Project Specification

Accelerating Performance of SPH Simulations Using DSL Methods with Exascale Techniques

Développement Sans Logique
Tom White, Scott Parker, Maciej Waszczuk,
Mikołaj Wąsacz, James Macer-Wright, Tom Divers

Department of Computer Science
University of Warwick

Supervisor: Dr Gihan Mudalige
Year of study: 2023/2024

Contents

1 Introduction **3**

2 Background **4**
2.1 Domain Selection 4
2.1.1 Monte Carlo 4
2.1.2 Multi-material 5
2.2 Smooth Particle Hydrodynamics 5
2.3 Prior Work 6
2.3.1 OPS [15] & OP2 [16] 6
2.3.2 SPHinXsys [9] 6
2.3.3 PySPH [11] 7
2.3.4 Nauticle [17] 7
2.4 Miniapps 7
2.4.1 SPH-EXA 7

3 Objectives **9**
3.1 Objective List 9
3.2 System Overview 10

4 Resources **12**
4.1 Miniapps 12
4.2 Libraries 12
4.3 Hardware 13
4.4 Software 14

5 Testing and Validation **15**
5.1 Unit Testing 15
5.2 Functional Testing 15
5.3 Integration Testing 15
5.4 Performance Testing 16

6 Project Management **17**
6.1 Methodology 17
6.2 Roles 17
6.3 Timeline 18

7 Risks **20**

8 Legal, Social and Ethical Issues **22**

Glossary

AST Abstract Syntax Tree.

CPU Central Processing Unit.

CUDA Compute Unified Device Architecture, an API allows software to use Nvidia GPUs for general purpose processing.

DSL Domain Specific Language.

IR An intermediate representation, a version of the program code that has been converted to a simpler form making it easier to optimise.

LLVM A compiler backend software that optimises and IR and converts it to an executable program.

LLVM IR The intermediate representation used by LLVM.

MPI Message Passing Interface, a tool for multi-node parallel computation.

Octree A data structure where each internal node has 8 children. Typically used to store points (e.g. particles) in a 3D space for faster search algorithms.

OpenMP Open Multi Processing, a tool for parallelising programs across multiple processing cores.

SPH Smoothed Particle Hydrodynamics.

Chapter 1

Introduction

Smooth Particle Hydrodynamics (SPH) is a mesh-free Lagrangian method used to obtain approximate solutions to fluid dynamics problems by simulating a large number of particles and their interactions [1]. This has been used extensively in scientific and industrial applications. These applications include simulations of dam breaking [2] and rigid bodies such as a ship hull impacting water [3]. SPH can also be easily extended to other problems such as gases or dust fluids [4], and thermal and matter diffusion. It is extensively used in the special effects and graphics industry, including video games, movies and TV advertising [5].

Due to the practical nature of the SPH problem, many software frameworks and libraries have been developed. In order to obtain meaningful results from SPH simulations, a very large number of particles must be considered. Additionally, the equations describing the particles' behaviour are fairly complex. Hence, large-scale simulations require massive amounts of computing power, far beyond the capabilities of a typical desktop PC. Because of this, high-performance computing nodes or clusters are used to perform such simulations. Such systems are characterised by high levels of parallelism, which requires the simulation code to be written in a way that can fully utilise these parallel systems.

Writing programs for high-performance systems is complex, as it requires in-depth knowledge of the target system in order to fully exploit its performance. This results in high-performance programs being expensive to develop and difficult to maintain. To address this issue, many libraries have been developed that aim to facilitate this process. Such libraries are often specific to a specialised problem domain and target platform. One of the techniques to making the code more readable and easier to write is using a Domain-Specific Language (DSL).

A DSL is a specialised programming language that includes convenient syntax for describing common operations in a specific problem domain. Since SPH simulations typically have similar structure and use a common class of operations, they are well-suited to be described using a specially-designed DSL. This is why we decided to use this technique to develop a framework that aids in creating SPH programs. This approach has another benefit of enabling the user to write platform-independent code, since the framework handles parallelism and applies problem and architecture specific optimisation techniques.

Chapter 2

Background

2.1 Domain Selection

Selecting an appropriate domain for a DSL project is important, as there is a balance to strike between how populated with research the areas are.

A domain with a large amount of research (e.g. Monte Carlo simulations) is likely to have a large amount of well-documented optimisations, with many example miniapps to look at to find common code between. This makes it easier to know what the generated code must look like. However, DSLs may already exist for the domain, resulting in the project not contributing greatly to the field of research.

A domain with very little research (such as Multi-material) is the opposite. It is difficult to find generalising optimisations that can be applied in the area without being an expert in that domain's specific computations, and very familiar with any fresh research surrounding it. There may also not be many, if any at all, miniapps that demonstrate the techniques used in the domain. Additionally, there may not be a widely agreed-upon architecture for the representation of the data structures for the domain.

This meant a number of possible domains were considered by the team and supervisor to ensure the project is well-realised, thought-out and that issues stemming from the domain selection do not come up later, hindering the scope of the project or costing significant development time.

The alternative domains that were considered, are described below, and reasons are given as to why the final chosen domain was favoured over them.

2.1.1 Monte Carlo

Monte Carlo simulations make use of statistical models to estimate mathematical functions and model complex systems. In a Monte Carlo Simulation, a system is modelled as a series of probability density functions (PDFs), the PDFs are repeatedly sampled from, and the results have some aggregate function applied to them to generate a statistic [6].

Monte Carlo problems are highly parallelisable, as there is no communication between threads until the aggregate function is performed at the end of computation. This makes them highly applicable to high performance computing, where they can be distributed across GPUs and compute nodes at a high scale.

