformly spaced points. The target polynomial degree for the standard interpolation, DBI, and PPI is set to $d = 16$. The standard polynomial interpolation approach, STD, does not preserve positivity with the uniform mesh and generates oscillations in both meshes. The PCHIP, MQS, DBI, SPS, and PPI methods preserve positivity.

5.3.2 Example II $f_2(x)$

The second example uses an analytic approximation of the Heaviside function defined as follows:

$$f_2(x) = \frac{1}{1 + e^{-2kx}}, \quad k = 100, \text{ and } x \in [-0.2, 0.2]. \tag{5.2}$$

A polynomial approximation of $f_2(x)$ is challenging because of the large gradient at about $x = 0$. Attempts to use a global polynomial approximation for this function result in

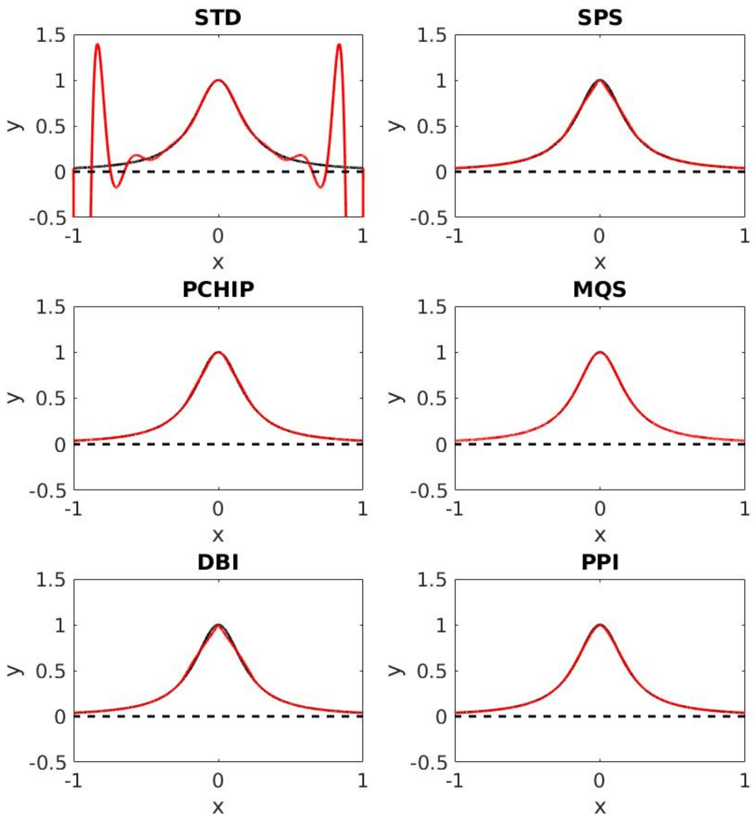


Fig. 5.1: Approximation of the Runge function with the $N = 17$ points that are uniformly distributed on the interval $[-1, 1]$. The parameters d, ϵ_0 and ϵ_1 are set to 16, 0.01, and 1.0, respectively.

unacceptable oscillations and negative values as observed in the Runge example above. Fig. 5.2 shows interpolations of $f_2(x)$ using uniform mesh of 17 points. Standard polynomial interpolation, DBI, and PPI are used with an interpolant of degree $d = 8$ for each interval. Standard polynomial interpolation fails to preserve positivity. The results demonstrate that the PCHIP, MQS, DBI, SPS, and PPI methods preserve positivity.

5.3.3 Example III $f_3(x)$

The third example uses a modified version of a function introduced by Tadmor and Tanner [102] and used by Berzins [5] in the context of DBI based upon uniform mesh points. The original function was modified by adding the value one to ensure that the function is positive over the interval $[-1, 1]$. The modified function is defined as

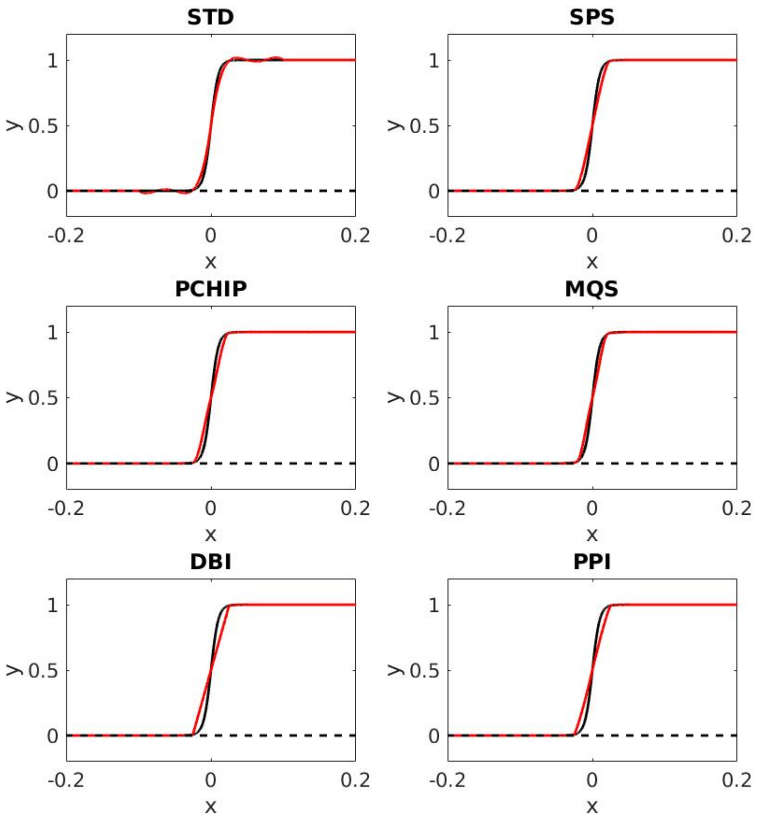


Fig. 5.2: Approximation of $f_2(x) = \frac{1}{1+e^{-2kx}}, \quad k = 100$, and $x \in [-0.2, 0.2]$, with $N = 17$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. The points are uniformly distributed, and the target polynomial degree for the DBI and PPI is $d = 8$.

$$f_3(x) = \begin{cases} 1 + \frac{2e^{2\pi(x+1)}-1-e^{\pi}}{e^{\pi}-1}, & x \in [-1, -0.5) \\ \\ 1 - sin\left(\frac{2\pi x}{3} + \frac{\pi}{3}\right), & x \in [-0.5, 1]. \end{cases} \tag{5.3}$$

This function is particularly challenging because of the discontinuity at $x = -0.5$. This example uses 17 uniformly spaced points. The target interpolant degree for the standard interpolation, DBI, and PPI methods is $d = 4$. Fig. 5.3 demonstrates that the interpolants built using the PCHIP, MQS, SPS, SPS and PPI methods remain positive whereas the standard polynomial interpolation approach fails to preserve positivity.

5.3.4 Example IV $f_4(x)$

This example consists of a function with multiple spikes defined as follows:

$$f_4(x) = 1.0 - \left| \frac{2}{\pi} arctan\left(\frac{sin(\pi \frac{x}{h})}{\delta}\right)\right|, \quad x \in [0, 1], \tag{5.4}$$

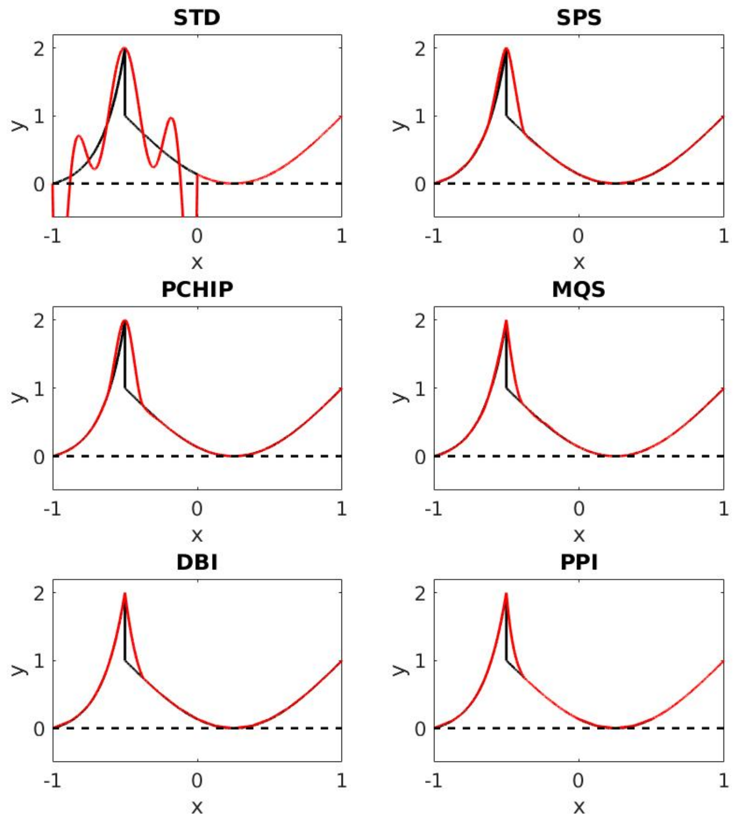


Fig. 5.3: Approximation of $f_3(x)$ with $N = 17$ points. The parameters d, ϵ_0, and ϵ_1 are set to 4, 0.01, and 1.0, respectively. The points are distributed uniformly over the interval $[-1, 1]$.

where h represent the element size, and $\delta = 0.01$. $f_4(x)$ depends on the element size h, and therefore, on the number of element in a given interval. At the element boundaries, $f_4(x)$ is C^0-continuous with large gradients of opposite signs. This example uses 33 points, four elements, and nine points in each element. The approximations in Fig. 5.4 use uniform points. The plots in Fig. 5.4 show the standard polynomial interpolation approach leads to oscillation and negative values, whereas the PCHIP, MQS, SPS, DBI, and PPI methods preserve positivity and remove the oscillations.

5.3.5 Example V $f_5(x)$

This example is constructed using the *tanh* function and by introducing C^0-continuities at the elements boundaries. The constructed function is defined as follows:

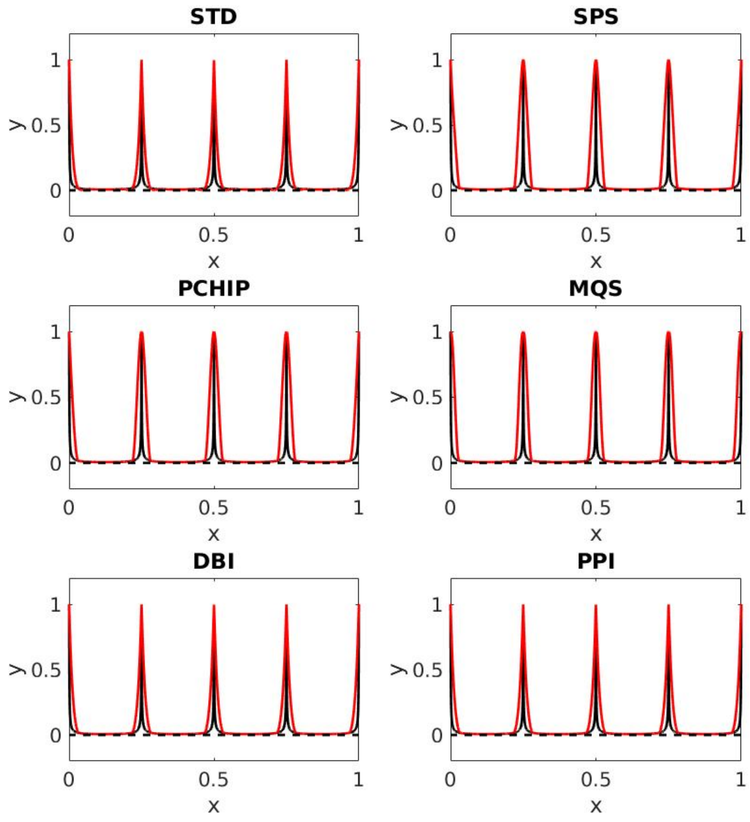


Fig. 5.4: Approximation of $f_4(x)$, with $N = 33$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01, and 1.0, respectively. The points are uniformly distributed, and the target polynomial degree for the DBI and PPI is $d = 8$.

$$f_5(x) = \begin{cases} tanh(xk) & \text{if } x \in [a, a+h] \\ 2tanh(xk) - tanh((a+h)k) & \text{if } x \in [a+h, a+2h] \\ 3tanh(xk) - tanh((a+h)k) - tanh((a+2h)k) & \text{if } x \in [a+2h, a+3h] \\ \quad \vdots \end{cases} \tag{5.5}$$

where the overall interval is $[-2, 0]$ with $a = -2$ and $k = 10$. h represents the size of each element. $f_5(x)$ depends on the element size h and, therefore, on the number of elements in a given interval. This example is built to mirror the C^0-continuity at the elements boundaries in the spectral element method used in NEPTUNE. In this example, the gradients at the elements boundaries are always positive, and are not as large as the ones in $f_4(x)$ from Example IV. The approximations shown in Fig. 5.5 uses 17 points. The plots in Fig. 5.5 show that the standard interpolation method does not preserve positivity and that the PCHIP, MQS, SPS, DBI, and PPI can be used to enforce positivity as required.

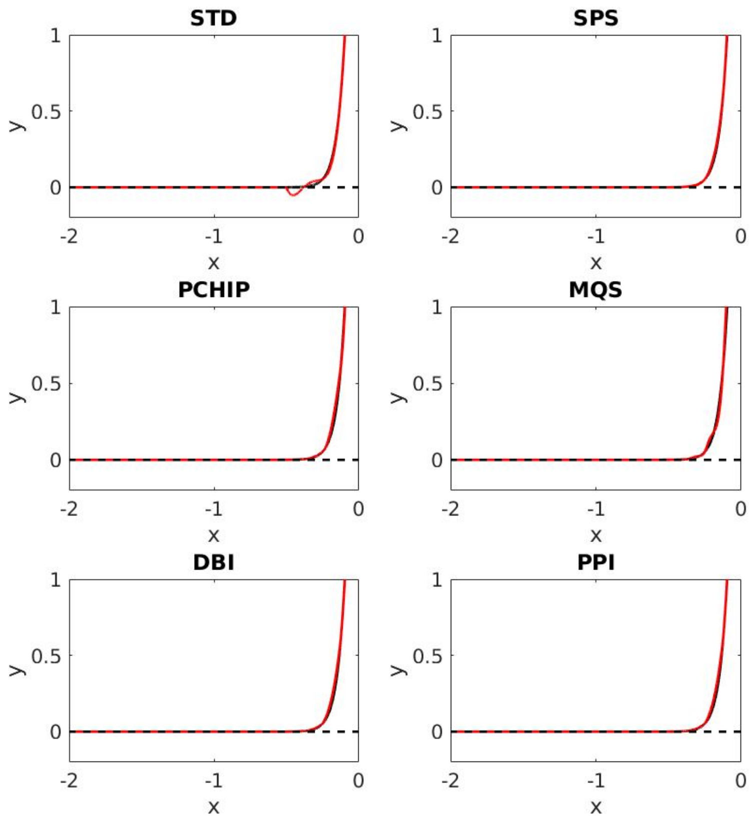


Fig. 5.5: Approximation of $f_5(x)$, with $N = 17$ points. The parameters ϵ_0, and ϵ_1 are set to 0.01 and 1.0, respectively. The points are uniformly distributed and the target polynomial degree for the DBI and PPI is $d = 4$.

5.4 Convergence

This section focuses on the second comparison criterion, which consists of evaluating the convergence of the different methods when applied to a smooth function. As NUMA [34], the dynamics part of NEPTUNE uses a spectral element method that has high-order accuracy, especially in smooth regions. It is important when interpolating solution values between dynamics and physics meshes for the interpolation scheme to not degrade the accuracy obtained from the spectral element method.

The test function

$$f_6(x) = 1 + sin(x), x \in [0, \pi] \tag{5.6}$$

is used to study the convergence of the different methods. $f_6(x)$ is infinitely smooth with no sharp gradients or discontinuities. These characteristics make $f_6(x)$ a suitable test function for evaluating which approach is a good choice for representing smooth functions. These experiments focus on the accuracy of the approximation as the resolution and the polynomial degree both increase.

Table 5.1 shows L^2-errors when approximating $f_6(x)$ using the different interpolation methods. In this experiment, the parameters ϵ_0, ϵ_1, and st are chosen to be 0.01, 1, and 1, respectively. In all cases, the L^2-error is estimated by sampling the error at 10000 equally spaced points in the interval and using trapezoidal quadrature. Table 5.2 shows the ratio, $e_{N_i}/e_{N_{i+1}}$ of the L^2-errors in Table 5.1 as the resolution increases. The DBI and PPI methods lead to smaller errors compared to the PCHIP, MQS, and SPS methods. As the average polynomial degree increases, the approximation using the DBI method does not improve because the global error is dominated by the local error from the intervals using lower degree interpolants compared to PPI. These results show that the conditions for data-boundedness may be more restrictive when it comes to enforcing positivity. The SPS method shows smaller errors compared to the other methods. Furthermore, as the polynomial degree increases, the accuracy of the approximation decreases. These results are consistent with those in [15] [14]. Costantini [13], [15] demonstrated that the SPS method is bounded by $O(h^4)$ and in the limit (as the spline degree increases) the spline tends to a linear interpolation. The B-spline and PPI methods have smaller L^2-errors compared to the other methods, and their accuracy improves as the polynomial degree increases. Table 5.2 shows that both methods have better convergence rates compared to PCHIP, MQS, and SPS. The

Table 5.1: L^2-errors when using the PCHIP, MQS, SPS, B-splines, DBI, and PPI methods to approximate the function $f_6(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the space of polynomials of degree j, with j being the target degree for each interval. The seventh and ninth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The input points are uniformly distributed over the interval $[0, \pi]$.

N_i	PCHIP	MQS	SPS	B-spline	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
					$\mathcal{P}_1$			
17	–	–	–	–	2.49E-3	1	2.49E-3	1
33	–	–	–	–	6.22E-4	1	6.22E-4	1
65	–	–	–	–	1.56E-4	1	1.56E-4	1
129	–	–	–	–	3.89E-5	1	3.89E-5	1
257	–	–	–	–	9.72E-6	1	9.72E-6	1
	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_4$					
17	4.49E-4	4.47E-5	4.84E-4	4.52E-6	6.70E-06	3.94	2.52E-06	4
33	7.83E-5	7.42E-6	1.20E-4	2.07E-7	2.85E-07	3.97	6.94E-08	4
65	1.38E-5	1.31E-6	3.01E-5	1.22E-8	1.24E-08	3.98	1.96E-09	4
129	2.45E-6	2.31E-7	7.52E-6	7.56E-10	5.43E-10	3.99	5.73E-11	4
257	4.34E-7	4.09E-8	1.88E-6	4.72E-11	2.39E-11	4.00	1.73E-12	4
			$\mathcal{P}_8$					
17	–	–	2.00E-3	2.45E-9	6.21E-06	7.69	1.06E-09	8
33	–	–	4.96E-4	3.47E-12	2.76E-07	7.84	1.83E-12	8
65	–	–	1.24E-4	6.11E-15	1.22E-08	7.92	3.44E-15	8
129	–	–	3.10E-5	3.23E-15	5.40E-10	7.96	1.00E-15	8
257	–	–	7.74E-6	2.93E-15	2.39E-11	7.98	9.64E-16	8
			$\mathcal{P}_{16}$					
17	–	–	3.10E-3	5.61E-15	6.21E-06	15.19	3.98E-15	16
33	–	–	7.75E-4	4.35E-13	2.76E-07	15.59	1.95E-15	16
65	–	–	1.94E-4	2.75E-13	1.22E-08	15.80	4.65E-15	16
129	–	–	4.84E-5	9.00E-14	5.40E-10	15.90	2.33E-15	16
257	–	–	1.21E-5	6.83E-14	2.39E-11	15.95	1.10E-15	16

PPI method leads to slightly smaller errors compared to the unmodified B-spline approach. For $\mathcal{P}_8$ and $\mathcal{P}_{16}$ the approximation errors are close to machine precision, which explains the slow rate of convergence observed for B-spline and PPI in Table 5.2.

5.5 Results

In this section, the different interpolation methods are used to approximate functions with steep gradients, C^0-continuity, and discontinuities. These experiments focus on the third criterion, which consists of evaluating the ability of the different methods to represent nonsmooth functions. The data points for the interpolation are sampled from 1D and 2D functions. Two types of meshes are used for the various experiments. The first type of mesh uses uniform elements and uniformly spaced nodes within each element. The second type

Table 5.2: Ratio of L^2-errors from Table 5.1 ($e_{N_i}/e_{N_{i+1}}$). N_i represents the number of input points used to build the approximation. P_j represents the space of polynomials of degree j, with j being the target degree for each interval.

$e_{N_i}/e_{N_{i+1}}$	PCHIP	MQS	SPS	B-spline	DBI	PPI
					$\mathcal{P}_1$	
e_{17}/e_{33}	–	–	–	–	4	4
e_{33}/e_{65}	–	–	–	–	4	4
e_{65}/e_{129}	–	–	–	–	4	4
e_{129}/e_{257}	–	–	–	–	4	4
	$\mathcal{P}_3$	$\mathcal{P}_5$		$\mathcal{P}_4$		
e_{17}/e_{33}	5.73	6.02	4.03	21.84	24	36
e_{33}/e_{65}	5.67	5.67	3.99	16.97	23	35
e_{65}/e_{129}	5.63	5.66	4.00	16.13	23	34
e_{129}/e_{257}	5.64	5.66	4.00	16.02	23	33
				$\mathcal{P}_8$		
e_{17}/e_{33}	–	–	4.03	706.05	22	576
e_{33}/e_{65}	–	–	4.00	567.92	23	533
e_{65}/e_{129}	–	–	4.00	1.89	23	3
e_{129}/e_{257}	–	–	4.01	1.10	23	1
				$\mathcal{P}_{16}$		
e_{17}/e_{33}	–	–	4.01	0.01	22	2
e_{33}/e_{65}	–	–	3.99	1.58	23	0
e_{65}/e_{129}	–	–	4.01	3.06	23	2
e_{129}/e_{257}	–	–	4.00	1.32	23	2

of mesh uses uniform elements and Legendre Gauss-Lobatto (LGL) quadrature nodes [36] within each element. The experimental results compare the DBI and PPI methods against the SPS, PCHIP [33], and MQS [64] methods.

The MQS algorithm is designed for monotonically increasing data. In order to use the MQS approach with the different 1D examples, we divide the data into monotonically increasing and decreasing regions. For the monotonically increasing data, the MQS algorithm is applied directly. For the monotonically decreasing data, we uses the reflection of the data about a vertical axis and applied the MQS algorithm. Because of the data transformation involved, the MQS method is used only for the 1D examples.

The tables show the L^2 error norms and the averaged polynomial degree (“avg. deg.") when using the different methods to approximate the 1D and 2D functions, respectively.

5.5.1 Example I $f_1(x)$

This example is the 1D Runge function [23] defined in Equation 5.1 with $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. Tables 5.3 and 5.4 demonstrate that the PPI method gives smaller

Table 5.3: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the Runge function $f_1(x) = \frac{1}{1+25x^2}$, $x \in [-1,1]$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The sixth and eighth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The input points are uniformly distributed over the interval $[-1,1]$.