However, due to the simplicity of this problem, there are many existing DSLs that have examined the Monte Carlo problem. These include Neb [7] which generates C++ code targeting GPUs and FPGAs from equations written in LaTeX, and Contessa [8] which generates code targeting FPGAs from a functional subset of C.

As a result, we decided that designing a DSL for a problem domain with a less saturated research field would be a more relevant project, and result in a greater contribution to the DSL space.

2.1.2 Multi-material

In many scenarios it would be useful to simulate involve more than a single material. This form of simulation is particularly challenging, with no standard procedure or best solutions agreed on in the little available literature, given this is a relatively new field.

This was considered as a potential direction for the project, though it was concluded that a more populated area is more feasible for this project. This project would be hard to find performance comparisons for due to a lack of available miniapps. Another consequence of this is that it would be far beyond the scope to find our own optimisations or create our own efficient code with no frame of reference.

2.2 Smooth Particle Hydrodynamics

SPH simulations (discussed extensively in [5]) are typically used to simulate the mechanics of a continuous medium. This is done by discretising the medium into a collection of Lagrangian particles. In order to simulate some field $f : \Omega \to \mathbb{R}$ over a domain $\Omega \subseteq \mathbb{R}^d$, we find the following for each particle i located at $\mathbf{r}_i \in \mathbb{R}^d$:

$$f(\mathbf{r}_i) = \int_\Omega f(\mathbf{r}) W(\mathbf{r}_i - \mathbf{r}, h) d\mathbf{r} \tag{2.1}$$

$W(\mathbf{r}_i - \mathbf{r}, h)$ is referred to as a *kernel function*, and governs how the values of $f(\mathbf{r})$ around $\mathbf{r}_i$ affect f_i. We assume that W is radially symmetric with respect to $\mathbf{r}_i$. $h \in \mathbb{R}$ is referred to as the *smoothing radius*, and allows us to adjust the shape of the kernel. A one-dimensional example of such a kernel is the Gaussian kernel, given by

$$W(x, h) = \frac{e^{-x^2/h^2}}{h\sqrt{\pi}} \tag{2.2}$$

We assume that the mass of each particle m_i is constant throughout the simulation; therefore, each of the particles has volume $V_i = m_i/\rho(\mathbf{r}_i)$, where $\rho : \Omega \to \mathbb{R}$ is the density of our medium. Therefore, in order to compute equation (2.1), we approximate it by summing over $N(i) := \{j : \|\mathbf{r}_j - \mathbf{r}_i\|_2 \leq \epsilon\}$, the set of all particles at most ϵ away from i [9]:

$$f(\mathbf{r}_i) \approx \sum_{j \in N(i)} V_j \cdot W(\mathbf{r}_i - \mathbf{r}_j, h) \cdot f(\mathbf{r}_j) = \sum_{j \in N(i)} \frac{m_j}{\rho_i} \cdot W(\mathbf{r}_i - \mathbf{r}_j, h) \cdot f(\mathbf{r}_j) \tag{2.3}$$

Note that $\rho(\cdot)$ is also a field over $\mathbb{R}^d$; therefore, we can also compute it using equation (2.3):

$$\rho(\mathbf{r}_i) \approx \sum_{j \in N(i)} \frac{m_j}{\rho(\mathbf{r}_j)} \cdot W(\mathbf{r}_i - \mathbf{r}_j, h) \cdot \rho(\mathbf{r}_j) = \sum_{j \in N(i)} m_j \cdot W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{2.4}$$

We can use this particle summation method to compute the gradient of f as

$$\begin{aligned} \nabla f(\mathbf{r}_i) &= \int_\Omega \nabla f(\mathbf{r}) W(\mathbf{r}_i - \mathbf{r}, h) dV = -\int_\Omega f(\mathbf{r}) \cdot \nabla W(\mathbf{r}_i - \mathbf{r}, h) dV \\ &\approx -\sum_{j \in N(i)} V_j \cdot \nabla_i W_{ij} \cdot f(\mathbf{r}_j) \end{aligned} \tag{2.5}$$

where $\nabla_i W_{ij} := \nabla_{\mathbf{r}_i} W(\mathbf{r}_i - \mathbf{r}_j, h)$ is the gradient of our smoothing function in the direction of $\mathbf{r}_i$. Similar derivations can be used to compute other derivatives of f.

Considering we have not specified the field we are trying to simulate, the framework above remains very general. Consequently, this framework can be used to simulate a wide variety of phenomena, in areas ranging from fluid dynamics to astrophysics [5].

Computing these particle-average sums gives rise to several challenging computational problems. One such problem stems from the fact that this framework is meshless. Because of this, computing $N(i)$ efficiently is a challenging and computationally expensive problem with a wide variety of existing algorithms [10, 11, 12]. This is especially true in a multi-node setting, where data locality is vital [13, 14]. We aim to simplify, abstract and partially automate the implementation of efficient and parallel algorithms for this task with our DSL.

2.3 Prior Work

SPH simulations have been an active research area ever since their introduction; consequently, prior efforts to coalesce their features into libraries and domain-specific languages. However, we believe that an active-library DSL (in a similar vein to OPS [15] and OP2 [16]) will fill a gap by providing new functionality and flexibility compared to existing solutions. It is also important to be familiar with features present in other related projects as to offer a viable product at the end, without missing features that experts in the field may find obvious and essential.

2.3.1 OPS [15] & OP2 [16]

OPS and OP2 are DSLs developed for specific problem domains that compile code written with their API to many different hardware platforms. This results in performance portability, near-optimal performance, and scaling on modern multi-core and many-core processor based systems. OPS targets algorithms over a structured mesh, which has a fixed topology of meshes, and OP2 targets algorithms over an unstructured mesh, which stores connectivity information to specify the mesh topology.

OP2 generates the target code by using Clang to generate an AST of the program, extracting key values from the kernel functions using this AST. These are then transformed as necessary, then used to populate templates of the target code developed for each backend.

A program in these DSLs is implemented via an access-execute pattern, where the user

1. Initialises the state of the data structure with OPS/OP2, by manually setting values or importing larger datasets with HDF5
2. Writes kernel functions for iterations and reductions
3. Declares the kernel functions with OPS/OP2 and specifying their data access patterns and other information about the functions

The DSL we develop will take heavy inspiration from the library design and architecture of OPS/OP2.

2.3.2 SPHinXsys [9]

SPHinXsys is an open-source library designed to simplify the process of writing SPH systems. It supports simulations of a large variety of physical objects and phenomena, including Newtonian and non-Newtonian fluids, rigid and elastic solids, diffusion-reaction processes, amongst others.[1] It is unlikely that we will be able to develop a DSL that accommodates all of these features in such a short amount of time; however, we do expect to be able to develop a solution that has several advantages over SPHinXsys.

[1] SPHinXsys's official documentation can be found at `www.sphinxsys.org`.

One of these advantages comes in the form of optimisations. By choosing to implement a DSL instead of a traditional library, we expect to be able to apply several optimisations to input programs *before* compile-time. Such optimisations are not possible for traditional libraries, and SPHinXsys does not make use of them.

To run simulations in parallel, SPHinXsys uses Intel's TBB C++ template library as a backend. In our DSL, we intend to include support for a variety of different backends, including OpenMP, CUDA and MPI (which is made significantly easier by choosing to write a DSL), thus allowing for a wider array of system architectures.

2.3.3 PySPH [11]

PySPH is an extensive Python framework for writing SPH simulations that targets OpenMP, MPI and CUDA. Architecturally, this library is very similar to the DSL that we intend to implement.

PySPH promises a lot of flexibility in the algorithms it uses, and allows the user to switch out some part of a simulation's implementation in a very modular way. PySPH claims performance to be "comparable to handwritten solvers implemented in FORTRAN", but their performance report within the main published paper claims to be around 50% slower than DualSPHysics v5.0, a handwritten C++ SPH solver. This is due to DualSPHysics' more rigid design, as it "does not allow us to replace the NNPS algorithm easily". It is mentioned that there is still a lot of performance to get out of the framework, though it is practically impossible to reach parity given the nature of Python, and the reliance on Cython's compiler to be optimal.

By using C++ we hope to get better performance compared to PySPH, while maintaining its flexibility of being able to implement a wide range of SPH problems.

2.3.4 Nauticle [17]

Nauticle is a YAML-based DSL for particle-based simulations. It allows users to specify a system of equations, and then uses the Arboria library to simulate them [18].

Nauticle has the advantage of being relatively easy for users without programming experience to learn. In order to accommodate for this, however, Nauticle sacrifices flexibility. It is also relatively limited in terms of performance and target architectures, and does not provide parallel implementations for OpenMP, CUDA, or MPI (we intend to target all three with our solution).

2.4 Miniapps

A reduced, proxy application or miniapp encapsulates the performance-intensive component of many scientific or research based workloads in a simpler version of the application [19]. This results in a more easily understandable and modifiable codebase that still represents the main performance characteristics of the workload. Existing miniapps implementing the SPH problem were investigated as part of the research for this project, to gain a greater understanding of the SPH algorithms and how they can be accelerated. This includes SPH-EXA [13], a miniapp targeting exascale computation with backend implementations using MPI, OpenMP and CUDA, as well as PBF-SPH [20], a simpler SPH miniapp with backends using OpenMP, OpenCL and SYCL.

2.4.1 SPH-EXA

SPH-EXA is a miniapp written by the HPC Group at the University of Basel, it is a SPH simulation written for C++20 which supports a number of different technologies. Specifically, MPI, OpenMP, CUDA and

HIP. The motivation behind the miniapp was to "extrapolate common basic features of SPH simulations and consolidate them in a fully optimised, Exascale-ready package" [13].

The package implements a number of well known test simulations, such as "sedov" (spherical blast wave) and "evrard" (spherical collapse) [21, 22, 23]. It also supports the use of custom initialisation files for these scenarios, or alternately simply specifying the side lengths of a cube of particles in the simulation. When running the simulation, the developer can control the number of iterations or give a real length of time for the simulation. Similarly, it supports outputting a dump of particle data and various properties at a controllable number of iterations (e.g. particle position, velocity, density). This data is aggregated in a HDF5 file, which provides a compact structured data storage format which is effective both in serial and parallel [24].

Similarly, SPH-EXA supports both running serially or in MPI, both with OpenMP controlling parallelism in each node. It also builds an executable for running the simulation on GPUs using CUDA. Experience testing the miniapp reveals issues with the parallel HDF5 writing, with severe I/O bottlenecks on iterations which dump particle data out to file.

SPH-EXA uses an octree internally as part of a specialised custom library called "Cornerstone" to distribute particle between nodes, as well to provide a data structure to efficiently represent the 3-dimensional simulation space at a low level and provide an efficient way to do nearest neighbour operations in the iteration. From the sequencing diagram of the iteration process included in the documentation finding nearest neighbour particles takes $\sim$10% of the total iteration time, and this is a critical operation to perform efficiently, especially distributed over multiple nodes, to keep the program performance high. We will consider using a similar octree-based model in our project.