N_i	PCHIP	MQS	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
				$\mathcal{P}_1$			
17	–	–	–	2.16E-2	1	2.16E-2	1
33	–	–	–	6.02E-3	1	6.02E-3	1
65	–	–	–	1.52E-3	1	1.52E-3	1
129	–	–	–	3.82E-4	1	3.82E-4	1
257	–	–	–	9.56E-5	1	9.56E-5	1
	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_4$				
17	7.15E-3	5.72E-3	8.34E-3	8.34E-3	4	7.02E-3	4
33	1.91E-3	3.95E-4	5.91E-4	5.91E-4	4	5.91E-4	4
65	3.70E-4	6.44E-5	4.26E-5	4.26E-5	3.98	2.39E-5	4
129	6.79E-5	5.27E-6	2.68E-6	2.68E-6	3.98	8.00E-7	4
257	1.22E-5	6.83E-7	8.63E-8	8.63E-8	4.00	2.55E-8	4
			$\mathcal{P}_8$				
17	–	–	1.21E-2	4.61E-3	7.88	3.11E-3	7.88
33	–	–	2.74E-3	4.43E-4	7.88	1.51E-4	8
65	–	–	6.86E-4	3.67E-5	7.92	1.05E-6	8
129	–	–	1.72E-4	2.56E-6	7.92	3.10E-9	8
257	–	–	4.30E-5	8.24E-8	7.97	6.80E-12	8
			$\mathcal{P}_{16}$				
17	–	–	1.64E-2	4.34E-3	11.31	3.44E-3	11.75
33	–	–	4.25E-3	4.21E-4	15.62	4.85E-5	16
65	–	–	1.07E-3	3.67E-5	15.69	5.92E-8	16
129	–	–	2.69E-4	2.56E-6	15.80	4.21E-12	16
257	–	–	6.71E-5	8.24E-8	15.91	2.18E-16	16

approximation errors when compared to the other approaches. The requirements of data boundedness in the DBI method are restrictive compared to the positivity requirements in PPI. These restrictions lead to lower average polynomial degrees for DBI compared to PPI, as shown in the sixth and eighth columns in Tables 5.3 and 5.4. In the case of the uniform mesh, as the average degree used by DBI increases the L^2-errors remain the same. The approximation error using the DBI method does not improve as the average degree increases because the global error is dominated by the local error of those subintervals with low degree interpolants. The polynomial degree of the interpolants used for these intervals

Table 5.4: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the Runge function $f_1(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1]$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The sixth and eight columns show the average polynomial degree used for the DBI and PPI methods, respectively. The interval [-1,1] is divided into $(N_i - 1)/j$ and $j+1$ LGL quadrature points are used in each element.

N_i	PCHIP	MQS	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
				$\mathcal{P}_1$			
17	–	–	–	2.16E-2	1	2.16E-2	1
33	–	–	–	6.02E-3	1	6.02E-3	1
65	–	–	–	1.52E-3	1	1.52E-3	1
129	–	–	–	3.82E-4	1	3.82E-4	1
257	–	–	–	9.56E-5	1	9.56E-5	1
	$\mathcal{P}_3$	$\mathcal{P}_5$		$\mathcal{P}_4$			
17	1.02E-2	6.29E-3	9.73E-3	8.63E-3	4	8.39E-3	4
33	1.86E-3	9.13E-4	1.63E-3	7.95E-4	4	7.80E-4	4
65	3.68E-4	8.47E-5	2.24E-4	4.76E-5	3.98	4.64E-5	4
129	7.20E-5	6.23E-6	6.03E-5	1.49E-6	3.98	1.27E-6	4
257	1.52E-5	5.72E-7	1.48E-5	4.68E-8	4	3.95E-8	4
				$\mathcal{P}_8$			
17	–	–	8.44E-3	3.49E-3	8.00	4.40E-3	8
33	–	–	2.69E-3	1.76E-4	7.88	1.76E-4	8
65	–	–	7.59E-4	3.25E-6	7.92	3.01E-6	8
129	–	–	2.61E-4	5.64E-8	7.94	8.82E-9	8
257	–	–	6.85E-5	3.51E-9	7.96	3.96E-11	8
				$\mathcal{P}_{16}$			
17	–	–	2.29E-2	9.12E-3	12.19	1.25E-2	12.62
33	–	–	3.26E-3	5.87E-5	15.28	5.86E-5	16
65	–	–	1.11E-3	1.41E-7	15.62	1.17E-7	16
129	–	–	3.12E-4	3.52E-9	15.90	4.44E-11	16
257	–	–	1.08E-4	1.56E-10	15.91	2.88E-15	16

remains the same as the average polynomial degree of the interpolant increases elsewhere. The PPI methods uses higher order interpolants compared to the SPS, DBI, PCHIP, and MQS methods in both the uniform and LGL meshes. The uniform mesh leads to slightly more accurate results than the LGL mesh. These results show that the PPI method is a suitable approach for interpolating data from one mesh to another in cases where the underlying function is similar to the Runge function.

5.5.2 Example II $f_2(x)$

The second example uses the analytic approximation of the Heaviside function defined in Equation 5.2 with $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. As mentioned in Section 3.2, this function, $f_2(x)$, is challenging because of the sharp gradient around $x = 0$. For polynomial degree five or less, the results from Tables 5.5 and 5.6 suggest that the MQS method leads to

Table 5.5: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the function $f_2(x) = \frac{1}{1+e^{-2kx}}$, $k = 100$, and $x \in [-0.2, 0.2]$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The sixth and eighth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The input points are uniformly distributed over the interval $[-1, 1]$.

N_i	PCHIP	MQS	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
					$\mathcal{P}_1$		
17	–	–	–	2.89E-2	1	2.89E-2	1
33	–	–	–	7.69E-3	1	7.69E-3	1
65	–	–	–	1.80E-3	1	1.80E-3	1
129	–	–	–	4.58E-4	1	4.58E-4	1
257	–	–	–	1.15E-4	1	1.15E-4	1
	$\mathcal{P}_3$	$\mathcal{P}_5$			$\mathcal{P}_4$		
17	2.02E-02	1.67E-2	1.82E-2	2.23E-2	2.75	2.23E-2	3.38
33	3.38E-03	4.16E-3	3.72E-3	4.09E-3	3.62	4.10E-3	3.72
65	3.59E-04	2.29E-4	3.40E-4	3.05E-4	3.86	3.05E-4	3.86
129	4.21E-05	7.48E-6	5.36E-5	1.35E-5	3.88	1.35E-5	3.88
257	5.12E-06	2.16E-7	1.27E-5	4.71E-7	3.85	4.71E-7	3.86
					$\mathcal{P}_8$		
17	–	–	3.75E-3	2.08E-2	3.25	2.08E-2	5.50
33	–	–	5.24E-3	3.36E-3	3.88	3.33E-3	5.72
65	–	–	8.71E-4	1.38E-4	7.59	1.38E-4	7.59
129	–	–	2.08E-4	1.22E-6	7.68	1.22E-6	7.73
257	–	–	5.17E-5	4.44E-9	7.61	4.44E-9	7.67
					$\mathcal{P}_{16}$		
17	–	–	5.90E-3	2.00E-2	4.25	2.00E-2	6.62
33	–	–	6.34E-3	2.93E-3	4.38	2.91E-3	9.72
65	–	–	1.30E-3	9.17E-5	14.64	9.17E-5	14.86
129	–	–	3.23E-4	1.70E-7	15.15	1.70E-7	15.41
257	–	–	8.08E-5	2.64E-11	15.05	2.64E-11	15.30

slighter better approximations than DBI and PPI for $f_2(x)$. Overall, the results from Tables 5.5 and 5.6 indicate that the DBI and PPI methods have smaller L^2-errors compared to the other methods. Approximating $f_2(x)$ from data on a uniform mesh leads to slightly better

Table 5.6: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the function $f_2(x) = \frac{1}{1+e^{-2kx}}$, $k = 100$, and $x \in [-0.2, 0.2]$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The sixth and eighth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The interval $[-0.2, 0.2]$ is divided into $(N_i - 1)/j$ elements and $j + 1$ LGL quadrature points are used in each element.

N_i	PCHIP	MQS	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
				$\mathcal{P}_1$			
17	–	–	–	2.89E-2	1	2.89E-2	1
33	–	–	–	7.69E-3	1	7.69E-3	1
65	–	–	–	1.80E-3	1	1.80E-3	1
129	–	–	–	4.58E-4	1	4.58E-4	1
257	–	–	–	1.15E-4	1	1.15E-4	1
	$\mathcal{P}_3$	$\mathcal{P}_5$		$\mathcal{P}_4$			
17	8.60E-3	7.38E-3	7.15E-3	1.26E-2	2.88	1.25E-2	3.44
33	2.50E-3	2.50E-3	8.04E-4	3.11E-3	3.03	2.83E-3	3.44
65	6.36E-4	2.11E-4	4.18E-4	3.28E-4	3.81	3.72E-4	3.84
129	1.02E-4	1.01E-5	9.07E-5	1.55E-5	3.88	1.55E-5	3.88
257	1.83E-5	2.93E-7	1.83E-5	6.29E-7	3.85	6.29E-7	3.86
				$\mathcal{P}_8$			
17	–	–	4.43E-3	4.87E-3	3.50	4.68E-3	5.00
33	–	–	2.51E-3	8.71E-4	4.34	7.84E-4	5.75
65	–	–	1.00E-3	7.57E-5	6.64	1.24E-4	7.28
129	–	–	3.65E-4	2.17E-6	7.65	2.17E-6	7.73
257	–	–	9.11E-5	1.95E-8	7.55	1.95E-8	7.73
				$\mathcal{P}_{16}$			
17	–	–	4.52E-2	3.77E-2	3.81	3.73E-2	7.25
33	–	–	2.03E-3	2.53E-4	5.56	5.23E-4	9.84
65	–	–	9.55E-4	1.37E-5	10.53	6.95E-5	12.56
129	–	–	4.16E-4	2.19E-7	15.16	2.19E-7	15.30
257	–	–	1.51E-4	1.56E-10	14.96	1.56E-10	15.30

results compared to LGL mesh data. For smooth data with a large gradient, these results indicate that both the DBI and PPI approaches are suitable for interpolating from one mesh to another.

5.5.3 Example III $f_3(x)$

The third example uses the modified function introduced in Equation 6.23 with $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. The function $f_3(x)$ is particularly challenging because of the discontinuity at $x = -0.5$. The results from Tables 5.7 and 5.8 show that the L^2-errors from the four interpolation methods have the same order of accuracy. The results from Tables 5.7 and 5.8

Table 5.7: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the function $f_3(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The fifth and seventh columns show the average polynomial degree used for the DBI and PPI methods, respectively. The input points are uniformly distributed over the interval $[-1, 1]$.

N_i	PCHIP	MQS	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
				$\mathcal{P}_1$			
17	–	–	–	1.82E-1	1	1.82E-1	1
33	–	–	–	1.39E-1	1	1.39E-1	1
65	–	–	–	1.01E-1	1	1.01E-1	1
129	–	–	–	7.16E-2	1	7.16E-2	1
257	–	–	–	5.05E-2	1	5.05E-2	1
	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_4$				
17	1.77E-1	1.59E-1	2.32E-1	1.82E-1	3.62	1.71E-1	3.81
33	1.39E-1	1.11E-1	1.56E-1	1.39E-1	3.97	1.29E-1	3.97
65	1.03E-1	7.90E-2	1.09E-1	9.38E-2	3.98	9.38E-2	3.98
129	7.42E-2	5.63E-2	7.69E-2	6.69E-2	3.99	6.70E-2	3.99
257	5.28E-2	4.04E-2	5.45E-2	4.73E-2	4	4.74E-2	4
			$\mathcal{P}_8$				
17	–	–	2.25E-1	1.83E-1	6.62	1.70E-1	7.06
33	–	–	1.53E-1	1.36E-1	7.81	1.31E-1	7.81
65	–	–	1.07E-1	9.62E-2	7.92	9.65E-2	7.92
129	–	–	7.58E-2	6.90E-2	7.96	6.93E-2	7.96
257	–	–	5.37E-2	4.90E-2	7.98	4.92E-2	7.98
			$\mathcal{P}_{16}$				
17	–	–	2.25E-1	1.82E-1	12.06	1.66E-1	13.06
33	–	–	1.50E-1	1.37E-1	14.09	1.33E-1	14.19
65	–	–	1.05E-1	9.75E-2	15.78	9.81E-2	15.78
129	–	–	7.45E-2	7.02E-2	15.90	7.07E-2	15.90
257	–	–	5.29E-2	4.99E-2	15.95	5.03E-2	15.95

show that the L^2-errors from the four interpolation methods have the same order. because around the discontinuity, the methods are as accurate as the other ones, and in smooth regions the method gives better approximation results than the other approaches. The DBI and PPI methods give slightly better approximation results compared to the other methods. The average polynomial degrees for the DBI and PPI approaches show that high-order polynomials are used. The high-order polynomial suggest that in the smooth regions away from the discontinuity, the DBI and PPI approaches lead to high-order accuracy. However, at the discontinuity, the DBI and PPI and other methods struggle to represent the underlying function. This example shows that both the DBI and PPI methods are

Table 5.8: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the function $f_3(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The sixth and eighth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The interval $[-1, 1]$ is divided into $(N_i - 1)/j$ elements and $j + 1$ LGL quadrature points are used in each element.

N_i	PCHIP	MQS	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
				$\mathcal{P}_1$			
17	–	–	–	1.82E-1	1	1.82E-1	1
33	–	–	–	1.39E-1	1	1.39E-1	1
65	–	–	–	1.01E-1	1	1.01E-1	1
129	–	–	–	7.16E-2	1	7.16E-2	1
257	–	–	–	5.05E-2	1	5.05E-2	1
	$\mathcal{P}_3$	$\mathcal{P}_5$		$\mathcal{P}_4$			
17	1.64E-1	1.39E-1	1.87E-1	1.61E-1	3.81	1.58E-1	3.81
33	1.20E-1	9.79E-2	1.29E-1	1.18E-1	3.97	1.18E-1	3.97
65	8.70E-2	6.94E-2	9.04E-2	8.32E-2	3.98	8.53E-2	3.98
129	6.21E-2	4.96E-2	6.39E-2	5.93E-2	3.99	6.08E-2	3.99
257	4.39E-2	3.58E-2	4.54E-2	4.19E-2	4.00	4.29E-2	4
				$\mathcal{P}_8$			
17	–	–	2.84E-1	1.85E-1	7.38	1.81E-01	7.50
33	–	–	9.61E-2	9.38E-2	7.62	1.27E-01	7.66
65	–	–	6.79E-2	6.75E-2	7.92	9.33E-02	7.92
129	–	–	4.82E-2	4.79E-2	7.96	6.72E-02	7.96
257	–	–	3.44E-2	3.37E-2	7.98	4.77E-02	7.98
				$\mathcal{P}_{16}$			
17	–	–	1.11E-1	1.08E-1	11.62	1.51E-1	12.12
33	–	–	1.86E-1	1.66E-1	14.31	1.55E-1	14.88
65	–	–	4.91E-2	5.06E-2	15.56	8.28E-2	15.58
129	–	–	3.51E-2	3.56E-2	15.90	5.92E-2	15.90
257	–	–	2.53E-2	2.48E-2	15.94	4.19E-2	15.94

appropriate approaches for interpolating from one mesh to another,

5.5.4 Example IV $f_4(x)$

The fourth example uses the function $f_4(x)$ defined in Equation 5.4 with $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. $f_4(x)$ depends on the size h of each element, and as the number of element changes, so does the element size h and the function $f_4(x)$.

At the element boundaries $f_4(x)$ is only C^0-continuous with large gradients of opposite signs. The results from Tables 5.9 and 5.10 show that all the methods struggle to approximate the underlying function. With the exception of using a uniform mesh with PCHIP and

Table 5.9: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the Runge function $f_4(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The value ne represents the number of elements. The seventh and ninth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The input points are uniformly distributed over the interval $[0, 1]$.

N_i	PCHIP	MQS	$\mathcal{P}_j$	SPS	DBI		PPI	
	L^2-error	L^2-error		L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
	$ne = 4$							
17	3.74E-01	3.95E-1	$\mathcal{P}_4$	3.72E-1	3.49E-1	2.62	3.49E-1	2.88
33	2.47E-01	2.59E-1	$\mathcal{P}_8$	2.46E-1	2.19E-1	5.19	2.19E-1	6.19
65	1.55E-01	1.63E-1	$\mathcal{P}_{16}$	1.54E-1	1.32E-1	13.34	1.32E-1	15.34
	$ne = 8$							
33	3.84E-01	3.94E-1	$\mathcal{P}_4$	3.83E-1	4.04E-1	2.69	4.04E-1	2.94
65	2.54E-01	2.59E-1	$\mathcal{P}_8$	2.52E-1	2.60E-1	5.34	2.74E-1	6.34
129	1.61E-01	8.23E-2	$\mathcal{P}_{16}$	1.57E-1	1.67E-1	13.55	1.78E-1	15.30
	$ne = 16$							
65	3.90E-01	3.93E-1	$\mathcal{P}_4$	3.89E-1	3.61E-1	2.72	3.61E-1	2.97
129	2.58E-01	2.58E-1	$\mathcal{P}_8$	2.55E-1	2.26E-1	5.42	2.26E-1	6.42
257	1.63E-01	8.06E-2	$\mathcal{P}_{16}$	1.58E-1	1.36E-1	13.65	1.36E-1	15.65

Table 5.10: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the Runge function $f_4(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The value ne represents the number of elements. The seventh and ninth columns show the average polynomial degree used for the DBI and PPI methods respectively. The interval $[0, 1]$ is divided into $(N_i - 1)/j$ elements and $j + 1$ LGL quadrature points are used in each element.

N_i	PCHIP	MQS	$\mathcal{P}_j$	SPS	DBI		PPI	
	L^2-error	L^2-error		L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
	$ne = 4$							
17	3.02E-1	3.18E-1	$\mathcal{P}_4$	3.02E-1	2.32E-1	2.75	2.32E-1	3.00
33	1.33E-1	1.39E-1	$\mathcal{P}_8$	1.33E-1	9.60E-2	5.25	9.60E-2	6.25
65	3.80E-2	3.92E-2	$\mathcal{P}_{16}$	3.77E-2	2.11E-2	11.88	2.11E-2	13.31
	$ne = 8$							
33	3.10E-1	3.17E-1	$\mathcal{P}_4$	3.10E-1	3.42E-1	2.75	3.42E-1	3.00
65	1.37E-1	1.39E-1	$\mathcal{P}_8$	1.36E-1	1.59E-1	5.25	1.65E-1	6.12
129	3.98E-2	3.72E-2	$\mathcal{P}_{16}$	3.87E-2	5.33E-2	11.88	5.60E-2	13.31
	$ne = 16$							
65	3.14E-1	3.16E-1	$\mathcal{P}_4$	3.14E-1	2.32E-1	2.75	2.32E-1	3.00
129	1.39E-1	1.38E-1	$\mathcal{P}_8$	1.37E-1	9.60E-2	5.25	9.60E-2	6.25
257	4.07E-2	3.37E-2	$\mathcal{P}_{16}$	3.90E-2	2.11E-2	11.88	2.11E-2	13.31

SPS, the remaining results from Tables 5.9 and 5.10 show that all the methods have the same order of accuracy for both uniform and LGL meshes. The PPI and DBI methods give slightly smaller L^2-errors compared to the other approaches.

5.5.5 Example V $f_5(x)$

The fifth experiment uses the function $f_5(x)$ defined in Equation 5.5 with $\epsilon_0 = 0.01$, $\epsilon_1 = 1$, and $st = 1$. $f_5(x)$ depends on the size h of each element, and as the number of elements changes, so does the element size h and $f_5(x)$. Similarly to $f_4(x)$, $f_5(x)$ is only C^0-continuous at the element boundaries. However, the gradients remain positive over the entire interval. This example is constructed to reflect the C^0-continuity observed in the spectral element method used in NEPTUNE. Tables 5.11 and 5.12 show that the approximation errors from PCHIP, MQS, and SPS methods improve slowly compared to the DBI and PPI methods, as we increase the polynomial degree and the number of points. The PCHIP, SPS, and MQS methods use approximations of the first derivatives and enforce C^1-continuity at the element boundaries. Overall, the results from Tables 5.11 and 5.12 show that the DBI and PPI methods has smaller L^2-errors compared to the remaining methods.

Table 5.11: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the function $f_5(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The value ne is the number of elements used. The seventh and ninth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The input points are uniformly distributed over the interval $[-2, 0]$.

N_i	PCHIP	MQS	$\mathcal{P}_j$	SPS	DBI		PPI	
	L^2-error	L^2-error		L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
	$ne = 4$							
17	1.68E-2	4.50E-2	$\mathcal{P}_4$	1.23E-2	2.36E-2	3.56	2.36E-2	3.56
33	9.95E-3	6.04E-3	$\mathcal{P}_8$	1.36E-2	4.20E-4	7.59	4.20E-4	7.62
65	1.67E-3	3.82E-4	$\mathcal{P}_{16}$	5.77E-3	3.65E-5	14.98	3.65E-5	14.98
	$ne = 8$							
33	1.99E-2	1.20E-2	$\mathcal{P}_4$	1.29E-2	1.65E-2	3.91	1.65E-2	3.91
65	3.35E-3	7.63E-4	$\mathcal{P}_8$	7.42E-3	2.17E-4	7.67	2.17E-4	7.67
129	3.70E-4	4.44E-5	$\mathcal{P}_{16}$	2.89E-3	5.01E-5	14.84	5.01E-5	14.88
	$ne = 16$							
33	6.73E-3	2.46E-3	$\mathcal{P}_4$	4.57E-3	1.18E-3	3.86	1.18E-3	3.86
65	8.22E-4	4.53E-4	$\mathcal{P}_8$	3.61E-3	5.27E-5	7.51	5.27E-5	7.51
256	1.53e-4	1.59E-4	$\mathcal{P}_{16}$	1.42E-3	4.27E-11	14.65	4.27E-11	14.70

Table 5.12: L^2-errors when using the PCHIP, MQS, SPS, DBI, and PPI methods to approximate the function $f_5(x)$. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree for each interval. The value of ne is the number of elements used. The seventh and ninth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The interval $[-2, 0]$ is divided into $(N_i - 1)/j$ elements and $j + 1$ LGL quadrature points are used in each element.

N_i	PCHIP	MQS	$\mathcal{P}_j$	SPS	DBI		PPI	
	L^2-error	L^2-error		L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
	$ne = 4$							
17	4.64E-2	3.10E-2	$\mathcal{P}_4$	1.23E-2	5.09E-2	3.06	5.09E-2	3.06
33	7.43E-3	4.29E-3	$\mathcal{P}_8$	1.36E-2	1.47E-3	6.34	1.47E-3	6.44
65	9.48E-4	1.58E-4	$\mathcal{P}_{16}$	5.77E-3	3.15E-6	14.83	3.15E-6	14.84
	$ne = 8$							
33	2.57E-2	9.45E-3	$\mathcal{P}_4$	1.29E-2	1.52E-2	3.84	1.52E-2	3.84
65	3.18E-3	9.15E-4	$\mathcal{P}_8$	7.42E-3	2.66E-4	7.66	2.66E-4	7.67
129	4.08E-4	2.71E-5	$\mathcal{P}_{16}$	2.89E-3	3.75E-4	14.88	4.53E-4	14.88
	$ne = 16$							
33	8.03E-3	3.77E-3	$\mathcal{P}_4$	4.57E-3	1.91E-3	3.81	1.91E-3	3.81
65	9.93E-4	2.22E-4	$\mathcal{P}_8$	3.61E-3	2.23E-6	7.51	2.23E-6	7.51
256	1.23E-4	3.30E-5	$\mathcal{P}_{16}$	1.42E-3	2.63E-12	14.37	2.63E-12	14.50

5.5.6 Example VII $f_7(x)$

This example uses an extended version of the 1D Runge function defined in Equation 5.1 from Section 5.3.1 to a 2D function.

$$f_7(x, y) = \frac{1}{1 + 25(x^2 + y^2)}, \quad x, y \in [-1, 1] \tag{5.7}$$

For the DBI and PPI algorithm, $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. The results from Tables 5.13 and 5.14 show that the DBI and PPI methods give smaller approximation errors compared to the PCHIP and SPS methods. In this case, the DBI and PPI methods use higher order polynomial interpolants for each interval. These higher order interpolants help improve the approximation compared to the PCHIP and SPS.