Chapter 3

Objectives

In order to ensure our system fulfils its intended purpose, we have devised a formal list of requirements. This will allow us to track progress and measure the success of the project. The requirements have been categorised and prioritised, according to the MoSCoW method. In particular, "must" means that a requirement is essential for the project and will be considered first. "Should" denotes the requirements that are important but not strictly necessary. "Could" means that a requirement is of less importance and will be implemented provided that the time constraints allow it. We have also identified the objectives which are out of scope for this project, and therefore will not be implemented. Such objectives are labelled as "won't".

3.1 Objective List

1. Problem domain
   - 1.1 The system **must** facilitate writing code for common SPH problems.
   - 1.2 The system **won't** support other problems outside of the SPH domain, such as gravitational interaction or Discrete Vortex Method (DVM).
2. Input code
   - 2.1 The system **must** handle input code written in C++, potentially with some syntax extensions.
   - 2.2 The system **should** use existing C++ syntax whenever possible.
   - 2.3 The system **could** process written equations into kernel functions for the user.
   - 2.4 The system **won't** process input code written in other languages, such as Python.
3. API design
   - 3.1 The system **must** expose a low-level API, similar to OPS and OP2.
   - 3.2 The system **should** allow the user to implement some functions in a platform-specific way (e.g. if they want to integrate an existing SPH HPC library).
   - 3.3 The system **could** implement a high-level API, to further simplify developing SPH simulations (e.g. use operator overloading to allow the user to define parts of the program as equations).
4. Target platforms
   - 4.1 The system **should** support multiple target processor architectures.

4.2 The system **should** be able to generate code for OpenMP.

4.3 The system **should** be able to generate code for CUDA.

4.4 The system **could** be able to generate code for MPI.

4.5 The system **could** implement combinations of platforms, such as CUDA + OpenMP or CUDA + MPI.

5. Performance

5.1 The system **should** generate code that achieves performance no more than 10% slower than the reference miniapp implementations.

5.2 The system **could** implement more advanced optimisations to further improve performance.

5.3 The system **should** be able to compile input code in reasonable time.

3.2 System Overview

Figure 3.1 gives a broad overview of the proposed project architecture to meet the above objectives. Broadly, we take the C++ code written with our DSL, and parse it. Extract the AST and then generate suitable code for one of a number (or a combination) of backend hardware using a template engine. In the process, applying transformations or optimisations as necessary. This is then compiled by the usual C++ toolchain, which can apply further compiler optimisations. Optionally, if time permits, we want to also develop a separate parser which can take a more elementary equation based representation of the SPH simulation and convert that into a representation based in our DSL.

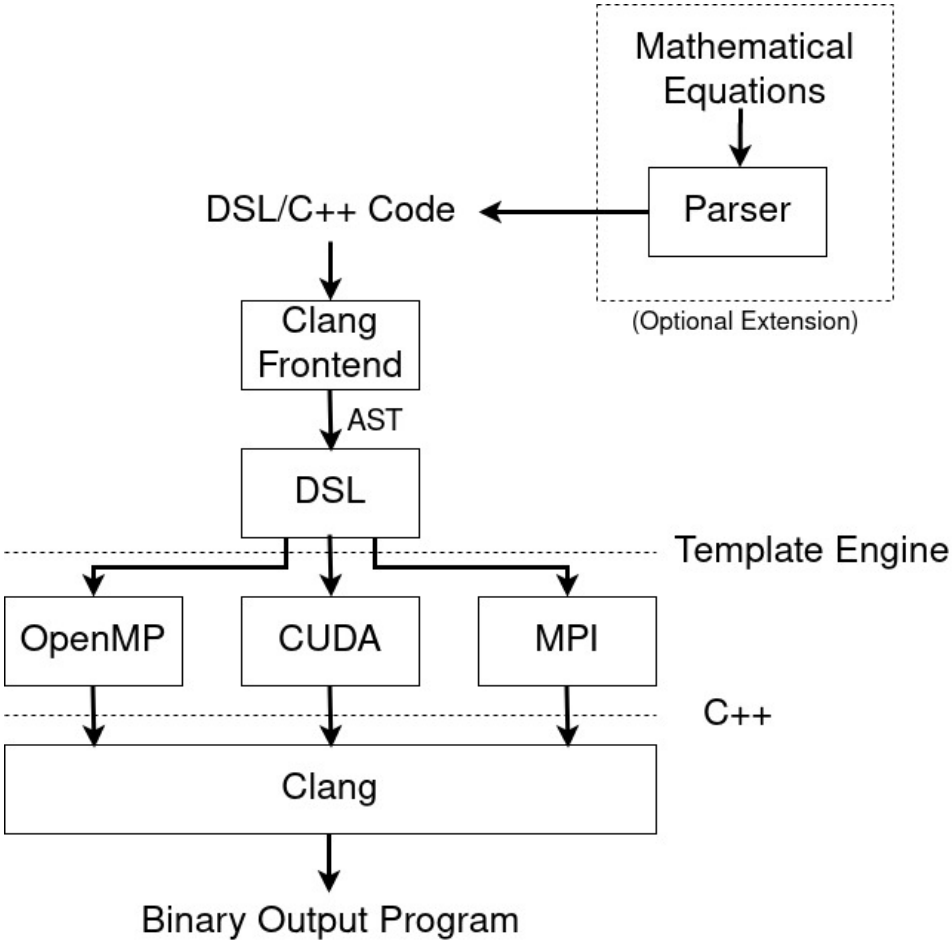

Figure 3.1: Overview of the proposed project architecture.

Chapter 4

Resources

Throughout the development of the project, we anticipate requiring several open source technologies and projects. These will vastly aid development, by saving time re-implementing common functions such as hardware compatibility or control, or by providing comparisons of performance and results for testing efficiency or validating program correctness. Furthermore, high-end multi-node multi-core computing systems are prohibitively expensive, so for practical purposes it is more feasible to utilise these systems.

4.1 Miniapps

Miniapp (or "Mini-App") is a miniature application which captures the inherent computational complexity of the problem domain whilst still keeping a small enough codebase to be manually analysed (or optimised). The term was coined fairly recently, and while there is no concrete definition of what a miniapp *is*, they also resemble closely benchmark applications developed previously [25].

We are planning on using two miniapps (SPH-EXA [13] and PBF-SPH [20]), which both implement SPH simulations for a number of different test scenarios, each targeting a number of different backends.

The use of miniapps in this project will be twofold: Firstly, as a benchmark. The miniapps are hand optimised to run on specific backend technologies such as PBH-SPH which runs on OpenMP, MPI, CUDA, OpenCL and SYCL. By comparing the run time of programs automatically generated by the project to these hand optimised programs we can get a benchmark of how much overhead our project adds to the expected performance a user would expect.

Secondly, we can use the miniapps to provide a simple verification check that the project is outputting the expected solutions to the simulation. The SPH-EXA miniapp includes a number of test simulations such as "sedov" [21] (spherical blast wave), "noh" (spherical implosion), "evrard" (gravitational collapse of an isothermal cloud), "turbulence" (turbulence in a box) and "kelvin-helmholtz" (Kelvin-Helmholtz instability in a slice). This will be in addition to other measures taken to ensure program verification and validation detailed in the testing chapter.

4.2 Libraries

As previously mentioned, the intention of this project is to develop a portable language which supports a number of high performance computing backends. These are often specialised or dedicated hardware architectures which require custom compiler wrappers or even open source libraries to provide compatibility to a higher level C++ API.

The backends we aspire to be able to support are the following. Note that due to time constraints, we may have to prioritise the initial and more common backends.

- **Serial** - Technically this is the absence of a specialised backend. However, it is important to note that the programs should be capable of being run in a serial environment. Perhaps taking advantage of other CPU optimisations such as SIMD vectorisation (i.e. AVX) though Intel Intrinsics library [26].
- **OpenMP** - This is the main library which supports shared memory parallel programing on CPUs in C++. It is intended that this be a primary focus for the project [27].
- **MPI** - Message Passing Interface is a standardised paradigm for multi-node parallel computing architectures. This involves orchestrating multiple independent CPUs through message passing. MPI is commonly implemented through an open source library, and we are planning on using OpenMPI [28].
- **CUDA** - CUDA is the primary parallel computing model developed by Nvidia for GPUs. CUDA is available in C++ through the Nvidia provided library and nvcc compiler [29].
- **SYCL & OpenCL** - A group of related technologies which are an open source alternative for developing parallel compute programs on GPUs [30].

In addition to the above general purpose libraries, the project is planned to involve a code generation functionality to create the optimised code for each backend. One method to implement this from a high level standpoint is to use Jinja [31] templates. These are short code snippets defined in a high level templating language called Jinja which are then placed in source code files before being compiled. Another option is to instead use the LLVM [32] parser and then directly manipulate the generated abstract syntax tree of the program.

Furthermore, the nature of particle simulations is that there will be a large amount of data to be dumped out from running simulations for visualisation or debugging purposes. One way of managing this in a parallel computing friendly way, as well as keeping file sizes small, is using a format such as HDF5 [24], which provides a structured and compact storage medium for the data. In addition, the Silo [33] library from LLNL[1] allows the stored data to be easily read by other programs, such as Python for analysis, or open source visualisation platforms like VisIt [34].

4.3 Hardware

To take advantage of the full potential of the massively parallel architectures we plan to use we will require the use of a much more powerful computing system than is available in personal computing hardware. We therefore plan to make use of two systems available at the University of Warwick:

- **DCS Batch Compute** - The Department of Computer Science's batch compute system allows for both CPU and GPU tests involving both single node and multi-node programs. The CPU partitions utilise Intel Xeon E5-2660 v3 with 5 nodes / 200 threads available, while the GPU partitions utilise Nvidia A10s.
  This system is also particularly useful for consistent benchmarking as they have fewer background processes running on them compared to traditional machines.
- **Scientific Computing RTP** - The University also provides access to more powerful compute clusters which are part of HPC Midlands [35].

[1] Lawrence Livermore National Laboratory

4.4 Software

In addition to the specialised hardware and libraries used to develop the project, we plan to use other software services to support project development.

- **GitHub** - The university provides an enterprise GitHub solution, which will be useful for version control and managing contributions between a large team working on multiple parts of the project simultaneously. It will also support the project management portion with GitHub Projects, providing a simple Kanban board implementation to keep track of important tasks and pending deadlines. Furthermore, GitHub Actions will allow automated testing to be run regularly to ensure correctness.
- **Visual Studio Code** - A lightweight IDE which supports C++ development as well as git version control. In addition, the remote functionality will enable development on target architecture as well.
- **Slurm** - A job scheduler used by the department to schedule jobs on the specific hardware. It will also allow deploying tasks and monitoring their status automatically via email, instead of having to stay at terminals continually during runtime.
- **GTest** [36] - A C++ unit testing framework developed by Google, which allows functions to be tested and unit tests to be automated through a CI tool such as GitHub Actions.