5.5.7 Example VIII $f_8(x)$

This example uses a 2D function that is used to study positive and monotonic splines [11,58,83]. The function is defined as follows:

Table 5.13: $L^2 - errors$ when approximating $f_7(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The mesh points are uniformly distributed on each dimension.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	1.60E-2	1	1.60E-2	1
33	–	–	4.42E-3	1	4.42E-3	1
65	–	–	1.12E-3	1	1.12E-3	1
129	–	–	2.82E-4	1	2.82E-4	1
257	–	–	7.06E-5	1	7.06E-5	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	5.01E-3	3.61E-3	5.16E-3	3.97	4.28E-3	4
33	1.23E-3	5.44E-4	3.51E-4	3.98	3.31E-4	4
65	2.33E-4	1.23E-4	2.55E-5	3.98	1.31E-5	4
129	4.27E-5	3.07E-5	1.20E-6	3.99	4.36E-7	4
257	7.72E-6	3.34E-6	4.96E-8	4	1.39E-8	4
		$\mathcal{P}_8$				
17	–	8.55E-3	3.19E-3	7.75	1.84E-3	7.99
33	–	1.99E-3	2.78E-4	7.82	7.86E-5	8
65	–	4.97E-4	2.31E-5	7.90	5.14E-7	8
129	–	1.25E-4	1.13E-6	7.95	1.49E-9	8
257	–	3.16E-5	4.78E-8	7.98	3.25E-12	8
		$\mathcal{P}_{16}$				
17	–	1.19E-2	3.49E-3	13.15	2.83E-3	14.26
33	–	3.10E-3	2.74E-4	15.43	2.68E-5	16
65	–	7.82E-4	2.30E-5	15.69	2.63E-8	16
129	–	1.96E-4	1.13E-6	15.84	1.77E-12	16
257	–	4.93E-5	4.76E-8	15.92	1.89E-15	16

$$f_8(x,y) = \begin{cases} 2(y-x) & \text{if } 0 \le y-x \le 0.5 \\ 1 & \text{if } y-x \ge 0.5 \\ cos\left(4\pi\sqrt{(x-1.5)^2+(y-0.5)^2}\right) & \text{if } (x-1.5)^2+(y-0.5)^2 \le \frac{1}{16} \\ 0 & otherwise \end{cases} \tag{5.8}$$

For the DBI and PPI algorithm, $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. As in Example V, the function $f_8(x)$ is C^0-continuous and the underlying mesh used for the approximations does not capture the sharp corners. The L^2-errors from the DBI and PPI methods are dominated by the local errors of the intervals with C^0-continuity and low-degree polynomial interpolants. Tables 5.15 and 5.16 show that the L^2-errors from the three methods have the same order,

Table 5.14: $L^2 - errors$ when approximating $f_7(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. For each dimension, the interval $[-1,1]$ is divided into $(N_i-1)/j$ elements and $j+1$ LGL quadrature points are used in each element.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	1.60E-2	1	1.60E-2	1
33	–	–	4.42E-3	1	4.42E-3	1
65	–	–	1.12E-3	1	1.12E-3	1
129	–	–	2.82E-4	1	2.82E-4	1
257	–	–	7.11E-5	1	7.11E-5	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	6.31E-03	5.41E-3	5.54E-3	3.97	5.33E-3	4
33	4.15E-04	1.05E-3	4.59E-4	3.98	4.53E-4	4
65	1.07E-04	1.61E-4	2.67E-5	3.98	2.54E-5	4
129	2.46E-05	4.31E-5	7.97E-7	3.99	6.86E-7	4
257	5.20E-06	1.12E-5	2.79E-8	4.00	2.15E-8	4
		$\mathcal{P}_8$				
17	–	5.89E-3	3.02E-3	7.75	2.81E-3	7.99
33	–	1.72E-3	9.41E-5	7.82	9.34E-5	8
65	–	5.44E-4	1.78E-6	7.90	1.51E-6	8
129	–	1.90E-4	4.31E-8	7.95	4.53E-9	8
257	–	4.91E-5	1.73E-9	7.98	1.87E-11	8
		$\mathcal{P}_{16}$				
17	–	1.88E-2	6.31E-3	13.15	8.93E-3	14.26
33	–	2.11E-3	2.93E-5	15.43	2.91E-5	16
65	–	6.96E-4	8.12E-7	15.69	5.41E-8	16
129	–	2.25E-4	2.27E-8	15.84	1.97E-11	16
257	–	7.93E-5	9.21E-10	15.92	7.22E-15	16

with DBI and PPI having slightly smaller errors than the other approaches. In the cases where the underlying function is C^0, the results from DBI and PPI are comparable to the other approaches. Furthermore, the results from DBI and PPI can be improved by using a mesh that captures C^0-continuity, as is the case with the spectral element methods in NEPTUNE.

5.5.8 Example IX $f_9(x)$

This example is used herein to study shape-preserving (monotonicity and convexity) splines [16].

Table 5.15: $L^2 - errors$ when approximating $f_8(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The points are uniformly distributed in each dimension.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	2.70E-2	1	2.70E-2	1
33	–	–	9.51E-3	1	9.51E-3	1
65	–	–	3.40E-3	1	3.40E-3	1
129	–	–	1.20E-3	1	1.20E-3	1
257	–	–	4.30E-4	1	4.30E-4	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	1.91E-2	1.87E-2	1.77E-2	2.04	1.73E-2	2.06
33	6.92E-3	6.11E-3	6.22E-3	1.93	6.21E-3	1.95
65	2.47E-3	2.69E-3	2.24E-3	1.89	2.24E-3	1.90
129	8.99E-4	7.71E-4	8.17E-4	1.88	8.16E-4	1.88
257	3.23E-4	2.77E-4	2.95E-4	1.87	2.94E-4	1.87
		$\mathcal{P}_8$				
17	–	1.91E-2	1.73E-2	3.19	1.69E-2	3.31
33	–	6.46E-3	6.20E-3	3.09	6.19E-3	3.16
65	–	2.24E-3	2.21E-3	3.04	2.20E-3	3.09
129	–	8.12E-4	7.98E-4	3.03	7.97E-4	3.04
257	–	2.92E-4	2.87E-4	3.01	2.87E-4	3.02
		$\mathcal{P}_{16}$				
17	–	2.19E-2	2.02E-2	4.58	2.34E-2	4.94
33	–	7.57E-3	6.14E-3	5.13	6.17E-3	5.35
65	–	2.68E-3	2.25E-3	5.25	2.26E-3	5.38
129	–	9.63E-4	8.07E-4	5.29	8.08E-4	5.35
257	–	3.45E-4	2.89E-4	5.29	2.89E-4	5.32

$$f_9(x,y) = max\Big(0, sin(\pi x)sin(\pi y)\Big) \quad x,y \in [-1,1] \tag{5.9}$$

For the DBI and PPI algorithm, $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. The function $f_9(x,y)$ is a C^0-continuous function. Tables 5.17 and 5.18 show L^2-errors when approximating $f_9(x,y)$ with the PCHIP, SPS, DBI, and PPI methods. The underlying mesh is such that the C^0-continuities are at the elements boundaries except for $\mathcal{P}_{16}$ and $N = 17$. The PCHIP and SPS methods struggle to capture the C^0-continuities because both methods enforce C^1-continuity. The L^2-error from DBI is dominated by the local error from the intervals with low-degree interpolants and so as the average polynomial degree increases the L^2-errors do not improve. The L^2-error for $\mathcal{P}_{16}$ and $N = 17$ is larger compared to the other cases

Table 5.16: $L^2 - errors$ when approximating $f_8(x, y)$ with $N_i \times N_i$ points. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. For each dimension, the interval $[-1, 1]$ is divided into $(N_i - 1)/j$ elements and $j + 1$ LGL quadrature points are used in each element.

N_i	PCHIP		DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	2.70E-2	1	2.70E-2	1
33	–	–	9.51E-3	1	9.51E-3	1
65	–	–	3.40E-3	1	3.40E-3	1
129	–	–	1.20E-3	1	1.20E-3	1
257	–	–	4.30E-4	1	4.30E-4	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	1.91E-2	2.55E-2	2.18E-2	2.04	2.17E-2	2.06
33	6.92E-3	5.76E-3	7.22E-3	1.93	7.18E-3	1.95
65	2.47E-3	2.11E-3	2.74E-3	1.89	2.72E-3	1.90
129	8.99E-4	8.08E-4	9.93E-4	1.88	9.87E-4	1.88
257	3.23E-4	2.92E-4	3.61E-4	1.87	3.59E-4	1.87
		$\mathcal{P}_8$				
17	–	4.06E-2	3.42E-2	3.19	3.19E-2	3.31
33	–	9.69E-3	8.68E-3	3.09	8.67E-3	3.16
65	–	2.46E-3	2.76E-3	3.04	2.82E-3	3.09
129	–	9.83E-4	1.09E-3	3.03	1.12E-3	3.04
257	–	3.63E-4	3.85E-4	3.01	3.99E-4	3.02
		$\mathcal{P}_{16}$				
17	–	4.36E-2	3.35E-2	4.58	2.81E-2	4.94
33	–	1.53E-2	1.06E-2	5.13	1.06E-2	5.35
65	–	4.31E-3	3.42E-3	5.25	3.46E-3	5.38
129	–	1.13E-3	9.84E-4	5.29	1.29E-3	5.35
257	–	4.38E-4	3.95E-4	5.29	5.07E-4	5.32

when the PPI method is used. For $\mathcal{P}_{16}$ and $N = 17$, there is no mesh point at the points of C^0-continuity, and so the L^2-error is dominated by the local error from those intervals where low degree interpolants are used. Overall, the results from Tables 5.17 and 5.18 demonstrate that the DBI and PPI methods lead to smaller approximation errors than the PCHIP and SPS methods.

5.5.9 Example X $f_{10}(x)$

This example uses a 2D extension of the 1D approximation of the Heaviside function $f_2(x)$ defined in Equation 5.2, which is defined as follows:

Table 5.17: $L^2-errors$ when approximating $f_9(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The points are uniformly distributed on each dimension.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	5.80E-2	1	5.80E-2	1
33	–	–	1.88E-2	1	1.88E-2	1
65	–	–	6.27E-3	1	6.27E-3	1
129	–	–	2.17E-3	1	2.17E-3	1
257	–	–	7.87E-4	1	7.87E-4	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	1.91E-2	1.80E-2	1.20E-2	2.56	1.20E-2	2.66
33	6.77E-3	6.21E-3	4.25E-3	2.53	4.25E-3	2.58
65	2.39E-3	2.18E-3	1.50E-3	2.52	1.50E-3	2.54
129	8.47E-4	7.70E-4	5.30E-4	2.51	5.30E-4	2.52
257	3.01E-4	2.74E-4	1.88E-4	2.50	1.88E-4	2.51
		$\mathcal{P}_8$				
17	–	1.47E-2	1.27E-2	4.22	1.27E-2	4.72
33	–	4.57E-3	4.48E-3	4.37	4.48E-3	4.61
65	–	1.51E-4	1.58E-3	4.44	1.58E-3	4.56
129	–	5.16E-4	5.60E-4	4.47	5.60E-4	4.53
257	–	1.84E-4	1.98E-4	4.48	1.98E-4	4.51
		$\mathcal{P}_{16}$				
17	–	1.50E-2	2.22E-2	5.92	2.74E-2	7.11
33	–	4.03E-3	4.79E-3	8.05	4.79E-3	8.67
65	–	1.15E-3	1.69E-3	8.28	1.69E-3	8.58
129	–	3.50E-4	5.98E-4	8.39	5.98E-4	8.54
257	–	1.17E-4	2.12E-4	8.44	2.12E-4	8.52

$$f_{10}(x,y) = \frac{1}{1+e^{-\sqrt{2}k(x+y)}}, \quad x,y \in [-0.2, 0.2] \tag{5.10}$$

For the DBI and PPI algorithm, $\epsilon_0 = 0.01$, $\epsilon_1 = 1$ and $st = 2$. The function $f_{10}(x,y)$ is challenging because of the large gradient at $y = -x$. Tables 5.19 and 5.20 show L^2-errors when approximating $f_{10}(x)$ using PCHIP, SPS, DBI, and PPI. As the average polynomial degree increases, the accuracy of the DBI and PPI methods improves. In this case, the L^2-error is dominated by the local error of the region with the steep gradient. The errors for the DBI and PPI methods are similar because the stencils used for both methods are the same in the region with the large gradient. Overall, the results from Tables 5.19 and 5.20 show that the DBI and PPI methods lead to smaller L^2-errors compared to the other

Table 5.18: $L^2 - errors$ when approximating $f_9(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. For each dimension, the interval $[-1,1]$ is divided into $(N_i-1)/j$ elements and $j+1$ LGL quadrature points are used in each element.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	5.80E-2	1	5.80E-2	1
33	–	–	1.88E-2	1	1.88E-2	1
65	–	–	6.27E-3	1	6.27E-3	1
129	–	–	2.17E-3	1	2.17E-3	1
257	–	–	7.87E-4	1	7.87E-4	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	3.56E-02	1.05E-2	1.85E-2	2.52	1.46E-4	3.91
33	1.79E-03	3.61E-3	4.74E-3	2.61	4.77E-6	3.95
65	2.53E-04	1.26E-3	1.19E-3	2.62	1.43E-7	3.98
129	9.94E-05	4.44E-4	2.98E-4	2.62	9.16E-9	3.99
257	3.26E-05	1.61E-4	7.45E-5	2.62	5.52E-9	3.99
		$\mathcal{P}_8$				
17	–	1.61E-2	1.85E-2	4.04	6.01E-8	7.36
33	–	3.34E-3	4.74E-3	4.57	1.37E-8	7.89
65	–	8.74E-4	1.19E-3	4.57	9.43E-9	7.95
129	–	2.40E-4	2.98E-4	4.56	6.15E-9	7.97
257	–	7.32E-5	7.45E-5	4.56	3.57E-9	7.99
		$\mathcal{P}_{16}$				
17	–	2.72E-2	1.85E-2	4.29	2.35E-3	8.75
33	–	6.65E-3	4.74E-3	7.93	9.88E-9	15.31
65	–	1.33E-3	1.19E-3	8.57	6.44E-9	15.88
129	–	3.35E-4	2.98E-4	8.54	3.78E-9	15.94
257	–	8.41E-5	7.45E-5	8.55	1.75E-9	15.97

methods.

5.6 Discussion and Conclusion

In this chapter, a representative sample of existing methods is compared against our new approaches on a number of different test functions, including smooth, C^0, discontinuous, and steep functions. The comparison undertaken here focuses on how accurately the different methods are able to represent this underlying set of test functions. Overall, the DBI and PPI methods perform well and are suited to the C^0 continuity of the spectral element methods in NEPTUNE. The experiments show that the DBI and PPI methods are suitable

Table 5.19: $L^2 - errors$ when approximating $f_{10}(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods, respectively. The points are uniformly distributed on each dimension.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	1.50E-2	1	1.50E-2	1
33	–	–	4.57E-3	1	4.57E-3	1
65	–	–	1.26E-3	1	1.26E-3	1
129	–	–	3.23E-4	1	3.23E-4	1
257	–	–	8.15E-5	1	8.15E-5	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	8.07E-3	1.99E-3	9.45E-3	2.75	9.44E-3	3.23
33	1.26E-3	2.43E-4	1.33E-3	3.53	1.29E-3	3.65
65	1.44E-4	4.92E-5	9.29E-5	3.82	9.29E-5	3.82
129	1.63E-5	1.20E-5	3.67E-6	3.81	3.67E-6	3.81
257	1.94E-6	3.05E-6	1.21E-7	3.79	1.21E-7	3.79
		$\mathcal{P}_8$				
17	–	1.80E-1	8.05E-3	3.15	8.67E-3	5
33	–	1.22E-1	1.03E-3	5.78	9.05E-4	6.55
65	–	8.48E-2	4.83E-5	7.54	4.99E-5	7.57
129	–	1.04E-1	2.57E-7	7.53	2.57E-7	7.55
257	–	8.02E-2	5.27E-10	7.49	5.27E-10	7.52
		$\mathcal{P}_{16}$				
17	–	4.72E-3	7.39E-3	3.57	1.86E-2	7.45
33	–	1.22E-3	1.02E-3	6.71	2.33E-3	10.23
65	–	3.11E-4	2.12E-4	14.76	2.41E-4	14.94
129	–	7.80E-5	1.03E-6	14.93	1.03E-6	15.05
257	–	1.95E-5	4.41E-11	14.83	4.41E-11	14.97

approaches for interpolating smooth functions and C^0 continuous functions while enforcing positivity. In detail the summary is that:

- The results in Section 3 Examples I, II, and III show that the improved DBI and new PPI approaches preserve positivity exactly as the proofs in [82] indicate;
- The results in Section 4 and Sections 5.1, 5.2, and 5.3 show that the DBI and PPI approaches give much higher levels of accuracy than the DBI method by allowing the solution to be outside the local bounds while remaining positive. The PPI method also appears to give better results than the SPS method in line with the studies

Table 5.20: $L^2 - errors$ when approximating $f_{10}(x,y)$ with $N_i \times N_i$ points. The parameters ϵ_0 and ϵ_1 are set to 0.01 and 1.0, respectively. N_i represents the number of input points used in each dimension to build the approximation. P_j represents the use of polynomials of degree j, with j being the target degree. The fourth and sixth columns show the average polynomial degree used for the DBI and PPI methods respectively. For each dimension, the interval $[-0.2, 0.2]$ is divided into $(N_i - 1)/j$ elements and $j+1$ LGL quadrature points are used in each element.

N_i	PCHIP	SPS	DBI		PPI	
	L^2-error	L^2-error	L^2-error	avg. deg.	L^2-error	avg. deg.
			$\mathcal{P}_1$			
17	–	–	1.50E-2	1	1.50E-2	1
33	–	–	4.57E-3	1	4.57E-3	1
65	–	–	1.26E-3	1	1.26E-3	1
129	–	–	3.23E-4	1	3.23E-4	1
257	–	–	8.15E-5	1	8.15E-5	1
	$\mathcal{P}_3$	$\mathcal{P}_4$				
17	1.32E-2	2.79E-3	1.12E-2	2.75	1.11E-2	3.23
33	2.75E-3	3.49E-4	1.73E-3	3.53	1.68E-3	3.65
65	3.57E-4	6.87E-5	1.28E-4	3.82	1.28E-4	3.82
129	4.09E-5	1.64E-5	5.47E-6	3.81	5.47E-6	3.81
257	5.04E-6	4.12E-6	1.77E-7	3.79	1.77E-7	3.79
		$\mathcal{P}_8$				
17	–	5.82E-3	1.22E-2	3.15	1.20E-2	5.00
33	–	1.26E-3	1.82E-3	5.78	1.67E-3	6.55
65	–	3.00E-4	4.98E-5	7.54	4.98E-5	7.57
129	–	7.58E-5	4.03E-7	7.53	4.03E-7	7.55
257	–	1.90E-5	1.21E-9	7.49	1.21E-9	7.52
		$\mathcal{P}_{16}$				
17	–	8.05E-3	1.34E-2	3.57	1.31E-2	7.45
33	–	2.06E-3	2.04E-3	6.71	1.92E-3	10.23
65	–	5.14E-4	3.81E-5	14.76	3.84E-5	14.94
129	–	1.26E-4	6.20E-8	14.93	6.20E-8	15.05
257	–	3.18E-5	6.20E-12	14.83	6.20E-12	14.97

in [13] and [15], which demonstrate that the SPS method does not achieve high-order accuracy; and

- In the cases when steep gradients or discontinuities force the use of low-order approximations, the DBI and PPI methods compete against the well-known cubic spline method PCHIP and the higher order MQS and the SPS spline methods.

Overall, it would seem that when it is possible to use higher order polynomial approximations the PPI method appears to give levels of accuracy that compete with standard unmodified high-order spline methods while at the same time preserving positivity.

CHAPTER 6

HPPIS: A HIG-HORDER POSITIVITY-PRESERVING MAPPING SOFTWARE FOR STRUCTURED MESHES

6.1 Introduction

This chapter introduces open-source software for high-order data-bounded and positivity-preserving interpolation (HiPPIS) that addresses the limitations of both the spline and polynomial rescaling methods. HiPPIS uses a given set of data points to construct high-degree polynomial interpolants that are positive over the domains in which they are defined. The high-order positive interpolants obtained from HPPIS are suitable for approximating and mapping physical quantities such as mass, density, and concentration between meshes while preserving positivity. HiPPIS provides Fortran and Matlab implementations of the data-bounded and positivity-preserving interpolation methods. Both the Fortran and Matlab versions are self-contained and are easy to integrate into other application software requiring positivity. In addition to the software, this chapter provides an analysis of the mapping error in the context of PDEs, uses several 1D and 2D numerical examples to demonstrate the benefits and limitations of HPPIS, and introduces different strategies to improve locality, vectorization, and overall, the performance of HiPPIS.

Mapping data values from one grid to another is a fundamental part of many computational problems. Preserving certain properties such as positivity when interpolating solution values between meshes is important. In many applications [1, 62, 87, 98, 100, 110], failure to preserve the positivity of quantities such as mass, density, and concentration results in negative values that are unphysical. These negative values may propagate to other calculations and corrupt other quantities. Many polynomial-based methods have been developed to address these limitations.

Positivity-preserving methods based on linear polynomial rescaling are introduced in [45,60,62,110,112]. These polynomial rescaling methods are often used in the context of hyperbolic PDEs, in numerical weather prediction (NWP) [60], combustion simulation [45,62], and other applications. These methods introduce rescaling parameters obtained from quadrature weights that are used to linearly rescale the polynomial to ensure positivity at the quadrature nodes and conserve mass. These approaches ensure positivity only at the set of mesh points used for the simulation but do not address the case of mapping data values between different meshes, which is the focus of HiPPIS.

Other approaches for preserving positivity that are based on splines can be found in computer-aided design (CAD), graphics, and visualization [33,47,53,88–90]. Several positivity- and monotonicity-preserving cubic splines have been developed. A widely used example of such an approach is the piecewise cubic Hermite interpolation (PCHIP) [33], which is available as open-source code in [72]. In addition, quartic and quintic spline-based approaches have been introduced in [41,46,48,64]. These methods impose some restrictions on the first and second derivatives to ensure monotonicity, positivity, and continuity. For instance, the monotonic quintic spline (MQS) in [64] uses the sufficient condition from [90] to check for monotonicity and the approaches in [104] to adjust the values of the first and second derivatives to ensure monotonicity.

Positivity can also be enforced using ENO-type methods [3,5,82,87], which enforce data boundedness and positivity by adaptively selecting mesh points to build the stencil used to construct the positive interpolant for each interval. ENO-type methods use divided differences to develop a sufficient condition for data boundedness or positivity that is used to guide the stencil selection process. The software introduced in this work is based on the high-order ENO-type data-bounded interpolation (DBI) and positivity-preserving interpolation (PPI) methods in [82]. The work in [82] provides a positivity-preserving method that uses higher degree polynomials compared to the other ENO-type methods in [3,5,87] and the spline-based methods.