Chapter 5

Testing and Validation

Testing of the project, and validation of its output, will be performed continuously in parallel with the development process. We plan to test the project in 3 levels: firstly at the unit test level, ensuring functions are operating as intended given dummy input and output data; secondly at the app level, we want to automate running test simulations to ensure program output is within acceptable tolerances of the solutions created by other SPH codes; thirdly, in between the unit and app level we need to ensure that the intermediate state is consistent, such that a program stopped at any point and restarted could still compute the correct solution.

Automated testing will help ensure the application works throughout the development process, and will allow developers to focus their energy on development of other components and features.

5.1 Unit Testing

Unit testing is the primary responsibility of the developer for that component, and will likely require a series of tests written in a framework such as GTest [36] this will allow for aggregating tests over the entire project to be automatically run by CI/CD tools such as GitHub to test completeness as well as check for problems such as regressions later on in development.

5.2 Functional Testing

At the app level, we will set up automated bash scripts to run the application on known SPH simulations, such as a spherical blast wave simulation, which are supported in comparable miniapps. These can then be run on the variety of different hardware backends we aim to support. The results can then be compared to a known miniapp output to ensure that generated code returns the correct solutions, within an acceptable tolerance.

5.3 Integration Testing

In the integration level, we want to ensure that the intermediate state of the project is consistent, especially when writing out intermediate particle data to a file. The ideal situation is that a program which then runs a simulation starting from the data output to a file could come to the same answer. This also helps validate that every time-step the program is working as intended. Ensuring that the intermediate state is consistent also allows the development of a check-pointed restart system to be developed, which would

be beneficial for longer running tests where for example, compute resources enforce a strict time limit the program can then be restarted from the last output intermediate numbers.

5.4 Performance Testing

Finally, in addition to all the validation of the project's correctness, we should design a suite of tests to ensure that the project meets the minimum set of requirements set down in this specification. Specifically, we should take timings of the runs of the project as a whole to compare to the handwritten miniapps. Ideally, our performance overhead should be no more than 10%, and our compile time not an additional unreasonable burden on the developer.

Chapter 6

Project Management

6.1 Methodology

In order to manage the project, the team will utilise a Scrumban methodology [37]. This agile methodology combines the system of sprint cycles typical of a scrum approach with the use of a kanban board to keep track of tasks [38, 39].

The project will be split into week-long sprints. At the beginning of each sprint, a scrum meeting will take place (attended by all team members and the project supervisor) in order to specify what tasks need to be completed during the week, and who these tasks will be assigned to. The project supervisor's input will be used to guide this process. If necessary, the project team will meet separately afterwards in order to debrief and decide exactly what tasks need to be completed, and whose responsibility each task will be.

Project supervisor meetings will also serve as sprint retrospectives for the previous sprint. To this end, the successes and failures of the previous week will be discussed in these meetings. Any blocking tasks will also be identified (the supervisor may be able to give advice as to how progress can be made towards these), and the overall methodology will also be evaluated and modified as appropriate.

A kanban board will be used to record new tasks as they arise, and to keep track of the execution of individual objectives. Columns of the kanban board will be used to track the progression of each objective through the execution cycle, with special markers being used to indicate any tasks that have become blocked by another task. Swimlanes will be used to rank each task according to its priority. All team members will have access to the kanban board, and will ensure that it is consulted and updated frequently, in order to ensure that it accurately describes the current state of the project's execution.

6.2 Roles

Each member of the team is expected to be deeply involved in all aspects of the project, including developing software, communicating our results, and planning sprint cycles. However, the team felt it was appropriate to define the following roles, in order to delegate some more specific responsibilities:

- **Project Manager (*Tom White*):** Tom's role will be to direct the execution of the project. He will be ultimately responsible for delegating tasks between the other team members, and making any key decisions (particularly if there is significant disagreement amongst the rest of the team about a specific decision).

- **Academic Specialist (*Scott Parker*):** Scott's role will be to interface with academics and their research in order to ensure that the team has access to up-to-date knowledge of developments in the field. He will also take responsibility for planning meetings with the project supervisor (and other experts in this area) as appropriate.
- **Customer Specialist (*Maciej Waszczuk*):** Maciej's role will be to interface with the customer in order to ensure that the outputs of the project satisfy their expectations. He will also take responsibility for planning meetings with the customer (and other external stakeholders) as appropriate.
- **DevOps Specialist (*Mikołaj Wąsacz*):** Mikołaj's role will be to ensure that development best-practices are adhered to by the rest of the team, particularly concerning version control. He will also be responsible for setting up environments (writing makefiles, installing libraries, etc.).
- **Testing and Validation Specialist (*James Macer-Wright*):** James will be primarily responsible for designing the testing and validation systems that will be used throughout development. Ensuring that the simulations generated by our DSL are correct is critical, and having a specific team member aiming to ensure this specifically was deemed appropriate. He will also be responsible for evaluating and comparing the performance of our DSL simulations to ensure that they are within targets.
- **Documentation Specialist (*Tom Divers*):** Tom will be primarily responsible for the documentation and other artefacts produced to communicate the progression and results of the project. Specifically, this includes the specification,[1] the progress presentation, the final report and the final presentation.

The team has also identified the following external stakeholders that are expected to have a significant impact on the direction of the project:

- **Project Supervisor (*Dr Gihan Mudalige*):** Dr Mudalige is the principle academic responsible for advising and assessing this project. He has a wealth of experience in developing active-library DSLs for HPC applications (specifically OPS [15] and OP2 [16]), and first introduced the team to SPH simulations. He is also responsible for teaching Warwick's CS325 Compiler Design module (which all team members took in 2022), further evidencing his knowledge and experience in this area.
- **Customer (*Dr Richard Kirk*):** Dr Kirk is designated as the official customer of this project. In practice, he will also serve as a source of expertise for the team members. He also has experience in HPC (evidenced by him teaching CS402 High-Performance Computing), and can also be trusted for advice.

6.3 Timeline

In order to plan the work effectively, we have created a provisional timetable, which is shown as a Gantt chart in figure 6.1. This timetable will be used as a reference to track progress and to ensure that the project is finished on time. However, if complications arise during development, the schedule may be adjusted slightly.

[1] This just so happens to be the document you are reading currently.

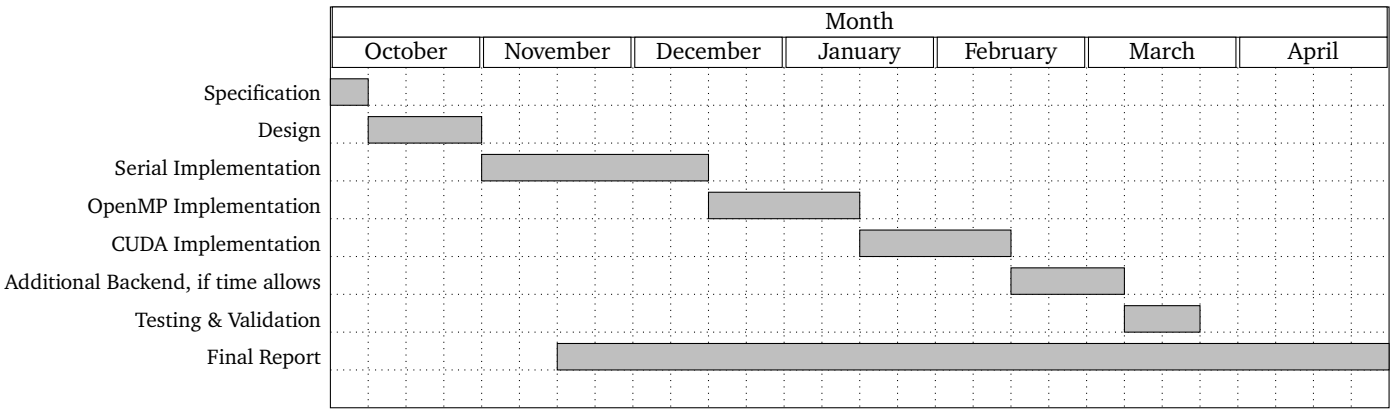

Figure 6.1: Project Gantt Chart.

Chapter 7

Risks

There are a number of risks to this project, as with all software development projects. A Risk Breakdown Structure (RBS) is a method defined in the PMBOK guide [40] of breaking down and categorising risks in a project to enumerate the risks. This ensures everyone is aware of what they are, so that they can best be avoided, mitigated, or otherwise dealt with. PRINCE2 [41] further describes these processes in-depth, which our team will incorporate to ensure the highest chance of success for the project.

After carefully analysing the project objectives and resources needed for their completion, the team devised a list of risks associated with the project using the RBS method. Next, each risk was evaluated in terms of their severity, occurrence, and detectability, according to the Failure Modes and Effects Analysis (FMEA). This analysis is invaluable in analysing individual systems' possible flaws and failures, and what risks they pose. Additionally, it provides a quantitative score of overall risk which can be compared to the risk after mitigations are applied, allowing for informed decision-making regarding prioritisation of countermeasures. The effects of our RBS and FMEA analyses are presented in table 7.1.

Potential Risk	Effect	Severity	Potential Causes	Occurence (frequency/ likeliness)	Detectability (lower is better)	Risk Score	Action Recommended
Scope Creep	Extra development time required to build unanticipated features	5	Poorly defined objectives or requirements.	2	3	30	Spend time early defining a clear specification
Acute Illness of Dev	The efficiency of the team is reduced	4	N/A	3	1	12	Reallocate the work to other developers whilst the team member is incapacitated
Release of new research	May make the project obsolete or overshadow its achievements	6	N/A	1	6	36	Determine a method to improve upon the new research, or figure out how to differentiate our project
Insufficient Communication with Customer	Product doesn't match customer's use case	5	Poor customer relations or low frequency of meetings	2	2	20	Schedule weekly meetings with the customer to ensure the project align's with the customer's requirements
Code changes / project work lost	Progress is lost, requiring extra development time to remedy	6	Poor collaboration workflow	2	2	24	Use remote version control (i.e. GitHub) and have a secondary environment to work on project if one is compromised
Inefficient Generated Code	Final product isn't able to be competitive with other existing solutions or directly written programs	6	Inexperienced or unprepared programmers	3	3	54	Read up on C++ best practices and ensure developers are familiar with the language. Consult more experienced C++ devs if uncertain
Team Members Leaving	Expertise of member lost and less overall developer time available	4	Incoherence of team vision / member leaves The University of Warwick	2	1	8	Reallocate work to other team members
Project design is no longer suitable for the goal	Product might require re-designing and extra development	5	Customer alters their requirements	4	2	40	Find an effective way to incoporate the customer's new requirements into the current project, if possible
Project is behind schedule	May not meet the project's deadline	7	Inefficiency in development / underestimating the amount of work for the project	3	4	84	Regularly reassess the project's progress against the original timeline

Table 7.1: Failure Modes and Effects Analysis

Chapter 8

Legal, Social and Ethical Issues

We have considered the legal, social and ethical implications of our project. We concluded that there are no immediate issues of concern in the specification of this project. However, as the project develops, we must be mindful and considerate of several factors.