The implementations available for positivity preservation are based on splines [33,41] and polynomial rescaling [60,110]. The spline-based approaches often require solving a linear system of equations to ensure continuity, and an optimization problem in the case of quartic and quintic splines. These spline approaches are often limited to fifth-order polynomial and can be computationally expensive in cases where solving a global optimization problem is required. A full suite of test problems comparing the DBI and PPI methods against different spline-based methods including PCHIP [33], MQS [64], and shape-preseving splines (SPS) [14] has been undertaken by the authors in [77]. The different polynomial rescaling methods allow for polynomial degrees higher than five and are built as part of larger partial differential equation (PDE) solvers [60,110]. As previously mentioned, the polynomial rescaling approaches guarantee positivity only at a given set of points, not over the entire domain. The present work provides an implementation of a high-order software (HiPPIS) based on [82] that is high-order and guarantees positivity over the entire domain where the interpolant is defined. In addition, this work evaluates the use of HiPPIS in the context of function approximation and mapping between different meshes. This evaluation provides an analysis of the mapping error in the case of PDEs and numerical examples demonstrating the benefits and limitations of HiPPIS.

The remaining parts of the paper are organized as follows: Section 6.2 presents a background for the mathematical framework required for the DBI and PPI methods. Section 6.3 provides the algorithms used to build the software, the descriptions of the different components HiPPIS, and the techniques used to enable vectorization, increase locality, and improve overall computational performance. Section 6.4 shows 1D and 2D function approximation examples using the DBI and PPI methods in HiPPIS. Section 6.5 provides an analysis of the mapping error in the context of time dependent PDEs, and Section 6.6 shows examples constructed based on NWP applications. In these examples, the DBI and PPI methods are used to map solution values between different meshes used in NWP. A discussion and concluding remarks are presented in Section 6.7.

6.2 Mathematical Framework

This section provides a summary and the theoretical background of both the DBI and PPI methods.

6.2.1 Adaptive Polynomial Construction

Both the DBI and PPI methods rely on the Newton polynomial [56,103] representation to build interpolants that are positive or bounded by the data values. The ability to adaptively choose stencil points to construct the interpolation, as in ENO methods [39], is the key feature employed to develop the data-bounded and positivity-preserving interpolants.

Consider a 1D mesh defined as follows:

$$\mathcal{M} = \{x_{i-J}, \cdots, x_i, x_{i+1}, \cdots, x_{i+L}\}, \tag{6.1}$$

where $x_{i-J} < \cdots < x_i < x_{i+1} < \cdots < x_{i+L}$, and $\{u_{i-J}, \cdots, u_{i+L}\}$ is the set of data values associated with the mesh points. The subscripts $J, L, i, \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$, and $x_k, u_k \in \mathbb{R}$ for $i - J \leq k \leq i + L$. The DBI and PPI procedure starts by setting the initial stencil $\mathcal{V}_0$,

$$\mathcal{V}_0 = \{x_i, x_{i+1}\} = \{x_0^l, x_0^r\}. \tag{6.2}$$

The stencil $\mathcal{V}_0$ is expanded by successively appending a point to the right or left of $\mathcal{V}_j$ to form $\mathcal{V}_{j+1}$. Once the final stencil $\mathcal{V}_{n-1}$ is obtained, the interpolant of degree n defined on $I_i = \{x_i, x_{i+1}\}$ can be written as

$$U_n(x) = \quad u_i + U[x_0^l, x_0^r]\pi_{0,i}(x) + U[x_1^l, \cdots, x_1^r]\pi_{1,i}(x) + \cdots + U[x_{n-1}^l, \cdots, x_{n-1}^r]\pi_{n-1,i}(x), \tag{6.3}$$

where $\pi_{0,i}(x) = (x - x_i), \pi_{1,i}(x) = (x - x_i)(x - x_1^e), \cdots$ are the Newton basis functions. x_j^e is the point added to expand the stencil $\mathcal{V}_{j-2}$ to $\mathcal{V}_{j-1}$ and can be explicitly expressed as

$$\begin{cases} x_0^e = x_i, \\ x_1^e = x_{i+1}, \\ x_j^e = \mathcal{V}_{j-1} \setminus \mathcal{V}_{j-2}, \quad 2 \leq j \leq n-1. \end{cases} \tag{6.4}$$

The divided differences are recursively defined as follows:

$$\begin{cases} U[x_i] = u_i \\ U[x_i, \cdots, x_{i+j}] = \frac{U[x_{i+1}, \cdots, x_{i+j}] - U[x_i, \cdots, x_{i+j-1}]}{x_{i+j} - x_i}. \end{cases} \tag{6.5}$$

The polynomial $U_n(x)$ can be compactly expressed as

$$U_n(x) = u_i + (u_{i+1} - u_i)S_n(x). \tag{6.6}$$

$S_n(x)$ is defined as

$$S_n(x) = s\Bigg(1 + \frac{(s-1)}{d_1}\lambda_1\Bigg(1 + \frac{(s-t_2)}{d_2}\lambda_2\Bigg(\cdots\Bigg(1 + \frac{(s-t_{n-1})}{d_{n-1}}\lambda_{n-1}\Bigg)\cdots\Bigg), \tag{6.7}$$

where s, t_j, and d_j are expressed as follows:

$$0 \leq s = \frac{x - x_i}{x_{i+1} - x_i} = \frac{x - x_0^e}{x_0^r - x_0^l} \leq 1, \tag{6.8}$$

$$t_j = -\frac{x_i - x_j^e}{x_0^r - x_0^l}, \text{ and} \tag{6.9}$$

$$0 \leq d_j = \frac{x_j^r - x_j^l}{x_0^r - x_0^l}. \tag{6.10}$$

s and d_j are defined such that $s \in [0,1]$ and $d_j \geq 0$. The positivity-preserving and data-bounded interpolants are obtained by imposing some bounds on $\bar{\lambda}_j$, defined as

$$\bar{\lambda}_j = \prod_{k=1}^{j} \lambda_j = \lambda_j \bar{\lambda}_{j-1} = \prod_{k=1}^{j} \lambda_k = \begin{cases} 1 \quad j = 0 \\ \frac{U[x_j^l, \cdots, x_j^r]}{U[x_0^l, x_0^r]} \prod_{k=1}^{j} (x_k^r - x_k^l), \quad 1 \leq j \leq n-1. \end{cases} \tag{6.11}$$

6.2.2 Positivity-Preserving and Data-Bounded Interpolation

The DBI and PPI algorithms are constructed by adaptively selecting stencil points and enforcing the conditions for positivity and data boundedness. Requiring positivity alone can lead to large oscillations and extrema that degrade the approximation. Positivity alone does not restrict how much the interpolant is allowed to grow beyond the data values. The large oscillations can be removed with the DBI and PCHIP methods. However, in the case where a given interval I_i has a hidden extremum, both DBI and PCHIP will truncate the extremum. As in [3, 91], the interval I_i has an extremum when two of the three divided differences $\sigma_{i-1} = U[x_{i-1}, x_i]$, $\sigma_i = U[x_i, x_{i+1}]$, and $\sigma_{i+1} = U[x_i, x_{i+1}]$ of neighboring intervals, are of opposite signs. The constrained PPI algorithm addresses these limitations by allowing the constructed interpolant to grow beyond the data values but not produce extrema that are too large.

The positive polynomial interpolant is constrained as follows:

$$u_{min} \leq U^p(x) = u_i + (u_{i+1} - u_i) S_n(x) \leq u_{max}. \tag{6.12}$$

The bounds u_{min} and u_{max} are defined as

$$\begin{cases} u_{min} = min(u_i, u_{i+1}) - \Delta_{min}, \\ u_{max} = max(u_i, u_{i+1}) + \Delta_{max}. \end{cases} \tag{6.13}$$

The parameters Δ_{min} and Δ_{max} are chosen according to

$$\Delta_{min} = \begin{cases} \epsilon_1 |min(u_i, u_{i+1})| & \text{if } \sigma_{i-1}\sigma_{i+1} < 0 \text{ and } \sigma_{i-1} < 0 \text{ or } \sigma_{i-1}\sigma_{i+1} \geq 0 \text{ and } \sigma_{i-1}\sigma_i < 0 \\ \epsilon_0 |min(u_i, u_{i+1})| & \text{otherwise,} \end{cases} \tag{6.14}$$

and

$$\Delta_{max} = \begin{cases} \epsilon_1 \big| max(u_i, u_{i+1}) \big| & \text{if } \sigma_{i-1}\sigma_{i+1} < 0 \text{ and } \sigma_{i-1} > 0 \text{ or } \sigma_{i-1}\sigma_{i+1} \geq 0 \text{ and } \sigma_{i-1}\sigma_i < 0 \\ \epsilon_0 \big| max(u_i, u_{i+1}) \big| & \text{otherwise.} \end{cases} \tag{6.15}$$

The parameters ϵ_0 and ϵ_1, used for intervals with and without extremum, respectively, are introduced to adjust Δ_{min} and Δ_{max}. This work extended the bounds in [82] by introducing the parameter ϵ_1 to allow for more flexibility on how to bound the interpolants in cases where an extremum is detected. The choice for the positive parameters ϵ_0 and ϵ_1 depends on the underlying function and the input data used for the approximation. As both ϵ_0 and ϵ_1 get smaller, the upper and lower bounds get tighter, and the PPI method converges to the DBI method. The choices for ϵ_0 and ϵ_1 are further discussed in Section 6.3.2. In Equation (6.14), the interval I_i has a local maximum if $\sigma_{i-1}\sigma_{i+1} < 0$ and $\sigma_{i-1} < 0$. Correspondingly, in Equation (6.15), the interval I_i has a local minimum if $\sigma_{i-1}\sigma_{i+1} < 0$ and $\sigma_{i-1} > 0$. In both Equations (6.14) and (6.15), the type of extremum is ambiguous if $\sigma_{i-1}\sigma_{i+1}$, and $\sigma_{i-1}\sigma_i < 0$. Equation (6.12) is equivalent to bounding $S_n(x)$ as follows:

$$m_\ell \leq S_n(x) \leq m_r, \tag{6.16}$$

where the factors m_ℓ and m_r are expressed as

1. : $u_{i+1} > u_i$

$$m_\ell = min\left(0, \frac{u_{min} - u_i}{u_{i+1} - u_i}\right), \text{ and } m_r = max\left(1, \frac{u_{max} - u_i}{u_{i+1} - u_i}\right) \tag{6.17}$$

2. : $u_{i+1} < u_i$

$$m_\ell = min\left(0, \frac{u_{max} - u_i}{u_{i+1} - u_i}\right), \text{ and } m_r = max\left(1, \frac{u_{min} - u_i}{u_{i+1} - u_i}\right). \tag{6.18}$$

The DBI method can be recovered from the PPI methods by setting $m_\ell = 0$ and $m_r = 1$. The positivity-preserving result in Equation (6.12) is obtained by successively imposing bounds on $\bar{\lambda}_j$ in the quadratic, cubic, and higher order terms in the expression of $S_n(x)$ defined in Equation (6.7). The lower and upper bounds are defined according to

$$B_j^- = \begin{cases} (-4(m_r - 1) - 1)d_1 & j = 1 \\ (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{1-t_j}, \text{ if } t_j \in (-\infty, 0] & j > 1 \\ (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{-t_j}, \text{ if } t_j \in (0, +\infty) & j > 1, \end{cases} \tag{6.19a}$$

and

$$B_j^+ = \begin{cases} (-4m_\ell + 1)d_1, & j = 1 \\ (B_{j-1}^+ - \bar{\lambda}_{j-1})\frac{d_j}{1-t_j}, \text{ if } t_j \in (-\infty, 0] & j > 1 \\ (B_{j-1}^- - \bar{\lambda}_{j-1})\frac{d_j}{-t_j}, \text{ if } t_j \in (0, +\infty) & j > 1. \end{cases} \tag{6.19b}$$

B_1^- and B_1^+ are defined as $-d_1$ and d_1 for the DBI method whereas for the PPI method, they are defined as $(-4(m_r - 1) - 1)d_1$ and $(-4m_\ell + 1)d_1$, respectively. We refer the reader to Theorem 1 and 2 in Chapter 4 or [82] for more details on the mathematical foundation used to build the positivity-preserving software.

6.3 Algorithms and Software

This section describes the algorithms and different components used in the data-bounded and positivity-preserving software. The software developed in this work provides 1D, 2D, and 3D implementations of the DBI and PPI methods for uniform and nonuniform structured meshes. The 1D implementation is constructed based on the mathematical framework provided in Section 6.2. The 2D and 3D implementation are obtained via a tensor product of the 1D version.

6.3.1 Algorithms

The algorithms provide the necessary elements to construct the data-bounded or positive interpolants. Rogerson [85] showed that the ENO reconstruction can lead to a left- or right-biased stencil that causes stability issues when used to solve hyperbolic equations. Shu [93] addressed this limitation by introducing a bias coefficient used to target a preferred stencil. As indicated in [82], the left- and right-biased stencil can fail to recover hidden extrema. For a given interval I_i, the left- and right-biased stencil does not include the points x_{i-1} or x_{i+1}, respectively. **Algorithm I** addresses these limitations by extending the algorithm in [82] by introducing more options for the adaptive stencil selection process described below. In addition to the symmetry-based points selection in [82], **Algorithm I** includes ENO-type and locality-based point selection processes.

At any given step j, the next point inserted into $\mathcal{V}_j$ can be to the left or right. Let $\bar{\lambda}_{j+1}^-$ and $\bar{\lambda}_{j+1}^+$ correspond to the case where the stencil inserted is to the left and right, respectively. Let x_p and x_q be the mesh points immediately to the left and right of $\mathcal{V}_j$, respectively. Given $\mathcal{V}_j$, let μ_j^l be the number of points to the left of x_i and μ_j^r the number of points to the right.

$$\begin{cases} \bar{\lambda}^{-}_{j+1} = \bar{\lambda}_{j+1} & \text{with } \mathcal{V}_{j+1} = \{x_p\} \cup \mathcal{V}_j \\ \bar{\lambda}^{+}_{j+1} = \bar{\lambda}_{j+1} & \text{with } \mathcal{V}_{j+1} = \mathcal{V}_j \cup \{x_q\}. \end{cases} \tag{6.20}$$

Algorithm I extends the algorithm in [82] by introducing a user-supplied parameter *st* used to guide the procedure for stencil construction. In the cases where adding both x_p (to the left) or x_q (to the right) are valid, the algorithm makes the selection based on the three cases below.

- If $st = 1$ (default), the algorithm chooses the point with the smallest divided difference, as in the ENO stencil.
- If $st = 2$, the point to the left of the current stencil is selected if the number of points to the left of x_i is smaller than the number of points to the right. Similarly, the point to the right is selected if the number of points to the right of x_i is smaller than the number of points to the left. When the number of points to both the right and left are the same, the algorithm chooses the point with the smallest $\bar{\lambda}_{j+1}$.
- If $st = 3$, the algorithm chooses the point that is closest to the starting interval I_i. It is important to prioritize the closest points in cases where the intervals surrounding I_i vary significantly in size. These variations are found in computational problems where different resolutions are used for different parts of the domain.

Algorithm II describes the 1D DBI and PPI methods built using the mathematical framework in Section 6.2 and **Algorithm I**. **Algorithm II** further extends the constraints in [82] by introducing the user-supplied positive parameter ϵ_1 that is used to impose upper and lower bounds on the interpolants according to Equations (6.14) and (6.15). The positive parameters ϵ_0 and ϵ_1 are used for intervals with and without an extremum, respectively. The user-supplied parameter *im* is used to choose between the DBI and PPI methods.

Algorithm I

Input: μ^l_j, μ^l_j, x_p, x_i, x_q, x_{i+1}, $U[x_p, \cdots, x^r_j]$, $U[x^l_j, \cdots, x_q]$ $\bar{\lambda}^-_{j+1}$, $\bar{\lambda}^+_{j+1}$, and *st*.

1. if $st = 1$
   - if $|U[x_p \cdots, x^r_j]| < |U[x^l_j, \cdots, x_q]|$, then insert a new stencil point to the left;
   - else if $|U[x_p \cdots, x^r_j]| > |U[x^l_j, \cdots, x_q]|$, then insert a new stencil point to the right;

- else insert a new stencil point to the right if $|\bar{\lambda}_{j+1}^{-}| \geq |\bar{\lambda}_{j+1}^{+}|$; otherwise, insert a new point to left;

2. if $st = 2$

   - if $\mu_j^l < \mu_j^r$, then insert a new stencil point to the left;
   - else if $\mu_j^l > \mu_j^r$ then insert a new stencil point to the right;
   - else insert a new stencil point to the right if $|\bar{\lambda}_{j+1}^{-}| \geq |\bar{\lambda}_{j+1}^{+}|$; otherwise, insert a new point to left;

3. else $st = 3$

   - if $|x_p - x_i| < |x_q - x_{i+1}|$, then insert a new stencil point to the left;
   - else if $|x_p - x_i| > |x_q - x_{i+1}|$, then insert a new stencil point to the right;
   - else insert a new stencil point to the right if $|\bar{\lambda}_{j+1}^{-}| \geq |\bar{\lambda}_{j+1}^{+}|$; otherwise, insert a new point to left;

Algorithm II (1D)

Input: $\{x_i\}_{i=0}^{n}$, $\{u_i\}_{i=0}^{n}$, $\{\tilde{x}_i\}_{i=0}^{\tilde{n}}$, d, st ϵ_0, im, and ϵ_1. **Output:** $\{\tilde{u}_i\}_{i=0}^{\tilde{n}}$.

1. Select an interval $[x_i, x_{i+1}]$. Let $\mathcal{V}_0 = \{x_i, x_{i+1}\} = \{x_0^l, x_0^r\}$.

2. If $\sigma_{i-1}\sigma_{i+1} < 0$ or $\sigma_{i-1}\sigma_i < 0$, then the interval I_i has a hidden local extremum. For the boundary intervals, we assume that the divided differences to the left and right have the same sign.

3. Compute u_{min} and u_{max} using Equations (6.13), (6.14), and (6.15).

4. Compute m_r and m_ℓ based on Equations (6.17) and (6.17) or Equations (4.77) and (4.77). For DBI, set $m_r = 1$ and $m_\ell = 0$.

5. Given a stencil $\mathcal{V}_j$,

   - if $B_{j+1}^{-} \leq \bar{\lambda}_{j+1}^{+} \leq B_{j+1}^{+}$ and $B_{j+1}^{-} \leq \bar{\lambda}_{j+1}^{-} \leq B_{j+1}^{+}$, choose the point to add based on **Algorithm I**
   - else if $B_{j+1}^{-} \leq \bar{\lambda}_{j+1}^{-} \leq B_{j+1}^{+}$, then insert a new stencil point to the left;

- else if $B^{-}_{j+1} \leq \bar{\lambda}^{+}_{j+1} \leq B^{+}_{j+1}$, then insert a new stencil point to the right;

6. This process (Steps 3) iterates until the halting criterion that the ratio of divided differences lies outside the required bounds stated above or the stencil has $d+1$ points, with d being the target degree for the interpolant.

7. Evaluate the final interpolant $U^{l}(x)$ (for DBI) or $U^{p}(x)$ (for PPI) at the output points $\tilde{x}_i$ that are in I_i.

8. Repeat Steps 1–7 for each interval in the input 1D mesh.

6.3.2 Software Description

The DBI and PPI software implementation is guided by the algorithms described above. HiPPIS is available at `https://github.com/ouermijudicael/HiPPIS`. The software can be organized into four major parts: 1) computation of of divided differences; 2) calculations of upper and lower bounds for each interval; 3) a stencil construction procedure; and 4) 1D, 2D, and 3D DBI and PPI implementations.

The divided differences are essential to the DBI and PPI methods because they are used in the calculations of $\bar{\lambda}_j$ and the stencil selection process. The divided differences are computed using the standard recurrence form in Equation (6.5) and stored in a table of dimension $n \times (d+1)$ where d is the maximum polynomial degree for each interpolant. Given that the maximum degree is d, it is sufficient to consider the $d+1$ divided differences for the stencil selection process and the construction of the final polynomial interpolant for each interval.

The bounds on each interpolant are obtained from Equation (6.13), (6.14), and (6.15) where the positive parameter ϵ_0 and ϵ_1 are user-supplied values used to adjust the bounds in for the interval with and without extremum, respectively. The adjustment focuses on removing large oscillations as much as possible while still allowing high-degree polynomial interpolants that meet the positivity requirements.

The stencil selection process requires the computation of B^{+}_{j} and B^{-}_{j}, which are both dependent on d_j, t_j, and $\bar{\lambda}_j$. The stencil $\mathcal{V}_j$ is constructed from $\mathcal{V}_{j-1}$ by appending a point to the left or right of $\mathcal{V}_{j-1}$. In the cases where both appending to either the right or left meets the requirements for positivity, the software offer three possible options for choosing from both points that can be set by the user. The first and default option ($st = 1$) chooses the stencil with the smallest divided difference, similar to the ENO-like approach. The second option ($st = 2$) prioritizes the choice that makes the stencil more symmetric around the point x_i. The third option ($st = 3$) chooses the point closest to the starting interval I_i, thus prioritizing locality.

The 1D DBI and PPI methods use the three options as building blocks to construct the final approximation. Once the final stencil has been selected, the interpolant is built using a Newton polynomial representation and then evaluated at the corresponding output points. The Newton polynomial is used here because its coefficients/divided differences are available. The 2D and 3D implementations successively use the 1D version along each dimension to construct the final approximation on uniform and nonuniform structured meshes.

The interfaces for the 1D, 2D, and 3D DBI and PPI subroutines are designed to be similar to widely used interfaces for polynomial interpolation such as PCHIP, which makes the software easy to use and incorporate into larger application codes. The interfaces require

- the input mesh points and the data values associated with those points,
- the maximum polynomial degree to be used for each interpolant,
- the interpolation method to be used (DBI or PPI), and
- the output mesh points.

Listing 6.1 shows examples of how to use the 1D, 2D, and 3D interfaces for DBI and PPI in HiPPIS. The variables *x*, *y*, and *z* are 1D arrays used to define the input meshes and *xout*, *yout*, and *zout* are used to define the output meshes. The variables *v*, *v2D*, and *v3D* correspond to the input data values associated with the input meshes. The parameters *d* and *im* (1, or 2) are used to indicate the target polynomial degree and the interpolation method to be used. For DBI and PPI, the parameter *im* is set to 1 and 2, respectively. The parameters *st*, ϵ_0, and ϵ_1 are optional parameters that are set to 3, 0.01, and 1 by default, as explained below. The choice for the optional parameters depends on the underlying function and the input data.

In problems where different resolutions are used for different parts of the computational domain, *st=3* (default) is a preferable choice. The algorithm prioritizes the closest points to starting interval I_i if *st=3*. This choice is particularly important in regions where the size of the intervals varies significantly. For cases where smoothness is the primary goal, *st=1* is a suitable choice. For *st=1*, **Algorithm I** prioritizes smoothness by choosing the points with the smallest divided differences during the stencil construction process. Both the *st=1* and *st=3* can lead to left- or right-biased stencil. In these instances, *st=2* can be used to remove the bias. For *st=2*, the algorithm prioritizes a symmetric stencil. The default value of *st* is set to 3 because the examples in this study indicate that st=3 lead better approximations compared to st=1, or 2 and locality is often a highly desired property in many computational problems.

The positive parameters ϵ_0 and ϵ_1 are used to bound the interpolants for the intervals with and without extrema, respectively. In [82], and [77] the parameters ϵ_0 and ϵ_1 are set to the default values of 0.01 and 1, respectively. The values of ϵ_0 and ϵ_1 are chosen such that the lower and upper bounds on each interpolant are relaxed enough to allow for a high-order polynomial that does not introduce undesirables oscillations. For profiles that are prone to oscillation such as the smoothed Heaviside function, it is important to choose small values for ϵ_0 and ϵ_1. For $N \times N = 17 \times 17$, the approximation leads to large oscillations if ϵ_0 and ϵ_1 are greater than 10^{-4}. For intervals without extrema, it is important to keep ϵ_0 small to not introduce new extrema. For the intervals with extrema, ϵ_1 needs to be large enough to allow the recovery of the hidden extrema but small enough to not cause undesired large oscillations. Choosing the parameter ϵ_1 is very challenging given that the size of the peaks are not know a priori. The default value of $\epsilon_1 = 1$ is such that the interpolant maximum value is twice $max(u_i, u_{I+1})$. This default value of one is sufficient for the modified Runge and TWP-ICE examples. However, in the case of BOMEX, smaller values of $\epsilon_1 \leq 10^{-5}$ are required to remove undesired oscillations. In practice, it is prudent to start with small values ϵ_0 and ϵ_1 and increase them as needed if the approximation fails to recover hidden extrema or uses low-degree polynomial interpolants.

Fig. 6.1 shows a diagram of the different components of the main module of HiPPIS. The function *divdiff(...)* is used to calculate the divided differences needed for **Algorithms I** and **II**. Once the final stencil is constructed, the function *newtonPolyVal(...)* is used to build and evaluate the positive interpolant at the corresponding output points. The major part of the data-boundedness and positivity preservation including **Algorithms I** and **II** is in

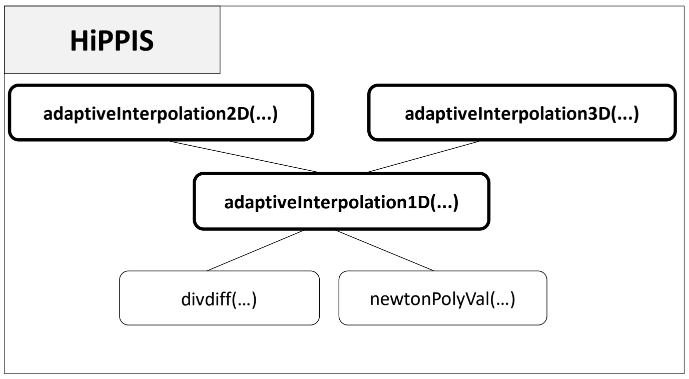

HiPPIS
adaptiveInterpolation2D(...)
adaptiveInterpolation3D(...)
adaptiveInterpolation1D(...)
divdiff(...)
newtonPolyVal(...)

Fig. 6.1: Diagram showing the components of the main module used to build the HiPPIS software.

the function *adaptiveInterpolation1D(...)*. This function is used for the 1D approximation or mapping problems and depends on the function *divdiff(...)* and *newtonPolyVal(...)*. The functions *adaptiveIterpolation2D(...)* and *adaptiveInterpolation3D(...)* use *adaptiveInterpolation1D(..)* to construct the data-bounded or positive polynomial approximations on 2D and 3D structured tensor-product meshes, respectively. The interfaces for the 1D, 2D, and 3D interpolations, in bold, require the parameter *im* which is used to indicate the interpolation method chosen. For the DBI and PPI methods, the parameter *im* is set using 1 and 2, respectively. HiPPIS does not allow for any other choices for the parameter *im*.

Listing 6.1: Interface Examples

```
% 1D example
vout = adaptiveInterpolation1D(x, v, xout, d, im, st, ϵ0, ϵ1 );

%2D example
vout2D = adaptiveInterpolation2D(x, y, v2D, xout,yout, d, im, st, ϵ0, ϵ1 );

%3D example
vout3D = adaptiveInterpolation3D(x, y, z, v3D, xout, yout, zout, d, im, st, ϵ0, ϵ1 );
```

6.3.3 Implementation and Performance

This section provides more details on the implementation and different techniques used to improve the performance of the DBI and PPI methods. HiPPIS is implemented using Fortran90 and Matlab. The Fortran implementation includes a Makefile and examples that are used to build an executable. The Makefiles require an Intel compiler with openMP4. The Makefiles can be modified for other compilers such as gfortran (gnu), HPE Cray, etc. The Fortran version contains all the examples presented in this manuscript. The Matlab version only requires the installation of the Matlab software to be able to use HiPPIS. The

Matlab version does not include the BOMEX example because the DBI and PPI methods are incorporated into a larger simulation code that is in Fortran.

Vectorizing adaptive algorithms is challenging, especially when the conditions used for the adaptive decisions are not known a priori. Both the DBI and PPI approaches are adaptive methods that are not suitable for vectorization and therefore fail to take advantage of vector units in computational resources. Let “i-loops" be the loops that iterate over the intervals from the input mesh, and “j-loops" the loops that iterate from one to the target polynomial degree d. Although ENO methods are adaptive, they can be vectorized along the “i-loops" and “j-loops" [96]. The vectorization along the “j-loops" are possible because the values (divided differences) used for the adaptive decisions are known before entering the loops. This type of vectorization is not suitable for the DBI and PPI methods. When adding a point to go from $\mathcal{V}_{j-1}$ to $\mathcal{V}_j$, the point selection depends on the selection order of previously added points, the conditions for data-boundedness and positivity, and the user-supplied parameters. These dependencies, which are not known a priori, and many nested conditionals inside the DBI and PPI algorithms prevent vectorization along the “j-loops". There are no dependencies between the intervals, which are suitable for parallelism. As a result, vectorization of the DBI and PPI algorithms along the “i-loops" is more complex than the approach used for the ENO method in [96].

Although there are no dependencies between the intervals, the dependencies along the “j-loops", and complex control flows remain. The implementation enables vectorization by structuring the code such that the “j-loops" and “i-loops" are the outer and inner loops respectively, by removing the complex control flows used, by introducing local variables to reduce decencies inside each loop, and by placing the OpenMP directive OMP SIMD before the “i-loops". Similar ideas for code restructuring are introduced in [79, 80] in the context of vectorizing complex numerical weather prediction codes.

Listings 6.2 and 6.3 show a unvectorized and vectorized pseudocode to highlight the transformations used to enable vectorization. In Listing 6.3 the computed values of *msk1, msk2, msk3,* and *msk4* encapsulate the conditionals in lines 4, 5, 8, 12, and 15 of Listing 6.2. The values of *msk1, msk2, msk3, msk4, msk5, and msk6* are set to ones if the conditions in lines 5, 9, 14, 18, 23, and 24 of Listing 6.3 are met; otherwise, the values are set to zeros. *msk5* and *msk6* are used to indicate the final choices based on conditionals that are encapsulated in precomputed values of *msk1, msk2, msk3,* and *msk4*. For a given index *i*, let us consider the case where the conditions in lines 4, and 5 of Listing 6.2 are met and *left(i) = left(i)-1*. In this case, *msk1(i)=msk5(i)=1* and *msk2(i)=msk3(i)=msk5(i)=msk6(i)=0*. Line 28 of Listing 6.3 becomes *left(i) = msk5(i)*(left(i)-1) = left(i)-1*, and the other evaluations collapse to zero.

Listing 6.2: Unvectorized pseudocode example

```
do i=1,n-1
  do j=1, degree
    ...
    if ( cdt1_unvec .and.
        cdt2_unvec)
         if(|ul_unvec| < |ur_unvec
             |)
           left_unvec = left_unvec
               - 1
           ...
         else
           right_unvec =
               right_unvec + 1
           ...
         endif
    elseif(cdt2_unvec )
           right_unvec =
               right_unvec + 1
           ...
    elseif(cdt1_unvec)
           left_unvec = left_unvec
               - 1
           ...
    end
  enddo
enddo
```

Listing 6.3: Vectorized pseudocode example

```
do j=1, degree
  ...
  !$OMP SIMD
  do i=1, n-1
    msk1(i) = (cdt1(i) .and. cdt2(i) .and. |ul|
        < |ur|)
  enddo
  !$OMP SIMD
  do i=1, n-1
    msk2(i) = (cdt1(i) .and. cdt2(i) .and. &
                |ul(i)| >= |ur(i)| .and. msk1(i)
                     ==0)
  enddo
  !$OMP SIMD
  do i=1, n-1
    msk3(i) = (cdt2(i) .and. msk1(i)==0 .and.
        msk2(i)==0)
  enddo
  !$OMP SIMD
  do i=1, n-1
    msk4(i) = (cdt1(i) .and. msk1==0(i) .and. &
               msk2==0(i) .and. msk3(i)==0)
  enddo
  !$OMP SIMD
  do i=1, n-1
    msk5(i) = ( msk1(i) .or. msk4(i))
    msk6(i) = (msk2(i) .or. msk3(i))
  enddo
  !$OMP SIMD
  do i=1, n-1
    left(i) = msk5(i)*(left(i)-1) + msk6(i)*left
        (i) &
              + (1-msk5(i))*(1-msk6(i))*left(i)
    right(i) = msk5(i)*right(i) + msk6(i)*(right
        (i)+1) &
               + (1-msk5(i))*(1-msk6(i))*right(i
                   )
    ...
  enddo
enddo
```

The pseudocode in Listing 6.2 fails to vectorize because of the nested conditionals. The transformations from Listing 6.2 to 6.3 enable vectorization by removing the nested conditionals, swapping the “j-loop", the “i-loop", and placing the OpenMP directive !$OMP SIMD. In Listing 6.3, the conditionals are precomputed in the loops before line 21. These loops are structured such that the nested conditionals are removed, and the variables used to compute the conditions are known prior to entering the loops. Swapping the “i-loop" and “j-loop" requires the introduction of new arrays to track the different changes for each interval as we iterate over the “j-loop". For example, *left_unvec* in Listing 6.2

becomes *left(i)*, with the index *i* going from 1 to n-1. Removing the nested conditionals and complex dependencies may still not be sufficient for compiler auto-vectorization. To ensure vectorization, the OpenMP directive !$OMP SIMD is placed before every “i-loop", as shown in Lisiting 6.3

Table 6.1 shows the runtimes for the PCHIP, the original unvectorized, and the vectorized DBI and PPI codes on a Knights Landing (KNL) core with a 2018 Intel compiler. The KNL architecture has a clock frequency of 1.3 GHz and AVX-512 vector processing units. Further code transformation is required for the gnu compiler as it is unable to vectorize the loops in Listing 6.3. The vectorization report indicates that the combination of control flow and the remaining conditionals prevents vectorizations in the case of the gfortran (gnu) compiler. In the case of the Intel compiler placing the directive $OMP SIMD is sufficient to vectorize the loops Listing 6.3. The performance examples used functions $sin(x)$ and $sin(x)sin(y)$ for the 1D and 2D cases, respectively. The PCHIP, DBI and PPI methods use n and $n \times n$ uniformly spaced points to approximate functions. The approximated functions are evaluated at $n+1$ and $(n+1) \times (n+1)$ uniformly spaced points, for the 1D and 2D cases, respectively. The runtimes for the PCHIP method are smaller than the runtimes for DBI and PPI methods with $\mathcal{P}_4$. The PCHIP method requires less data and has a less complex control flow compared to the DBI and PPI methods. The runtimes for the vectorized version of the DBI and PPI methods are closer to the runtimes for the PCHIP method. The results from Table 6.1 show that reorganizing the code to improve vectorization and locality leads to smaller runtimes compared to unvectorized code. These performance improvements correspond to a minimum and maximum speed-up of 1.89 and 4.13 over the unvectorized version.

6.4 Numerical Examples

This section provides 1D and 2D numerical examples used to evaluate the use of the PCHIP, DBI and PPI methods. These examples include a subset of the full suite of test problems considered in [77]. The interpolation methods are used to approximate positive functions from provided data values that are obtained by evaluating the 1D and 2D functions on a given set of mesh points. Using a standard polynomial interpolation to approximate the different functions leads to negative values and oscillations. The L^2-norms in the tables

Table 6.1: Runtimes in miliseconds (ms) for the different interpolation method. "unvec" and "vec" correspond to unvectorized and vectorized, respectively.

N	PCHIP	PPI and DBI					
		$\mathcal{P}_4$		$\mathcal{P}_8$		$\mathcal{P}_{16}$	
		unvec	vec	unvec	vec	unvec	vec
				1D $f_1(x)$			
17	8.11E-3	2.34E-2	1.24E-2	3.30E-2	1.22E-2	5.82E-2	1.85E-2
33	1.41E-2	4.09E-2	1.65E-2	6.31E-2	2.09E-2	1.18E-1	3.20E-2
65	2.52E-2	7.30E-2	2.72E-2	1.21E-1	3.89E-2	2.31E-1	6.13E-2
127	4.14E-2	1.30E-1	4.59E-2	2.46E-1	7.98E-2	4.74E-1	1.26E-1
257	8.73E-2	2.28E-1	8.80E-2	4.83E-1	1.52E-1	9.75E-1	2.36E-1
			2D $f_4(x,y)$				
17^2	1.89E-1	5.42E-1	2.53E-1	1.11	4.36E-1	2.02	6.48E-1
33^2	7.18E-1	2	8.20E-1	4.19	1.45	7.97	2.18
65^2	2.79	7.52	2.92	1.63E+1	5.22	3.19E+1	7.99
127^2	1.08E+1	2.96E+1	1.19E+1	6.34E+1	2.12E+1	1.24E+2	3.40E+1
257^2	4.45E+1	1.16E+2	4.45E+1	2.54E+2	8.05E+1	5.16E+2	1.26E+2

below are approximated using the trapezoid rule with 10^4 and $10^3 \times 10^3$ uniformly spaced points for the 1D and 2D examples, respectively. For the numerical examples in Sections 6.4.1 - 6.6.2, the errors from using $st = 1, 2$, and 3 are similar with $st = 3$ leading to slightly smaller errors compared to $st = 1$ and $st = 2$. Given that the results are similar, the tables show only errors with the parameter st set to 3. For the BOMEX example, the errors from the three choices are significantly different. Therefore, the results from all three choices are included. More test examples can be found in [77].

6.4.1 Example I: Modified Runge Function

This example uses a modified version of the canonical Runge function defined as

$$g_1(x) = \frac{0.1}{0.1 + 25x^2}, \quad x \in [-1, 1]. \tag{6.21}$$

Approximating the modified Runge function $g_1(x)$ with a global standard polynomial leads to large oscillations. Table 6.2 shows the L^2-errors norms when using the PCHIP, DBI, and PPI methods to approximate $g_1(x)$. The DBI and PPI methods lead to better approximation results compared to the PCHIP method. As the target polynomial degree increases from $d = 4$ to $d = 8$, the DBI approximation does not improve significantly compared to the PPI method. The relaxed nature of the PPI method allows for higher degree polynomial interpolants compared to DBI and PCHIP, which leads to better approximations.

Table 6.2: L^2-errors when using the PCHIP, DBI, and PPI methods to approximate the function $g_1(x)$. N represents the number of input points used to build the approximation. The parameters ϵ_0, ϵ_1, and st are set to 0.01, 1.0, and 3, respectively.

N	PCHIP	DBI			DBI		
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$
17	3.99E-2	5.10E-2	2.91E-2	4.61E-2	5.10E-2	2.91E-2	4.61E-2
33	4.52E-3	6.31E-3	9.57E-3	3.05E-3	6.31E-3	9.57E-3	3.05E-3
65	2.79E-3	2.44E-3	2.49E-3	1.33E-3	2.44E-3	2.49E-3	9.92E-4
129	6.23E-4	2.22E-4	1.21E-4	1.05E-4	2.22E-4	1.21E-4	2.43E-5
257	1.17E-4	1.51E-5	1.15E-5	1.07E-5	1.51E-5	4.68E-6	9.89E-8

6.4.2 Example II: 1D Smoothed Heaviside Function

This test case uses a smoothed version of the Heaviside function defined as

$$g_2(x) = \frac{1}{1+e^{-2kx}}, \quad k = 100, \text{ and } x \in [-0.2, 0.2]. \tag{6.22}$$

The smoothed Heaviside function in Equation (6.22) is challenging because of the steep gradient at about $x = 0$. Approximating $g_2(x)$ with a standard polynomial interpolation leads to large oscillations to the left and right of the gradient. In addition, the oscillations to the left produce negative values.

Table 6.3 shows L^2-error norms when using the PCHIP, DBI and PPI methods to approximate the smoothed Heaviside function $g_2(x)$. For a target polynomial $d = 3$, the approximation errors using PCHIP, DBI, and PPI are comparable. Increasing the target polynomial improves the approximations for DBI and PPI, as shown in Table 6.3. The errors from both the DBI and PPI methods are similar because the smoothed Heaviside example has no hidden extrema, and the stencils used for both methods are the same, around $x = 0$. The global error is dominated by the local errors in the region with the steep gradients around $x = 0$. Fig. 6.2 shows approximation plots of $g_2(x)$ using $N = 17$ uniformly spaced

Table 6.3: L^2-errors when using the PCHIP, DBI, and PPI methods to approximate the function $g_2(x)$. N represents the number of input points used to build the approximation. The parameters ϵ_0, ϵ_1, and st are set to 0.01, 1, and 3, respectively.

N	PCHIP	DBI			DBI		
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$
17	2.02E-2	2.41E-2	2.41E-2	2.14E-2	2.41E-2	2.41E-2	2.17E-2
33	3.38E-3	4.89E-3	4.86E-3	3.59E-3	4.90E-3	4.84E-3	3.63E-3
65	3.59E-4	4.17E-4	4.07E-4	1.47E-4	4.17E-4	4.07E-4	1.47E-4
129	4.21E-5	3.09E-5	1.56E-5	1.70E-6	3.09E-5	1.56E-5	1.70E-6
257	5.12E-6	2.04E-6	5.19E-7	5.22E-9	2.04E-6	5.19E-7	5.22E-9

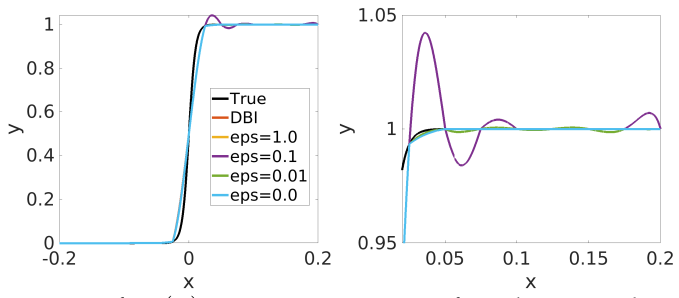


Fig. 6.2: Approximation of $g_2(x)$ using $N = 17$ uniformly spaced points with different values of ϵ_0 for the PPI method. The target polynomial is set to $d = 8$, $st = 2$, and $\epsilon_1 = 1$.

points with different values of ϵ_0 and $\epsilon_1 = 1$. The target polynomial degree is set to $d = 8$. For $\epsilon_0 = 1$, we observe oscillations, as shown in the right part of Fig. 6.2. As ϵ_0 decreases, the oscillations decrease. For $\epsilon_0 \leq 0.01$, the errors oscillations are negligible compared to errors in the region with the steep gradient. The oscillation are completely removed for $\epsilon_0 = 0.0$.

6.4.3 Example III

This example uses a modified version of a function introduced by Tadmor and Tanner [102] and used by Berzins [5] in the context of DBI. The value one is added to the original function in [102] to ensure that the function is positive over the interval [-1,1] The modified function is defined as

$$g_3(x) = \begin{cases} 1 + \frac{2e^{2\pi x} - 1 - e^{\pi}}{e^{\pi} - 1}, & x \in [-1, -0.5) \\ \\ 1 - sin\left(\frac{2\pi x}{3} + \frac{\pi}{3}\right), & x \in [-0.5, 1]. \end{cases} \tag{6.23}$$

Table 6.4 shows L^2-error norms when using the PCHIP, DBI and PPI methods to approximate the smoothed Heaviside function $g_3(x)$. Approximating $g_3(x)$ is challenging because $g_3(x)$ is a piecewise function with a discontinuity at $x = 0.5$. The global error is dominated by the local errors around the discontinuity. The PCHIP, DBI, and PPI approximation results are comparable. Increasing the target polynomial degree does not decrease the L^2-error norms. The approximations in the smooth regions improve as we increase the target polynomial degree, but the global error is dominated by the error around the discontinuity. The error around the discontinuity does not decrease with higher polynomial degrees.

Fig. 6.3 shows approximation plots of $g_3(x)$ using $N = 17$ uniformly spaced points with

Table 6.4: L^2-errors when using the PCHIP, DBI, and PPI methods to approximate the function $g_3(x)$. N represents the number of input points used to build the approximation. The parameters ϵ_0, ϵ_1, and st are set to $0.01, 1$, and 3, respectively.

N	PCHIP		DBI			DBI	
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$
17	1.77E-1	1.82E-1	1.83E-1	1.82E-1	1.73E-1	1.72E-1	1.70E-1
33	1.39E-1	1.35E-1	1.39E-1	1.36E-1	1.35E-1	1.39E-1	1.36E-1
65	1.03E-1	9.95E-2	1.04E-1	1.02E-1	9.95E-2	1.04E-1	1.02E-1
129	7.42E-2	7.12E-2	7.54E-2	7.35E-2	7.15E-2	7.55E-2	7.38E-2
257	5.28E-2	5.06E-2	5.38E-2	5.24E-2	5.07E-2	5.39E-2	5.26E-2

different values of ϵ_0. The target polynomial degree is set to $d = 8$. The right part of Fig. 6.3 shows oscillations at the left boundary for $\epsilon_0 = 1$. The oscillations are removed for $\epsilon_0 \leq 0.1$. As expected, all the interpolation methods have difficulties approximating the function around the discontinuity, as shown in Fig. 6.3.

6.4.4 Example IV: 2D Modified Runge Function

This example extends the previously modified 1D Runge function to 2D as follows:

$$g_4(x, y) = \frac{0.1}{0.1 + 25(x^2 + y^2)}, \quad x, y \in [-1, 1]. \tag{6.24}$$

Table 6.5 shows L^2-error norms when using the PCHIP, DBI and PPI methods to approximate the 2D modified Runge function $g_4(x)$. The DBI and PPI methods with $d = 3$ lead to better approximation results compared to the PCHIP. As the target polynomial degree d increases, the approximation errors from PPI decrease much faster than DBI. The relaxed nature of the PPI methods allows for higher degree polynomials compared to DBI. The bounds for data boundedness are more restrictive than positivity. In addition, the approximation does not lead oscillations for ϵ_0 and $\epsilon_1 \in [0, 1]$.

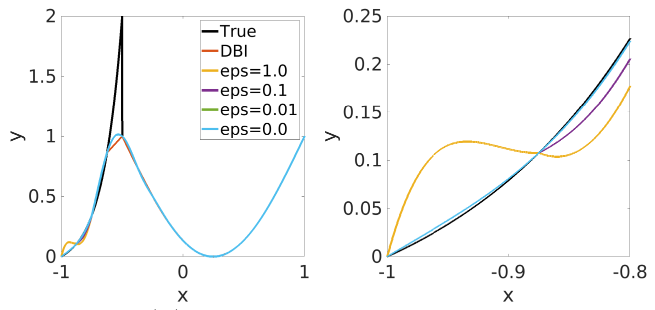


Fig. 6.3: Approximation of $g_3(x)$ using $N = 17$ uniformly spaced points with different values of ϵ_0 for the PPI method. The target polynomial is set to $d = 8$, $st = 2$, and $\epsilon_1 = 1$.

Table 6.5: L^2-errors when using the PCHIP, DBI, and PPI methods to approximate the function $g_4(x, y)$. N represents the number of input points used to build the approximation. The parameters ϵ_0, ϵ_1, and st are set to $0.01, 1$, and 3, respectively.

N	PCHIP	DBI			DBI		
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$
17^2	1.76E-2	1.57E-2	9.09E-3	1.91E-2	2.12E-2	9.09E-3	1.91E-2
33^2	2.05E-3	9.58E-3	4.61E-3	1.25E-3	2.45E-3	4.61E-3	1.24E-3
65^2	1.05E-3	1.19E-3	9.33E-4	4.99E-4	8.59E-4	9.33E-4	3.51E-4
129^2	2.23E-4	1.20E-4	4.76E-5	4.12E-5	7.47E-5	4.64E-5	7.16E-6
257^2	4.19E-5	7.14E-6	4.20E-6	3.80E-6	5.05E-6	1.62E-6	2.91E-8

6.4.5 Example V: 2D Smoothed Heaviside Function

The test case extends the 1D smoothed Heaviside function from Example II to 2D.

$$g_5(x, y) = \frac{1}{1 + e^{-\sqrt{2}k(x+y)}}, \quad x, y \in [-0.2, 0.2] \tag{6.25}$$

The function $g_5(x, y)$ is challenging because of the large gradient at $y = -x$. Approximating $g_5(x, y)$ with a standard polynomial interpolation leads to oscillations and negative values that violate the desired property of positivity.

Table 6.6 shows L^2-error norms when using the PCHIP, DBI, and PPI methods to approximate the 2D smoothed Heaviside function $g_5(x, y)$. The DBI and PPI methods lead to better approximation results compared to the PCHIP approach. Increasing the target polynomial degree improves the approximation for DBI and PPI, as shown in Table 6.6. The global error is dominated by the local around the steep gradients at $y = -x$. The approximations for both DBI and PPI are the same because both methods use the stencil for the intervals around the discontinuity.

Fig. 6.4 shows an approximation plots of $g_5(x, y)$ using $N \times N = 17 \times 17$ uniformly spaced points with different values of ϵ_1 and ϵ_1. The left and right plots are approximated

Table 6.6: L^2-errors when using the PCHIP, DBI, and PPI methods to approximate the function $g_5(x, y)$. N represents the number of input points used to build the approximation. The parameters ϵ_0, ϵ_1, and st are set to $0.01, 1$, and 3, respectively.

N	PCHIP	DBI			DBI		
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$
17^2	8.07E-3	1.05E-2	1.02E-2	8.31E-3	1.05E-2	1.02E-2	8.74E-3
33^2	1.26E-3	1.67E-3	1.58E-3	1.09E-3	1.64E-3	1.53E-3	9.19E-4
65^2	1.44E-4	1.58E-4	1.06E-4	4.94E-5	1.58E-4	1.06E-4	5.05E-5
129^2	1.63E-5	1.13E-5	4.01E-6	2.66E-7	1.13E-5	4.01E-6	2.66E-7
257^2	1.94E-6	7.29E-7	1.27E-7	5.44E-10	7.29E-7	1.27E-7	5.44E-10

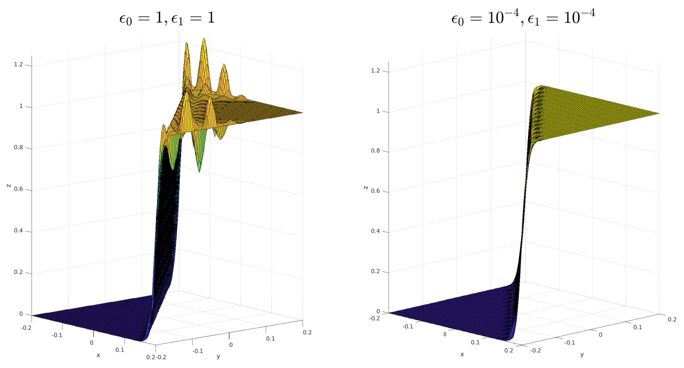


Fig. 6.4: Approximation of $g_5(x, y)$ using $N \times N = 17 \times N$ uniformly spaced points with different values of ϵ_0 and ϵ_1 for the PPI method. The target polynomial is set to $d = 8$ and $st = 2$.

solutions using PPI with $\epsilon_0 = \epsilon_1 = 1$ for the left plot and $\epsilon_0 = \epsilon_1 = 10^{-4}$ for the right plot. The target polynomial degree is set to $d = 8$. For $\epsilon_0 = \epsilon_1 = 10^{-4}$, the oscillations are significantly reduced and the approximation is closer to the target solution.

6.4.6 Example VI

This example uses a 2D function used to study positive and monotonic splines [11, 83]. The function is defined as follows:

$$g_6(x, y) = \begin{cases} 2(y - x) & \text{if } 0 \leq y - x \leq 0.5 \\ 1 & \text{if } y - x \geq 0.5 \\ cos\left(4\pi\sqrt{(x-1.5)^2 + (y-0.5)^2}\right) & \text{if } (x-1.5)^2 + (y-0.5)^2 \leq \frac{1}{16} \\ 0 & otherwise. \end{cases} \tag{6.26}$$

The function $g_6(x, y)$ is challenging because it is only $\mathbb{C}^0$ continuous at various locations.

Table 6.7 shows L^2-error norms when using the PCHIP, DBI, and PPI methods to approximate the 2D smoothed Heaviside function $g_6(x, y)$. The PCHIP, DBI, and PPI methods lead to comparable L^2-error norms. Increasing the target polynomial degree does not significantly improve the approximation for DBI and PPI, as shown in Table 6.7. The global error is dominated by the local around the $\mathbb{C}^0$. The approximation for both DBI and PPI can be improved by using an underlying mesh that better captures the $\mathbb{C}^0$-continuity. Fig. 6.5 shows approximation plots of $g_6(x, y)$ using $N \times N = 17 \times 17$ uniformly spaced points with different values of ϵ_0. The left and right plots show approximated solutions using the PPI method. For the left plot $\epsilon_1 = \epsilon_0 = 1$, and for the right plot, $\epsilon_1 = 1$ $\epsilon_0 = 10^{-4}$

Table 6.7: L^2-errors when using the PCHIP, DBI, and PPI methods to approximate the function $g_6(x, y)$. N represents the number of input points used to build the approximation. The parameters ϵ_0, ϵ_1, and st are set to $0.01, 1$, and 3, respectively.

N	PCHIP		DBI			DBI	
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_8$
17^2	1.91E-2	1.72E-2	1.69E-2	1.63E-2	1.72E-2	1.68E-2	1.59E-2
33^2	6.92E-3	6.16E-3	5.81E-3	5.88E-3	6.16E-3	5.80E-3	5.87E-3
65^2	2.47E-3	2.24E-3	2.14E-3	2.11E-3	2.24E-3	2.14E-3	2.11E-3
129^2	8.99E-4	8.21E-4	7.77E-4	7.63E-4	8.20E-4	7.77E-4	7.63E-4
257^2	3.23E-4	2.97E-4	2.81E-4	2.76E-4	2.96E-4	2.81E-4	2.76E-4

The target polynomial degree is set to $d = 8$. The oscillations observed for $\epsilon_1 = \epsilon_0 = 1$ are removed for small values of ϵ_0, shown in right plot of Fig. 6.4.

6.5 Mapping Error in an Application Example

In addition to the development and study of the DBI and PPI methods, it is important to provide some insight into the behavior of the mapping error in the context of time-dependent PDEs. An example of a time-dependent problem where a positivity-preserving mapping is required is the US Navy Environmental Prediction System Utilizing a Nonhydrostatic Engine (NEPTUNE) [49]. NEPTUNE is a next-generation global NWP system being developed at the Naval Research Laboratory (NRL) and the Naval Postgraduate School (NPS). In NEPTUNE, the physics and dynamics are calculated using different meshes and require mapping the solution values between both meshes. NEPTUNE uses a nonuniform structured meshes that have vertical columns with nonuniformly spaced points inside each column. The mapping must preserve positivity for quantities such density and cloud water

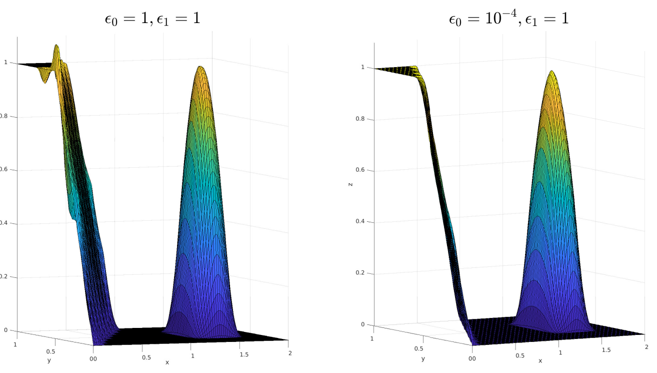


Fig. 6.5: Approximation of $g_6(x, y)$ using $N \times N = 17 \times N$ uniformly spaced points with different values of ϵ_0 for the PPI method. The target polynomial is set to $d = 8$, $st = 2$, and $\epsilon_1 = 1$.

mixing ratio. The cloud water mixing ratio is the amount of cloud water in air. At each time step, the dynamics (advection) solutions, which are calculated on the dynamics mesh, are mapped to the physics mesh to be use as input for the physics calculations. The physics results are then mapped back to the dynamics to be used as input for the next time step. Enforcing positivity alone may still lead to large oscillations and approximation errors. Using the DBI method will remove the large oscillations but will truncate any hidden extremum and may be too restrictive for high order accuracy in some cases. For simulations where different structured meshes are used and mapping is required, the errors from both the DBI and unconstrained PPI will propagate into other calculations and may even cause the simulation to fail. This section provides an analysis of the mapping error when interpolating from one mesh to another and back to the starting mesh. The mapping error is considered within time-dependent PDEs. For example, when interpolating the data values between the dynamics and physics mesh in NEPTUNE, a mapping error is introduced in addition to the physics and time integration errors. The error in approximating a function $u(x)$ with the Newton polynomial $U_n(x)$ over the interval I_i is

$$E_n(x) = u(x) - U_n(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}\prod_{k=0}^{n}(x - x_k^e), \quad x \in I_i \tag{6.27}$$

where $\xi \in [x_n^l, x_n^r]$. Given that ξ and $u^{(n+1)}$ are not known, the local interpolation error can approximated as follows:

$$\tilde{E}_n = U[x_n^l \cdots x_n^r]\prod_{k=0}^{n}\Delta x_k, \tag{6.28}$$

where

$$\Delta x_k = max\bigg(|x(i) - x_k^e|, |x(i+1) - x_k^e|\bigg). \tag{6.29}$$

The error approximation in Equation (6.28) is based on the mean value theorem for divided differences, which states that there exist $\xi_0 \in [x_n^l, x_n^r]$ such that

$$U[x_n^l \cdots x_n^r] = \frac{u^{(n+1)}(\xi_0)}{(n+1)!}. \tag{6.30}$$

Equation (6.28) approximates the local interpolation error for each interval when mapping from one set of points to another. To consider a mapping error for interpolating from one meshes to another and back to the starting mesh, let $\mathcal{M}_D$ and $\mathcal{M}_P$ be the dynamics and physics mesh, respectively. In addition, let I_{DP} and I_{PD} be the interpolation operators that map a given set of data values from $\mathcal{M}_D$ to $\mathcal{M}_P$ and from $\mathcal{M}_P$ to $\mathcal{M}_D$, respectively. We consider an advection-reaction problem where the advection part is calculated on $\mathcal{M}_D$ and the reaction on $\mathcal{M}_P$. A simple forward Euler time integration in used. Let $\bar{u}_\tau$ and $\hat{u}_\tau$ be the approximate and the exact solution at time τ_τ. The dynamics/advection part is written as

$$\bar{u}^1_{\tau+\Delta\tau} = \bar{u}_\tau + \Delta\tau F\big(\bar{u}_\tau\big), \tag{6.31}$$

and the physics/reaction $\bar{w}^{\tau+\Delta\tau}$ is expressed as

$$\bar{w}_{\tau+\Delta\tau} = H\bar{u}^1_{\tau+\Delta\tau}, \tag{6.32}$$

where $H\bar{u}^1_{\tau+\Delta\tau} = I_{DP}G(I_{PD}\bar{u}^1_{\tau+\Delta\tau})$. Let $\bar{E}_{\tau+\Delta\tau}$ be the global space and time error accumulated up to $\tau + \Delta\tau$ after the advection and before mapping the solution values to $\mathcal{M}_P$. $\bar{E}_{\tau+\Delta\tau}$ does not include the mapping errors at $\tau + \Delta\tau$. The final solution after applying the operator H is

$$\bar{u}_{\tau+\Delta\tau} = \bar{u}^1_{\tau+\Delta\tau} + H\bar{u}^1_{\tau+\Delta\tau}. \tag{6.33}$$

The true solution $\hat{u}_{\tau+\Delta\tau}$ at the end of time step $\tau + \Delta\tau$ and after the mapping from $\mathcal{M}_D$ to $\mathcal{M}_P$ and back $\mathcal{M}_D$ to can be expressed as

$$\hat{u}_{\tau+\Delta\tau} = \bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau} + \hat{H}\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big), \tag{6.34}$$

where $\hat{H}$ is assumed to be the corresponding "exact" operator for H. Subtracting Equation (6.34) from (6.33) gives an expression for the true error that can be written as

$$E^G_{\tau+\Delta\tau} = \bar{E}_{\tau+\Delta\tau} + \hat{H}\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big) - H\bar{u}^1_{\tau+\Delta\tau}, \tag{6.35}$$

where E^G is the global space and time error including the mapping errors at $\tau + \Delta\tau$. Adding and subtracting $H\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big)$ yields

$$E^G_{\tau+\Delta\tau} = \bar{E}_{\tau+\Delta\tau} + \hat{H}\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big) - H\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big) + H\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big) - H\bar{u}^1_{\tau+\Delta\tau}. \tag{6.36}$$

Using a Taylor expansion of $H\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big)$ about $\bar{u}^1_{\tau+1}$ and dropping the high order terms, we can approximate the total errors as

$$E^G_{\tau+1} \approx \bar{E}_{\tau+\Delta\tau} + \hat{H}\big(\bar{u}^{\tau+\Delta\tau}_1 + \bar{E}_{\tau+\Delta\tau}\big) - H\big(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}\big) + \frac{\partial H}{\partial u}(u^1_{\tau+\Delta\tau})E_{\tau+\Delta\tau}. \tag{6.37}$$

The results in Equation (6.37) indicate that the total error is dependent on

- the existing global space and time error $\bar{E}_{\tau+\Delta\tau}$ that does not include the mapping error at $\tau + \Delta\tau$,

- the mapping error $E^M_{\tau+\Delta\tau}$ at $\tau + \Delta\tau$,

$$\begin{aligned} E^M_{\tau+\Delta\tau} &= \hat{H}(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}) - H(\bar{u}^1_{\tau+\Delta\tau} + \bar{E}_{\tau+\Delta\tau}) \\ &= \hat{H}\hat{u}^1_{\tau+\Delta\tau} - H\hat{u}^1_{\tau+\Delta\tau}, \text{ and} \end{aligned} \tag{6.38}$$

- a multiplier of the existing global space and time error $\bar{E}_{\tau+\Delta\tau}$,

$$E^N_{\tau+\Delta\tau} = \frac{\partial H}{\partial u}(u^1_{\tau+\Delta\tau})\bar{E}_{\tau+\Delta\tau}. \tag{6.39}$$

Mapping data values from $\mathcal{M}_D$ to $\mathcal{M}_P$ and back to $\mathcal{M}_D$ introduces the interpolation errors that degrade the solution if $E^M_{\tau+\Delta\tau}$ is greater than the existing global space and time error $\bar{E}_{\tau+\Delta\tau}$. This problem is resolved when the mapping error is kept smaller than the existing global space and time error. Similar ideas in the context of time dependent differential equations are explored in [4, 35, 59]. The studies in [35] and [59] develop strategies for balancing the space and time error for better error control and improved performance whereas [4] shows that in mesh adaptivity the spatial interpolation error must be controlled smaller than the temporal error.

6.6 Mapping Examples

This section evaluates the use of the positivity-preserving interpolation to map data values between two different meshes. The Runge and TWP-ICE examples use meshes that emulate the dynamics and physics meshes used in NEPTUNE. These meshes are constructed by linearly scaling the NEPTUNE vertical mesh points to the desired interval for the Runge and TWP-ICE examples. In the BOMEX example, the dynamics mesh is composed of uniformly spaced points, and the physics mesh is constructed using the mid-point of each interval from the dynamics mesh. In Examples 6.6.1 and 6.6.2, we consider the max error instead of L^2-norm error because the global error is dominated by the local errors in few locations around the large gradients.

6.6.1 1D Modified Runge Function

The examples are based on a modified version of the Runge function defined in Equation (6.21) and two meshes that are similar to the dynamics and physics meshes used in

NEPTUNE. The modified Runge function increased the steepness on the left and right side of the $x = 0$ compared to the canonical Runge function $1/(1+25x^2)$. The function $g_1(x)$ is evaluated on the first mesh (dynamics mesh) to create the initial data values. These data values are mapped to the second mesh (physics) mesh and back to the starting mesh.

Table 6.8 shows maximum values of mapping errors over the grid points for g_1 when using the PCHIP, DBI, and PPI methods to map the data values from the dynamics to the physics mesh and back to the dynamics mesh. For $n = 64$ points, increasing the interpolant degree does not significantly improve the approximation. The global error is dominated by the local error in the regions with steep gradients that are to left and right of the peak at $x = 0$. The mapping errors can be improved by increasing the resolution and adding more points in the regions with steep gradients. The resolution is increased by adding one or three uniformly spaced points in each interval from the initial profile with 64 points. Increasing the resolution leads to better approximations when mapping data values between both meshes, and the error decreases as we increase the polynomial degree from 3 to 7. This example demonstrates that in cases with steep gradients, using the PPI method high-order interpolants may not improve the approximation unless there is sufficient resolution. In order to benefit from the positivity and the high-order interpolants, it is important to be in the regime where the problem has sufficient points to observe convergence as the polynomial degree increases. Overall, the PPI method leads smaller errors compared to the other methods.

6.6.2 TWP-ICE Example

This study uses the tropical warm pool international cloud experiment (TWP-ICE) test case from the common community physics package (CCPP). The input mesh for the

Table 6.8: Maximum values of mapping errors for the modified Runge function $g_4(x)$ when using the PCHIP, DBI, and PPI methods to map the data values from dynamics to physics mesh and back to dynamics mesh. The target polynomials are set to $d = 3$, $d = 5$ and $d = 7$. N represents the number of input points used for both meshes. The parameter st is set to 3

N	PCHIP	DBI			PPI		
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_7$	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_7$
64	1.07E-2	1.76E-2	2.59E-2	1.83E-2	1.76E-2	1.45E-2	1.35E-2
127	2.68E-3	3.43E-3	3.44E-3	3.44E-3	1.42E-3	5.50E-4	1.41E-4
253	7.47E-4	8.62E-4	8.57E-4	8.57E-4	1.40E-4	2.27E-5	1.41E-5

simulation is configured to emulate a vertical column in NEPTUNE. The simulation result at time $t = 1440$ sec is extracted scaled, and used to evaluate different interpolation approaches when mapping solution values between dynamics and physics meshes. The domain and range are scaled to $[-1, 1]$ and $[0, 1]$, respectively. This study considers the cloud water mixing ratio profile, which represents the amount of cloud water in air. The extracted profile is then fitted using a radial basis function interpolation to construct an analytical function that can be used as the starting point of the mapping evaluation. The radial basis function is based on multiquadrics.

$$b_i = \sqrt{1 + (\epsilon|x - x_i|)^2}. \tag{6.40}$$

The parameter ϵ is approximated using cross validation [27]. The initial values are obtained by evaluating the analytical function on the dynamics mesh. These values are then mapped to the physics mesh and back to the dynamics mesh.

Table 6.9 shows maximum values of mapping errors for the extracted profile when using the PCHIP, DBI, and PPI methods to map the data values from the dynamics to the physics mesh and back to the dynamics mesh. For $n = 64$, the global error is dominated by the local error at a couple points located in the regions with steep gradients. Increasing the polynomial degree does not significantly improve the approximation compared to using PCHIP for $n = 64$. More points are required to better approximate the underlying profile in the regions with steep gradients. The resolutions are increased by adding one and three uniformly spaced point in each interval from the initial $n = 64$ mesh points. Table 6.9 shows that with the increased resolution, the approximation improves as the polynomial degree increases. The number of points used in each region with steep gradients increased as more points were added. This example provides an application example using simulation data from TWP-ICE. In cases of coarse resolution (64) points, the PPI, DBI, and PCHIP results are comparable, and going to higher degree interpolants does not significantly improve the approximation. The approximation improves with higher degree interpolants when the resolution is increased, as shown in Table 6.9. The results from this experiment suggest that increasing the resolution is needed for the mapping between meshes to benefit from the high-order interpolants from the PPI methods.

Table 6.9: Maximum values of mapping errors for the TWP-ICE profile when using the PCHIP, DBI, and PPI methods to map the data values from dynamics to physics mesh and back to dynamics mesh. The target polynomials are set to $d = 3$, $d = 5$ and $d = 7$. N represents the number of input points used for both meshes. The parameter st is set to 3

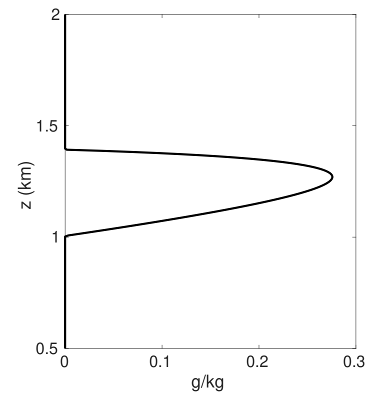
N	PCHIP	DBI			PPI		
	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_7$	$\mathcal{P}_3$	$\mathcal{P}_5$	$\mathcal{P}_7$
64	1.25E-2	4.02E-2	3.77E-2	3.67E-2	1.68E-2	7.20E-3	4.96E-3
127	6.71E-3	1.84E-2	1.89E-2	1.89E-2	3.72E-3	1.39E-3	7.10E-4
253	2.68E-3	4.95E-3	4.77E-3	4.75E-3	6.60E-4	1.09E-4	2.78E-5

6.6.3 BOMEX Example

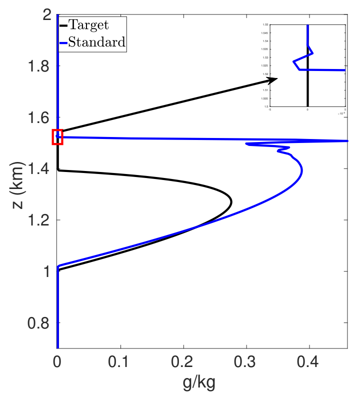
The 1D Barbados Oceanographic and Meteorological Experiment (BOMEX) [32] is a single column test case that was developed to measure and study changes in the properties of heat, moisture, and momentum. In this example, the dynamics and physics results are calculated on different meshes. The dynamics uses uniformly spaced points that indicate the boundary of each level in the vertical column. The physics mesh is constructed using the mid-point of each level. The advections in the dynamics are approximated using fifth-order weighted essentially nonoscillatory (WENO) and third-order Runge-Kutta methods [94]. At each time step, the dynamics are calculated on the dynamics mesh, and the results are interpolated to the physics mesh for the use of the physics routines. The physics terms are calculated using the physics mesh, and the results are interpolated back to the dynamics mesh.

As in [86], let q_c be the cloud water mixing ratio profile in the different experiments. The cloud water mixing ratio represents the amount of cloud water in air. Fig. 6.6a - 6.7f show the cloud mixing ratio profile q_c at $t = 5h$ that is used as input for the physics routines. The physics calculations require positive input values for q_c. Fig. 6.6a shows the target profile for q_c. This target profile is obtained by using the same mesh for both dynamics and physics calculations where mapping is not required and q_c remains positive during the simulation. In addition, as the temporal and spatial resolution increases, q_c converges to the profile shown in Fig. 6.6a. Fig. 6.6b - 6.7f are used to investigate different interpolation methods for mapping the solution values between meshes in the case where the dynamics and physics are calculated using different meshes.

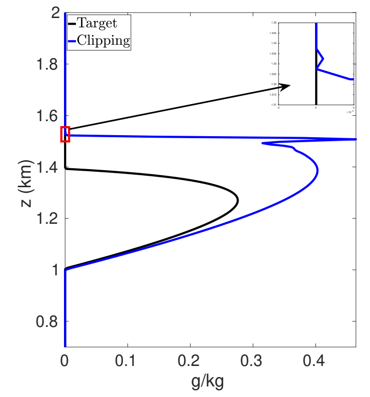
Fig. 6.6b shows the cloud mixing ratio profiles q_c for the target and approximated solution at $t = 5h$. In the case of the approximated solution, a fifth-order standard



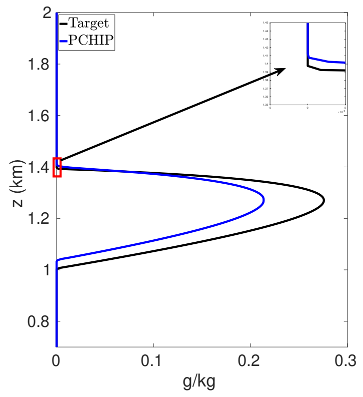
(a) Target (no mapping required).



(b) Standard interpolation.



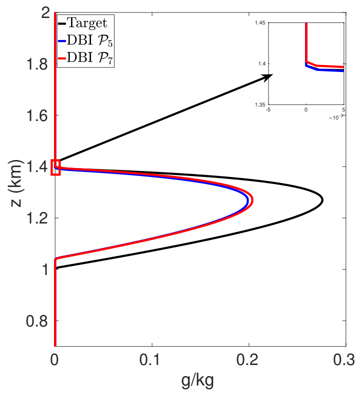
(c) Standard interpolation with “clipping".



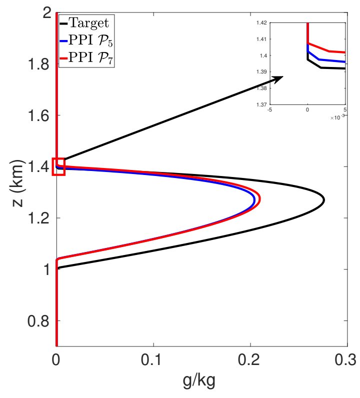
(d) PCHIP.

Fig. 6.6: Cloud mixing ratio q_c profile from the BOMEX test case at $t = 5h$ with $nz = 600$ points. A fifth-order WENO and third-order Runge-Kutta schemes with $CFL = 0.1$ are used for the dynamics (advection). 6.6a the black plot in 6.6b, 6.6c, and 6.6d represents the target profile where the same mesh is used for the dynamics and physics calculations. In 6.6b, 6.6c, and 6.6d, the profiles in blue use different meshes for the dynamics and physics calcultions which require mapping the solution values between both meshes. A standard polynomial interpolation, a standard polynomial interpolation with “clipping", and PCHIP methods are used for the mapping in 6.6b, 6.6c, and 6.6d, respectively.

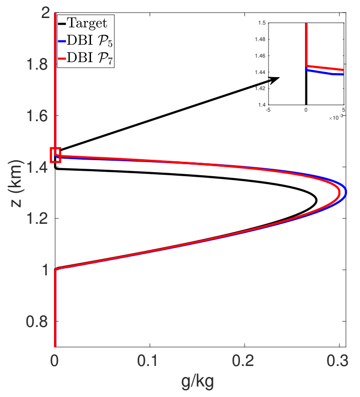
polynomial interpolation is used when mapping between dynamics and physics meshes. For a given interval I_i, the polynomial interpolant is constructed using the stencil $\mathcal{V}_4 = \{x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$. At the boundary and nearby boundary intervals, the stencil $\mathcal{V}_4$ is biased toward the interior of the domain. The results in Fig. 6.6b demonstrate that using the standard polynomial interpolation leads to oscillations, negatives values, and an overestimation of the peak and total cloud mixing ratio of the profile q_c. Using standard polynomial interpolation leads to an overproduction of the total cloud mixing ratio by 93.45%. The peak is $max(q_c) = 0.46g/kg$, which is larger than the target peak



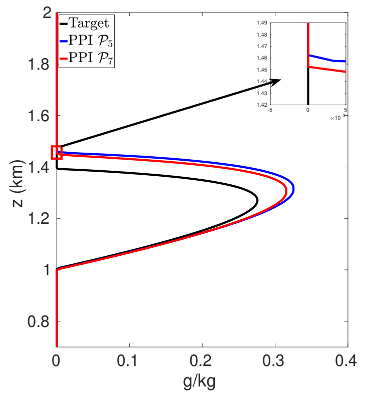
(a)



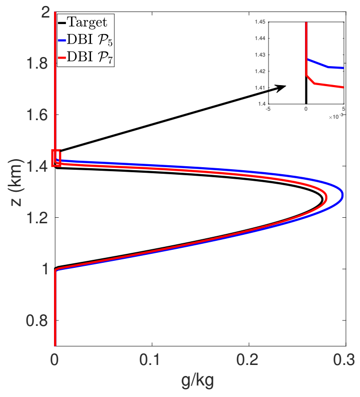
(b)



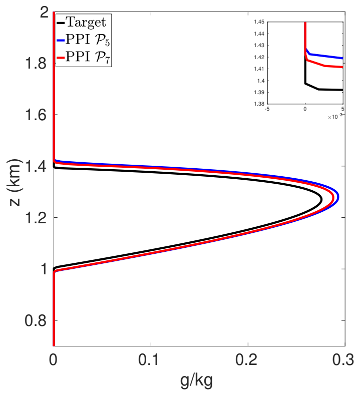
(c)



(d)



(e)



(f)

Fig. 6.7: Cloud mixing ratio q_c profile from the BOMEX test case at $t = 5h$ with $nz = 600$ points with $\epsilon_0 = \epsilon_1 = 10^{-5}$. The profile in black is the target solution. The profiles on the left and right are obtained using the DBI and PPI methods, respectively, to map solution values between meshes. The maximum polynomial degrees are set to 5 and 7 for the blue and red plots, respectively. A fifth-order WENO and third-order Runge-Kutta schemes with $CFL = 0.1$ are used for the dynamics (advection).

$max(q_c) = 0.28g/kg$.

The negative values in Fig. 6.6b can be removed via “clipping", which is a procedure that consists of removing the negative values by setting them to zero [98]. Fig. 6.6c shows the cloud mixing ratio profiles for the target solution and an approximated solution that uses “clipping" to remove the negative values at each time step. The approximated solution uses a standard interpolation to map the data values from one mesh to another. The interpolant for each interval is constructed using the stencil $\mathcal{V}_4 = \{x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ with a fifth-order polynomial. Once the interpolation is completed, “clipping" is used to remove the negative values. Fig. 6.6c shows that using “clipping" still allows for oscillations and a positive bias in the prediction of the cloud mixing ratio q_c. The total cloud mixing ratio is 2.09 times greater than the target solution, and the peak $max(q_c) = 0.46g/kg$ is larger than the target peak $max(q_c) = 0.28g/kg$.

Using PCHIP to map between the dynamics and physics meshes eliminates the negative values, removes oscillations, and reduces the positive bias in the cloud mixing ratio prediction compared to the standard interpolation with and without “clipping". Fig. 6.6d shows the target profile q_c and an approximated profile that uses PCHIP for mapping solution values between dynamics and physics meshes. The total cloud mixing ratio is now 27.21% less than the target with a peak $max(q_c) = 0.21g/kg$. In the BOMEX test case, NEPTUNE, and similar codes, using PCHIP for mapping data values from one mesh to another can degrade the high-order accuracy obtained from the high-order methods used for the dynamics calculations. PCHIP is only third-order whereas the dynamics calculations use a fifth-order method. This limitation can be addressed via high-order DBI and PPI.

Fig.s 6.7a-6.7f show cloud mixing ratio profiles for the target and approximated solutions that use the DBI and PPI methods to map the solution values between meshes. The maximum polynomial degree for the DBI and PPI methods is set to 5 and 7, and the parameters ϵ_0 and ϵ_1 are both set a value of 10^{-5}. For larger values of ϵ_0 and ϵ_1, the PPI approach introduces oscillations that lead to positive bias prediction of the cloud mixing ratio. These oscillation are caused by the relaxed nature of the PPI approach, which still allows the interpolants to oscillate while remaining positive. The positive bias and oscillations can be removed using the DBI or PPI method with small values for ϵ_0 and ϵ_1. When using the PPI method for mapping, the total amount of the cloud mixing ratio is less than the target for $st = 1$ and more than the target for $st = 2$ and $st = 3$. Fig. 6.7a-6.7f show that using the DBI and PPI methods with $\epsilon_0 = \epsilon_1 = 10^{-5}$ to map data values between the dynamics and physics meshes eliminates the negative values, removes the oscillations, and significantly reduces the positive bias in the cloud mixing ratio prediction. Using the DBI and PPI methods leads to a better approximation of the peak value of the total cloud mixing ratio compared to using the standard interpolation and PCHIP approaches. The best approximation of the total amount of the cloud mixing ratio is with the DBI method, which is 7.57% more than the target with a peak of $max(q_c) = 0.28 g/kg$.

In summary, using DBI and PPI methods to map data values between both dynamics and physics meshes produces better approximation results compared to the standard interpolation and PCHIP methods. Tables 6.10 and 6.11 provide a summary of the maximum values and the total amount of cloud mixing ratios for each case. The DBI and PPI methods with a target polynomial set to $d = 7$ lead to a better approximation of the peak and the total the cloud mixing ratios compared to the standard interpolation and PCHIP approaches. The results from Tables 6.10 and 6.11 indicate that the DBI method is the most suitable approach to map data values between meshes for the BOMEX test case. This study provided an example demonstrating how to use the DBI and PPI methods for mapping data values between meshes in the context of NWP. The BOMEX example also demonstrated that positivity alone may not be sufficient to remove oscillations in the solution, and the interpolants may need to be constrained to be between the data values for a better approximation.

Table 6.10: Maximum values of q_c and the total amount of the cloud mixing ratio at $t = 5h$ with $nz = 600$ points. The total amount of the cloud mixing ratio is calculated by estimating the integral q_c. The units of q_c are g/kg.

	Target	STD	Clipping	PCHIP
maximum q_c	0.28	0.46	0.46	0.21
total q_c	69.82	135.07	145.89	50.82

Table 6.11: Maximum values of q_c and the total amount of the cloud mixing ratio at $t = 5h$ with $nz = 600$ points and $\epsilon_0 = \epsilon_1 = 10^{-5}$. The total amount of the cloud mixing ratio is calculated by estimating the integral q_c. The units of q_c are g/kg.

	$st = 1$	$st = 2$	$st = 3$	$st = 1$	$st = 2$	$st = 3$	
		$\mathcal{P}_5$			$\mathcal{P}_7$		Target
				DBI			
maximum q_c	0.20	0.20	0.31	0.30	0.30	0.28	0.28
total q_c	45.91	47.74	87.98	86.57	82.67	75.11	69.82
				PPI			
maximum q_c	0.20	0.21	0.33	0.32	0.29	0.29	0.28
total q_c	47.87	50.09	97.60	92.54	81.44	78.85	69.82

6.7 Discussion and Concluding Remarks

This work presents a high-order 1D, 2D, and 3D data-bounded and positivity-preserving interpolation software HiPPIS for function approximation and mapping data values between different structured meshes. The software implementation is based on the mathematical framework in Section 6.2 and the algorithms in Section 6.3. The software is self-contained and easy to incorporate into larger codes. The interface is designed to be similar to commonly used PCHIP and splines interfaces. The algorithms used in the software extend the DBI and PPI methods introduced in [82] by adding more options for the stencil construction process that can be set by the user with the parameter st. For a given interval I_i, the algorithm starts with the stencil $\mathcal{V}_0 = \{x_i, x_{i+1}\}$ and successively appends points the left and/or right of $\mathcal{V}_0$ to form the final stencil. The stencil construction is done in accordance with the DBI and PPI conditions outlined in Equations (6.19a) and (6.19b). In addition to the different options for stencil selection process, the software introduces a parameter ϵ_1 that can be used to adjust the bounds of the interpolants in the intervals where extrema are detected.

Section 6.5 provides an analysis of the mapping error when the PPI and DBI methods are used to map data values between different meshes. The analysis shows that it is important to keep mapping errors smaller than the already existing global errors from other calculations. Removing negative values and spurious oscillations can help reduce the mapping error.

Various 1D and 2D examples are employed to evaluate the use of the DBI and PPI software in different contexts. The results in Tables 6.8 and 6.9 show that using small values for parameters ϵ_0 and ϵ_1 improves the approximation in cases where the input data are coarse. Small values of ϵ_0 and ϵ_1 further restrict how much the interpolant is allowed to grow beyond the data values. The parameters ϵ_0 and ϵ_1 are used to adjust the lower and upper bounds on each interpolant according to Equations (6.14) and (6.15).

The differences between $st = 1$, $st = 2$, and $st = 3$ are negligible in the case of function approximation, as shown in Tables 6.2 - 6.7. However, in the BOMEX test case, prioritizing a symmetry ($st = 2$) or locality ($st = 3$) leads to better approximations compared to the ENO stencil ($st = 1$) using the DBI PPI methods. Using the ENO stencil ($st = 1$) produces significantly less cloud mixing ratio compared to both the prioritizing symmetry and locality. In the BOMEX example with parameters ϵ_0 and ϵ_1 greater than 10^{-5}, the PPI method allows for oscillations that degrade the approximation compared to the DBI and PCHIP approaches. The study of the modified Runge example in Section 6.6.1 and TWP-ICE example in Section 6.6.2 demonstrated that for a profile with steep gradients or fronts, more points are required to better take advantage of the DBI and PPI algorithm. If there are not enough points in the regions with steep gradients or fronts increasing the polynomial degree may not improve the accuracy. The results in Tables 6.8 and 6.9 show that once the resolution is sufficiently increased, the approximations improve as the polynomial degree increases.

In summary, this work provided: 1) a high-order DBI and PPI software for 1D, 2D, and 3D structured meshes; 2) an analysis of the mapping error when using the DBI or PPI to map data values between meshes; 3) an evaluation of the DBI and PPI methods in the context of function approximation and interpolating data values between different meshes; and 4) code and data restructuring techniques used to enable vectorization, increase locality, and improve overall computational performance. As this work continues, we plan to investigate different approaches for extending the DBI and PPI methods to unstructured 2D and 3D meshes.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Summary and Contributions

The advances in NWP are largely dependent on the progress in computational resources and the ability to develop different methods to effectively use these resources. The exascale systems offer a new opportunity to further improve NWP skills and reach the desired 1 km resolution for global weather and climate model. This body of work highlights some of the challenges in preparing NWP code for the exascale era and beyond. The research in the dissertation is organized into two major parts. The first part, composed of Chapters 2 and 3, focuses on indentifying performance bottlenecks and developing strategies to remove those bottlenecks. The second part, composed of Chapters 4, 5, and 6, introduces a novel approach for mapping solutions values between meshes and evaluating the use of this approach in the context of coupling physics and dynamics in NWP code.

Chapter 2 examines the impact of OpenMP directives on a Fortran-based WSM6 microphysics code in WRF. The synthetic examples measured the cost of thread overhead and tested the effectiveness of various directives with and without OMP SIMD. This study suggests that whereas greater scalability may be possible with high-level OpenMP constructs, parallelization of dependency-free code sections is possible with a few modifications to the original code. Extending the lesson learned from the synthetic examples to WSM6 delivers 50x - 100x speed-ups over serial code.

Chapter 3 investigates high-level and low-level optimization strategies for NWP codes. These strategies employ thread-local structures of arrays (SOA) and OpeMP directives such as OPM SIMD, and minor code transformations to enable better utilizations of SIMD units, increase parallelism, improve locality, and reduce memory traffic. The studies and examples in Chapter 3 demonstrate the benefits of high-level optimization using thread-local SOA, coupled with low-level SIMD using OMP SIMD. The optimized versions of WSM6, GFS physics, and GFS radiation run 70, 27, and 23 times faster, respectively, on KNL and 26, 18, and 30, times faster, respectively on Haswell compared to their respective original serial versions. Although this work targets WRF physics schemes, the findings are transferable to other performance optimization contexts and provide insight into the optimization of codes with complex physical model models for present and near-future architectures with many core and vector units.

Chapter 4, which starts the second part of the dissertation, introduces new data-bounded and positivity-preserving interpolation methods for function approximation and for mapping solution values between meshes. This chapter demonstrates that it is possible to construct high-order interpolation methods over arbitrarily spaced interpolation points in a way that ensures either data boundedness or positivity preservation within user-supplied bounds. The algorithm developed comes with theoretical estimates, presented herein, that provide sufficient conditions for data boundedness and positivity preservation. This work extends the ideas in [5] by addressing data boundedness and positivity (within user-supplied bounds) in the same framework and by allowing irregular meshes. Thus, these new proofs provide the previously missing theoretical underpinning for complex interpolation cases such as those like the NWP case described above. The new approach used here both generalizes the DBI method to nonuniform meshes and extends the approach to preserve positivity (positivity-preserving interpolation PPI) rather than the more restrictive data-bounded approach in [5].

Chapter 5 evaluates the new approach against several typical algorithms in use on a range of test problems in one or more space dimensions. The results obtained show that the new method is competitive in terms of observed accuracy while at the same time preserving the underlying positivity of the functions being interpolated. The different test functions include smooth, C^0-continuous, discontinuous, and steep-gradients. The comparison undertaken focuses on how accurately the different methods can represent this underlying set of test functions. This study shows that the new methods are well suited for function approximation and mapping data values between meshes. The generality of this approach suggests that these methods also have application to other problems for which preserving positivity is important.

Chapter 6 introduces open-source software for high-order data-bounded and positivity-preserving interpolation (HPPIS) that addresses the limitations of both the spline and polynomial rescaling methods. HPPIS uses a given set of data points to construct high-degree polynomial interpolants that are positive over the domains in which they are defined. The high-order positive interpolants obtained from HPPIS are suitable for approximating and mapping physical quantities such as mass, density, and concentration between meshes while preserving positivity. HPPIS provides a Fortran and Matlab implementation of the data-bounded and positivity-preserving methods. Both the Fortran and Matlab versions are self-contained and easy to integrate into other application software requiring positivity. In addition to the software, this work provides an analysis of the mapping error in the context of PDEs, uses several 1D and 2D numerical examples to demonstrate the benefits and limitations of HPPIS, and introduces different strategies to improve locality, vectorization, and, overall, the performance of the data-bounded and positivity-preserving interpolation methods in HPPIS.

7.2 Lessons Learned

The research undertaken in this dissertation provided valuable insights to consider when optimizing performance scientific application codes and developing high-order numerical methods for these scientific applications.

- OpenMP is a suitable choice for node level performance on many- and multicore architectures. Chapters 2 and 3 demonstrated that OpenMP directives can be used to enable and improve thread and vector parallelism with minor changes in NWP codes and similar applications for many- and multicore architectures. The evaluation of the overhead associated with using OpenMP directives indicates how much work is required to minimize the overhead and benefit from thread and vector parallelism.

- Complex control flows are detrimental to performance at the node level because they prevent vectorization. Chapters 2 and 3 showed that to benefit from vector parallelism, it is important to remove or reduce conditionals and dependencies used inside the loops.

- The performance can be further improved by organizing the code and data structures that maximize spatial and temporal locality while minimizing memory traffic. Chapter 3 demonstrated that using high- and low-level code and data restructuring coupled thread-local structures of arrays can help improve performance.

- In many cases, enforcing data boundedness may be too restrictive and truncate hidden extrema whereas enforcing positivity alone may lead to large oscillations. We observe this behavior because enforcing positivity alone does not restrict how much the polynomial is allowed to grow beyond the data values. In addition to enforcing positivity, it is important to remove the undesirable oscillations and extrema as much as possible. Chapter 4 addresses this limitation by introducing an user-supplied parameter that is used to constrain the positive interpolant as needed.

- The extensive testing in Chapters 2 and 3 demonstrated that the DBI and PPI methods are well suited for approximation and mapping solution values between different meshes. Both methods provide high-order accuracy while enforcing data boundedness and positivity preservation. In regions with sharp gradients, discontinuities, and C^0-continuity, the approximations from both approaches are comparable to cubic and quintic shape-preserving spline methods.

- For time-dependent PDEs where positivity-preserving mapping is required, it is important to keep interpolation error smaller than the already existing errors from time integration and previous time steps. Chapter 6 showed that if the interpolation error dominates, it may be amplified after several steps and cause the simulation to fail. Increasing the resolution and constraining the interpolant are approaches that can be used to improve the interpolation accuracy.

7.3 Future Directions

Although this body of work provides several methods addressing the various performance and numerical challenges in NWP codes, several bottlenecks and interesting research questions remain need to be solved.

- KNL is no longer the target architecture for NEPTUNE and will not be used in the Exascale systems. Evaluating the performance strategies introduced in Chapters 2 and 3 on the intended architectures may further highlight new performance challenges and help prepare NWP codes for the Exascale systems. In addition, it is important to explore performance portable approaches for NEPTUNE across hybrid systems that use different architectures, including CPU and GPU.

- The DBI and PPI methods presented in this dissertation build on ideas from ENO-type reconstructions that are challenging to vectorize and less accurate compared to WENO-type reconstruction. Building DBI and PPI methods based on WENO-type reconstruction will help improve accuracy and performance.

- In many scientific applications, there is a strong incentive to use high-order methods. However, in problems with coarse resolutions, sharp gradients, and discontinuities, high-order approximations may degrade the accuracy. A convergence study to investigate and determine the resolution and conditions required for high-order polynomial approximation to improve as the polynomial decrease would provide a guide for when and how to use high-order interpolation for scientific applications such as NWP.

- NWP code and many other scientific applications are constructed by coupling several other applications that have been developed independently. These coupling processes often do not investigate how the errors from the different models interact and affect the applications. For example, an interesting research question would be to evaluate the effective order of accuracy for coupling the dynamical cores with the physics schemes.

- The positivity preserving mapping in NEPTUNE is required because the dynamics uses a spectral element mesh that is different from the uniform mesh used for physics calculation. Another approach worth investigating would be to consider using the same meshes for both dynamics and physics. This change would remove the need for mapping solution values between meshes and the errors introduced from such mapping.

7.4 Publications

1. **Timbwaoga A. J. Ouermi**, Robert M. Kirby, and Martin Berzins. “HPPIS: A High-Order Positivity-Preserving Mapping Software for Structured Meshes." *Submited for*

publication (2022)

2. **Timbwaoga A. J. Ouermi**, Robert M. Kirby, and Martin Berzins. “ENO-Based High-Order Data-Bounded and Constrained Positivity-Preserving Interpolation." *Numerical Algorithms. 2022.*

3. **Timbwaoga. A. J. Ouermi**, Robert M. Kirby, and Martin Berzins. “Numerical Testing of a New Positivity-Preserving Interpolation Algorithm." *Technical Report*. arXiv preprint arXiv:2009.08535. 2020.

4. **Timbwaoga A. J. Ouermi**, Robert M. Kirby, and Martin Berzins. "Performance Optimization Strategies for WRF Physics Schemes Used in Weather Modeling. *International Journal of Networking and Computing 8(2), pp. 301-327. 2018.*

5. **Timbwaoga A. J. Ouermi**, Aaron Knoll, Robert M. Kirby, and Martin Berzins. “Openmp 4 Fortran Modernization of WSM6 for KNL." In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, pp. 1-8. 2017.*

6. **Timbwaoga A. J. Ouermi** , Aaron Knoll, Robert M. Kirby, and Martin Berzins. “Optimization Strategies for WRF Single-Moment 6-Class Microphysics Scheme (WSM6) on Intel Microarchitectures." In *2017 Fifth International Symposium on Computing and Networking (CANDAR), pp. 146-152. IEEE, 2017.*

REFERENCES

[1] D. S. Balsara, "Self-adjusting, positivity preserving high order schemes for hydrodynamics and magnetohydrodynamics," *J. Comput. Phys.*, vol. 231, no. 22, pp. 7504–7517, 2012.

[2] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, no. 7567, pp. 47–55, Sep. 2015.

[3] M. Berzins, "Nonlinear data-bounded polynomial approximations and their applications in eno methods," *Numer. Algor.*, vol. 55, no. 2, pp. 171–189, 2010.

[4] M. Berzins, P. J. Capon, and P. K. Jimack, "On spatial adaptivity and interpolation when using the method of lines," *Appl. Numer. Math.*, vol. 26, no. 1, pp. 117–133, 1998.

[5] M. Berzins, "Adaptive polynomial interpolation on evenly spaced meshes," *SIAM Review*, vol. 49, no. 4, pp. 604–627, 2007.

[6] J. M. Bull, "Measuring synchronisation and scheduling overheads in OpenMP," in *Proceeedings of First European Workshop on OpenMP*, Sep. 1999, pp. 99–105.

[7] J. M. Bull and D. O'Neil, "A microbenchmark suite for OpenMP 2," in *Proceeedings of Third European Workshop on OpenMP (EWOMP'01)*, Sep. 2001.

[8] J. M. Bull, F. Reid, and N. McDonnell, "A microbenchmark suite for OpenMP tasks," in *Proceedings of the 8th International Conference on OpenMP in a Heterogeneous World*, ser. IWOMP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 271–274.

[9] D. Buono, M. Danelutto, T. D. Matteis, G. Mencagli, and M. Torquati, "A lightweight run-time support for fast dense linear algebra on multi-core." IASTED, ACTA Press, Feb. 2014.

[10] S. Butt and K. Brodlie, "Preserving positivity using piecewise cubic interpolation," *Comput. Graph.*, vol. 17, no. 1, pp. 55 – 64, 1993.

[11] E. Chan and B. Ong, "Range restricted scattered data interpolation using convex combination of cubic Bézier triangles," *J. Comput. Appl. Math.*, vol. 136, no. 1, pp. 135 – 147, 2001.

[12] G. Chrysos, "Intel Xeon Phi coprocessor (codename knights corner)," in *2012 IEEE Hot Chips 24 Symposium (HCS)*, Aug. 2012, pp. 1–31.

[13] P. Costantini, "On some recent methods for bivariate shape-preserving interpolation," in *Multivariate Approximation and Interpolation: Proceedings of an International Workshop held at the University of Duisburg, August 14–18, 1989*, W. Haußmann and K. Jetter, Eds. Basel: Birkhäuser Basel, 1990, pp. 55–68.

[14] ——, “Algorithm 770: Bvspis–a package for computing boundary-valued shape-preserving interpolating splines,” *ACM Trans. Math. Softw.*, vol. 23, no. 2, pp. 252–254, Jun 1997.

[15] ——, “Boundary-valued shape-preserving interpolating splines,” *ACM Trans. Math. Softw.*, vol. 23, no. 2, pp. 229–251, Jun 1997.

[16] P. Costantini and F. Fontanella, “Shape-preserving bivariate interpolation,” *SIAM J. Numer. Anal.*, vol. 27, no. 2, pp. 488–506, 1990.

[17] L. Dagum and R. Menon, “OpenMP: An industry-standard api for shared-memory programming,” *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

[18] M. Danelutto, T. D. Matteis, D. D. Sensi, G. Mencagli, and M. Torquati, “P3arsec: towards parallel patterns benchmarking,” in *In Proceedings of the Symposium on Applied Computing (SAC '17)*. New York, NY, USA: ACM, Apr. 2017, pp. 1582–1589.

[19] C. De Boor, *A practical guide to splines*. springer-verlag New York, 1978, vol. 27.

[20] V. V. Dimakopoulos, P. E. Hadjidoukas, and G. C. Philos, *A Microbenchmark Study of OpenMP Overheads under Nested Parallelism*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–12.

[21] R. Dougherty, A. Edelman, and J. Hyman, “Nonnegativity-, monotonicity-, or convexity-preserving cubic and quintic hermite interpolation,” *Math. Comput.*, vol. 52, no. 186, pp. 471–494, 1989.

[22] P. E., M. J., H. M., H. B, H. H. L. A., and L. T., “GPU-accelerated longwave radiation scheme of the rapid radiative transfer model for general circulation models (rrtmg),” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 8, pp. 3660–3667, Aug. 2014.

[23] J. F. Epperson, “On the Runge example,” *The Amer. Math. Monthly*, vol. 94, no. 4, pp. 329–341, 1987.

[24] R. T. Farouki, C. Giannelli, C. Manni, and A. Sestini, “Identification of spatial ph quintic Hermite interpolants with near-optimal shape measures,” *Comput. Aided Geom. Des.*, vol. 25, no. 4, pp. 274 – 297, 2008.

[25] R. T. Farouki, C. Manni, M. L. Sampoli, and A. Sestini, “Shape-preserving interpolation of spatial data by Pythagorean-hodograph quintic spline curves,” *IMA J. Numer. Anal.*, vol. 35, no. 1, pp. 478–498, Feb. 2014.

[26] R. T. Farouki, C. Manni, and A. Sestini, “Shape-preserving interpolation by G1 and G2 PH quintic splines,” *IMA J. Numer. Anal.*, vol. 23, no. 2, pp. 175–195, 04 2003.

[27] G. E. Fasshauer and J. G. Zhang, “On choosing “optimal" shape parameters for rbf approximation,” *Numer. Algor.*, vol. 45, no. 1, pp. 345–368, 2007.

[28] U. S. Fjordholm, S. Mishra, and E. Tadmor, “Arbitrarily high-order accurate entropy stable essentially nonoscillatory schemes for systems of conservation laws,” *SIAM J. Numer. Anal.*, vol. 50, no. 2, pp. 544–573, 2012.

[29] ——, "Eno reconstruction and eno interpolation are stable," *Found. Comput. Math.*, vol. 13, no. 2, pp. 139–159, 2013.

[30] B. Fornberg, "Generation of finite difference formulas on arbitrarily spaced grids," *Math. comput.*, vol. 51, no. 184, pp. 699–706, 1988.

[31] M. P. Forum, "Mpi: A message-passing interface standard," Knoxville, TN, USA, 1994.

[32] H. A. Friedman, G. Conrad, and J. D. McFadden, "Essa research flight facility aircraft participation in the Barbados oceanographic and meteorological experiment," *Bulletin of the American Meteorological Society*, vol. 51, no. 9, pp. 822–834, 1970.

[33] F. N. Fritsch and R. E. Carlson, "Monotone piecewise cubic interpolation," *SIAM J. Numer. Anal.*, vol. 17, no. 2, pp. 238–246, 1980.

[34] F. X. Giraldo, J. F. Kelly, and E. M. Constantinescu, "Implicit-explicit formulations of a three-dimensional nonhydrostatic unified model of the atmosphere (numa)," *SIAM Journal on Scientific Computing*, vol. 35, no. 5, pp. B1162–B1194, 2013.

[35] C. E. Goodyer and M. Berzins, "Adaptive timestepping for elastohydrodynamic lubrication solvers," *SIAM J. Sci. Comput.*, vol. 28, no. 2, pp. 626–650, 2006.

[36] N. Hale and A. Townsend, "Fast and accurate computation of Gauss–Legendre and Gauss–Jacobi quadrature nodes and weights," *SIAM J. Sci. Comput.*, vol. 35, no. 2, pp. A652–A674, 2013.

[37] A. Harten, "Eno schemes with subcell resolution," *J. Comput. Phys.*, vol. 83, no. 1, pp. 148–184, 1989.

[38] ——, "Multiresolution algorithms for the numerical solution of hyperbolic conservation laws," *Commun. Pure Appl. Math.*, vol. 48, no. 12, pp. 1305–1342, 1995.

[39] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy, "Uniformly high order accurate essentially non-oscillatory schemes, iii," *J. Comput. Phys.*, vol. 131, no. 1, pp. 3 – 47, 1997.

[40] T. Henretty, K. Stock, L.-N. Pouchet, F. Franchetti, J. Ramanujam, and P. Sadayappan, "Data layout transformation for stencil computations on short-vector SIMD architectures," in *Proceedings of the 20th International Conference on Compiler Construction: Part of the Joint European Conferences on Theory and Practice of Software*, ser. CC'11/ETAPS'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 225–245.

[41] W. Heß and J. W. Schmidt, "Positive quartic, monotone quintic C2-spline interpolation in one and two dimensions," *J. Comput. Appl. Math.*, vol. 55, no. 1, pp. 51 – 67, 1994.

[42] M. Hirzel, "Data layouts for object-oriented programs," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 265–276, Jun. 2007.

[43] H. Homann and F. Laenen, "Soax: A generic C++ structure of arrays for handling particles in HPC codes," *CoRR*, vol. abs/1710.03462, 2017.

[44] S.-Y. Hong and J.-O. jade Lim, "The WRF single-moment 6-class microphysics scheme (wsm6)," *J. of the Korean Meteorological Society*, vol. 42, no. 2, pp. 129–151, Apr. 2006.

[45] X. Y. Hu, N. A. Adams, and C.-W. Shu, "Positivity-preserving method for high-order conservative schemes solving compressible euler equations," *J. Comput. Phys.*, vol. 242, pp. 169–180, 2013.

[46] M. Z. Hussain, M. Hussain, and Z. Yameen, "A C2-continuous rational quintic interpolation scheme for curve data with shape control," *Journal of The National Science Foundation of Sri Lanka*, vol. 46, p. 341, 2018.

[47] M. Z. Hussain and M. Sarfraz, "Positivity-preserving interpolation of positive data by rational cubics," in *The Proceedings of the Twelfth International Congress on Computational and Applied Mathematics*, vol. 218, no. 2, 2008, pp. 446 – 458.

[48] M. Hussain, M. Z. Hussain, and R. J. Cripps, "C2 rational quintic function," *Journal of Prime Research in Mathematics*, vol. 5, pp. 115–123, 2009.

[49] James D. Doyle and P. A. Reinecke, K. C. Viner, S. Gabersek, M. Martini, D. D. Flagg, J. Michalakes, D. R. Ryglicki, and F. X. Giraldo, "Next generation nwp using a spectral element dynamical core," Jan. 2017.

[50] J. Jeffers, J. Reinders, and A. Sodani, "Chapter 4 - knights landing architecture," in *Intel Xeon Phi Processor High Performance Programming (Second Edition)*, second edition ed., J. Jim, R. James, and A. Sodani, Eds. Boston: Morgan Kaufmann, 2016, ch. 4, pp. 63–84.

[51] H.-M. H. Juang and S.-Y. Hong, "Forward semi-lagrangian advection with mass conservation and positive definiteness for falling hydrometeors," *Monthly Weather Review*, vol. 138, no. 5, pp. 1778 – 1791, 2010.

[52] H.-M. H. Juang and S.-Y. Honmg., "Forward semi-lagrangian advection with mass conservation and positive definiteness for falling hydrometeors," *Monthly Weather Review*, vol. 138, no. 04, pp. 1778–1791, 2010.

[53] A. Karim, S. Ariffin, K. Voon Pang, and A. Saaban, "Positivity preserving interpolation using rational bicubic spline," *J. Appl. Math.*, vol. 2015, 2015.

[54] S. A. A. Karim and K. P. Pang, "Shape preserving interpolation using rational cubic spline," *J. Appl. Math.*, vol. 2016, 2016.

[55] K.C. Viner and P.A. Reinecke and J.D. Doyle and S. Gabersek and M. Martini and D.D. Flagg and J. Michalakes and D.R. Ryglicki and F.X. Giraldo, "Next generation nwp using a spectral element dynamical core," pp. A34A–02, Dec. 2016.

[56] F. T. Krogh, "Efficient algorithms for polynomial interpolation and numerical differentiation," *Math. Comput.*, vol. 24, no. 109, pp. 185–190, 1970.

[57] J. LaGrone, A. Aribuki, and B. Chapman, "A set of microbenchmarks for measuring OpenMP task overheads," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. Citeseer, Nov. 2011, pp. 594–600.

[58] P. Lancaster and K. Šalkauskas, "Curve and surface fitting: an introduction," 1986.

[59] J. Lawson, M. Berzins, and P. M. Dew, "Balancing space and time errors in the method of lines for parabolic equations," *SIAM J. Sci. Statist. Comput.*, vol. 12, no. 3, pp. 573–594, 1991.

[60] D. Light and D. Durran, "Preserving nonnegativity in discontinuous galerkin approximations to scalar transport via truncation and mass aware rescaling (tmar)," *Monthly Weather Review*, vol. 144, no. 12, pp. 4771–4786, 2016.

[61] P. Lin, P. Yew, P. R. Woodward, and J. Jayaraj, "Moving scientific codes to multicore microprocessor cpus," *Computing in Science and Engineering*, vol. 10, no. 6, pp. 16–25, 2008.

[62] H. Liu, Z. Gao, C. Jiang, and C. Lee, "Numerical study of combustion effects on the development of supersonic turbulent mixing layer flows with weno schemes," *Computers and Fluids*, vol. 189, pp. 82–93, 2019.

[63] X.-D. Liu, S. Osher, and T. Chan, "Weighted essentially non-oscillatory schemes," *J. Comput. Phys.*, vol. 115, no. 1, pp. 200–212, 1994.

[64] T. C. H. Lux, L. T. Watson, and T. H. Chang, "An algorithm for constructing monotone quintic interpolating splines," in *SpringSim '20: Proceedings of the 2020 Spring Simulation Conference, May 2020*, 2019, pp. 1–12.

[65] G. Mencagli, M. Vanneschi, and S. Lametti, "The home-forwarding mechanism to reduce the cache coherence overhead in next-generation cmps," *Future Gener. Comput. Syst.*, 2017.

[66] J. Michalakes and M. Vachharajani, "GPU acceleration of numerical weather prediction," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, Apr. 2008, pp. 1–7.

[67] J. Michalakes, M. J. Iacono, and E. R. Jessup, "Optimizing weather model radiative transfer physics for Intel's many integrated core (MIC) architecture," *Parallel Processing Letters*, vol. 27, no. 04, p. 1650019, 2016.

[68] ——, "Optimizing weather model radiative transfer physics for Intel's many integrated core (MIC) architecture," *Parallel Processing Letters*, vol. 26, no. 04, p. 1650019, 2016.

[69] J. Mielikainen, B. Huang, and A. H.-L. Huang, "Intel Xeon Phi accelerated weather research and forecasting (WRF) Goddard microphysics scheme," *Geoscientific Model Development Discussions*, vol. 7, pp. 8941–8973, Dec. 2014.

[70] J. Mielikainen, B. Huang, and A. H.-L. Huang, "Optimizing Purdue-Lin Microphysics scheme for Intel Xeon Phi coprocessor," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 9*, Jan. 2016.

[71] J. Mielikainen, B. Huang, H. L. A. Huang, and M. D. Goldberg, "Improved GPU/CUDA based parallel weather and research forecast (wrf) single moment 5-class (wsm5) cloud microphysics," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 4, pp. 1256–1265, Aug. 2012.

[72] C. B. Moler, *Numerical computing with MATLAB*. SIAM, 2004.

[73] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, and F. X. Giraldo, “Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA,” *CoRR*, vol. abs/1511.01561, 2015.

[74] P. Neumann, P. Düben, P. Adamidis, P. Bauer, M. Brück, L. Kornblueh, D. Klocke, B. Stevens, N. Wedi, and J. Biercamp, “Assessing the scales in numerical weather and climate predictions: will exascale be the rescue?” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 377, no. 2142, p. 20180148, 2019.

[75] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with cuda,” *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008.

[76] OpenMP Architecture Review Board, “OpenMP application program interface version 4.0,” Jul. 2013. [Online]. Available: http://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf

[77] T. A. J. Ouermi, R. M. Kirby, and M. Berzins, “Numerical testing of a new positivity-preserving interpolation algorithm,” 2020. [Online]. Available: https://arxiv.org/abs/2009.08535

[78] ——, “HPPIS: A high-order positivity-preserving mapping software for structured meshes,” *Manuscript Submitted to TOMS*, 2022.

[79] T. A. Ouermi, A. Knoll, R. M. Kirby, and M. Berzins, “OpenMP 4 Fortran modernization of WSM6 for KNL,” in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ser. PEARC17. New York, NY, USA: ACM, 2017, pp. 12:1–12:8.

[80] T. Ouermi, R. Kirby, and M. Berzins, “Performance optimization strategies for WRF physics schemes used in weather modeling,” *Int. J. Netw. Comput.*, vol. 8, no. 2, pp. 301–327, 2018.

[81] T. Ouermi, A. Knoll, R. M. Kirby, and M. Berzins, “Optimization strategies for WRF single-moment 6-class microphysics scheme (wsm6) on Intel microarchitectures,” in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, 2017, pp. 146–152.

[82] T. A. Ouermi, R. M. Kirby, and M. Berzins, “Eno-based high-order data-bounded and constrained positivity-preserving interpolation,” *Numer. Algor.*, Jul. 2022.

[83] A. R. M. Piah, T. N. T. Goodman, and K. Unsworth, “Positivity-preserving scattered data interpolation,” in *Mathematics of Surfaces XI*, R. Martin, H. Bez, and M. Sabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 336–349.

[84] E. Price, J. Mielikainen, B. Huang, H. A. Huang, and T. Lee, “GPU acceleration experience with rrtmg long wave radiation model,” in *Proceedings SPIE 8895 High-Performance Computing in Remote Sensing III,*, B. Huang, A. J. Plaza, and Z. Wu, Eds., vol. 8895, 2013.

[85] A. Rogerson and E. Meiburg, “A numerical study of the convergence properties of eno schemes,” *J. of Sci. Comput.*, vol. 5, no. 2, pp. 151–167, 1990.

[86] L. D. Rotstayn, B. F. Ryan, and J. J. Katzfey, “A scheme for calculation of the liquid fraction in mixed-phase stratiform clouds in large-scale models,” *Monthly Weather Review*, vol. 128, no. 4, pp. 1070–1088, 04 2000.

[87] D. Sahasrabudhe, M. Berzins, and J. Schmidt, “Node failure resiliency for Uintah without checkpointing,” *Concurrency and Computation: Practice and Experience*, p. e5340, 2019.

[88] M. Sarfraz, “A C2 rational cubic spline alternative to the Nurbs,” *Comput. Graph.*, vol. 16, no. 1, pp. 69 – 77, 1992.

[89] J. W. Schmidt and W. Heß, “Positive interpolation with rational quadratic splines,” *Comput.*, vol. 38, no. 3, pp. 261–267, Sep. 1987.

[90] ——, “Positivity of cubic polynomials on intervals and positive spline interpolation,” *BIT Numerical Mathematics*, vol. 28, no. 2, pp. 340–352, Jun. 1988.

[91] M. Sekora and P. Colella, “Extremum-preserving limiters for muscl and ppm,” 2009.

[92] C. Shen, J.-M. Qiu, and A. Christlieb, “Adaptive mesh refinement based on high order finite difference weno scheme for multi-scale simulations,” *J. Comput. Phys.*, vol. 230, no. 10, pp. 3780–3802, 2011.

[93] C.-W. Shu, “Numerical experiments on the accuracy of eno and modified eno schemes,” *J. Sci. Comput.*, vol. 5, no. 2, pp. 127–149, 1990.

[94] ——, “High-order finite difference and finite volume weno schemes and discontinuous galerkin methods for cfd,” *Int. J. Comput. Fluid Dyn.*, vol. 17, no. 2, pp. 107–118, 2003.

[95] ——, “Essentially non-oscillatory and weighted essentially non-oscillatory schemes,” *Acta Numerica*, vol. 29, p. 701–762, 2020.

[96] C.-W. Shu, T. A. Zang, G. Erlebacher, D. Whitaker, and S. Osher, “High-order eno schemes applied to two- and three-dimensional compressible flow,” *Appl. Numer. Math.*, vol. 9, no. 1, pp. 45–71, 1992.

[97] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers, “A description of the advanced research wrf version 3. ncar technical note -475+str,” 2008.

[98] W. C. Skamarock and M. L. Weisman, “The Impact of Positive-Definite Moisture Transport on NWP Precipitation Forecasts,” *Monthly Weather Review*, vol. 137, no. 1, pp. 488–494, 2009.

[99] J. J. J. R. A. Sodani, *Intel Xeon Phi Processor High Performance Programming*, 2016.

[100] P. K. Subbareddy, A. Kartha, and G. V. Candler, “Scalar conservation and boundedness in simulations of compressible flow,” *J. Comput. Phys.*, vol. 348, pp. 827–846, 2017.

[101] E. Tadmor, “Entropy stability theory for difference approximations of nonlinear conservation laws and related time-dependent problems,” *Acta Numerica*, vol. 12, p. 451–512, 2003.

[102] E. Tadmor and J. Tanner, “Adaptive mollifiers for high resolution recovery of piecewise smooth data from its spectral information,” *Found. Comput. Math.*, vol. 2, no. 2, pp. 155–189, Jan. 2002.

[103] H. Tal-Ezer, “High degree polynomial interpolation in newton form,” *SIAM J. Sci. Statist. Comput.*, vol. 12, no. 3, pp. 648–667, 1991.

[104] G. Ulrich and L. T. Watson, “Positivity conditions for quartic polynomials,” *SIAM J. Sci. Comput.*, vol. 15, no. 3, pp. 528–544, 1994.

[105] C. Wang, X. Dong, and C.-W. Shu, “Parallel adaptive mesh refinement method based on weno finite difference scheme for the simulation of multi-dimensional detonation,” *J. Comput. Phys.*, vol. 298, pp. 161–175, 2015.

[106] P. R. Woodward, J. Jayaraj, and R. Barrett, “mppm, viewed as a co-design effort,” in *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, ser. Co-HPC ’14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 33–40.

[107] K. Yotov, T. Roeder, K. Pingali, J. Gunnels, and F. Gustavson, “An experimental comparison of cache-oblivious and cache-conscious programs,” in *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA ’07. New York, NY, USA: ACM, 2007, pp. 93–104.

[108] V. Zala, M. Kirby, and A. Narayan, “Structure-preserving function approximation via convex optimization,” *SIAM J. Sci. Comput.*, vol. 42, no. 5, pp. A3006–A3029, 2020.

[109] V. Zala, R. M. Kirby, and A. Narayan, “Structure-preserving nonlinear filtering for continuous and discontinuous galerkin spectral/hp element methods,” 2021.

[110] X. Zhang, “On positivity-preserving high order discontinuous galerkin schemes for compressible navier–stokes equations,” *J. Comput. Phys.*, vol. 328, pp. 301 – 343, 2017.

[111] X. Zhang and C.-W. Shu, “Maximum-principle-satisfying and positivity-preserving high-order schemes for conservation laws: survey and new developments,” in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 467, no. 2134. The Royal Society, 2011, pp. 2752–2776.

[112] ——, “Positivity-preserving high order finite difference weno schemes for compressible euler equations,” *J. Comput. Phys.*, vol. 231, no. 5, pp. 2245–2258, Mar. 2012.

[113] X. Zhang, Y. Xia, and C.-W. Shu, “Maximum-principle-satisfying and positivity-preserving high order discontinuous galerkin schemes for conservation laws on triangular meshes,” *J. Sci. Comput.*, vol. 50, no. 1, pp. 29–62, Jan. 2012.