Firstly, we will depend on a number of open source projects and libraries to develop the project; including but not limited to the miniapps for testing and verification, as well as the libraries to enable technologies to support various specialised hardware backends or other performance enhancing optimisations such as vectorisation. We must be careful to follow the licence terms of those projects and make it abundantly clear which sections of code were not written by ourselves.

Secondly, it is possible that the development of such a DSL could be used for projects and simulations which have unintended ethical consequences. Smoothed Particle Hydrodynamic simulations are used as tools in a large variety of industries, including complex particle interactions, nuclear engineering, and aeronautics. It is easy to imagine these fields may be used by, for example, military organisations. Hence, it is important to consider the moral implications of being associated with, or contributing to such fields.

Finally, we should also consider the environmental impact of running these simulations. Our project is designed to be used for long-running, highly computationally intensive scientific simulations of complex physical phenomena. It is highly unlikely that these codes will be run on normal "consumer grade" hardware, but instead on high performance compute clusters. While in some regards, the use of centralised hardware which is shared between many different computing projects at any one time cuts down on extraneous electrical waste, power consumption, and by extension environmental impact; on the other hand, we must be mindful that long-running inefficient programs would cause a more adverse environmental impact in terms of radiated heat and power consumption and as such we should ensure that our project produces as efficient code as possible.

References

[1] MB Liu and GR Liu. "Smoothed particle hydrodynamics (SPH): an overview and recent developments". In: *Archives of computational methods in engineering* 17 (2010), pp. 25–76.

[2] JJ Monaghan. "Simulating free surface flows with SPH". In: *Journal of computational physics* 110.2 (1994), pp. 399–406.

[3] Daniel John Veen. "A smoothed particle hydrodynamics study of ship bow slamming in ocean waves". PhD thesis. Curtin University, 2010.

[4] JJ Monaghan and A Kocharyan. "SPH simulation of multi-phase flow". In: *Computer Physics Communications* 87.1-2 (1995), pp. 225–235.

[5] JJ Monaghan. "Smoothed particle hydrodynamics". In: *Reports on Progress in Physics* 68.8 (July 2005), p. 1703. DOI: 10.1088/0034-4885/68/8/R01. URL: https://dx.doi.org/10.1088/0034-4885/68/8/R01.

[6] Robert L Harrison. "Introduction to monte carlo simulation". In: *AIP conference proceedings*. Vol. 1204. 1. American Institute of Physics. 2010, pp. 17–21.

[7] Ben Lindsey, Matthew Leslie, and Wayne Luk. "A domain specific language for accelerated multi-level Monte Carlo simulations". In: *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2016, pp. 99–106.

[8] David B Thomas and Wayne Luk. "A domain specific language for reconfigurable path-based monte carlo simulations". In: *2007 International Conference on Field-Programmable Technology*. IEEE. 2007, pp. 97–104.

[9] Chi Zhang et al. "SPHinXsys: An open-source multi-physics and multi-resolution library based on smoothed particle hydrodynamics". In: *Computer Physics Communications* 267 (2021), p. 108066. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108066. URL: https://www.sciencedirect.com/science/article/pii/S0010465521001788.

[10] Zheng Dequn et al. "Two Improved Nearest Neighbor Search Algorithms for SPH and Their Parallelization". In: *Computer, Informatics, Cybernetics and Applications*. 2012, pp. 1171–1179. DOI: 10.1007/978-94-007-1839-5_127. URL: https://link.springer.com/chapter/10.1007/978-94-007-1839-5_127.

[11] Prabhu Ramachandran et al. "PySPH: A Python-Based Framework for Smoothed Particle Hydrodynamics". In: *ACM Trans. Math. Softw.* 47.4 (Sept. 2021). ISSN: 0098-3500. DOI: 10.1145/3460773. URL: https://doi.org/10.1145/3460773.

[12] José Antonio Fernández-Fernández et al. "Fast Octree Neighborhood Search for SPH Simulations". In: *ACM Trans. Graph.* 41.6 (Nov. 2022). ISSN: 0730-0301. DOI: 10.1145/3550454.3555523. URL: https://dl.acm.org/doi/10.1145/3550454.3555523.

[13] Aurélien Cavelan et al. "A smoothed particle hydrodynamics mini-app for exascale". In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. 2020, pp. 1–11. DOI: 10.1145/3394277.3401855. URL: https://dl.acm.org/doi/10.1145/3394277.3401855.

[14] Sebastian Keller et al. “Cornerstone: Octree Construction Algorithms for Scalable Particle Simulations”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC ’23. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9798400701900. DOI: 10.1145/3592979.3593417. URL: https://dl.acm.org/doi/10.1145/3592979.3593417.

[15] István Z. Reguly et al. “The OPS Domain Specific Abstraction for Multi-block Structured Grid Computations”. In: *2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*. 2014, pp. 58–67. DOI: 10.1109/WOLFHPC.2014.7.

[16] G.R. Mudalige et al. “OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures”. In: *2012 Innovative Parallel Computing (InPar)*. 2012, pp. 1–12. DOI: 10.1109/InPar.2012.6339594.

[17] Balázs Havasi-Tóth. “Nauticle: A general-purpose particle-based simulation tool”. In: *Computer Physics Communications* 246 (2020), p. 106855. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2019.07.018. URL: https://www.sciencedirect.com/science/article/pii/S0010465519302322.

[18] Martin Robinson and Maria Bruna. “Particle-based and meshless methods with Aboria”. In: *SoftwareX* 6 (2017), pp. 172–178. ISSN: 2352-7110. DOI: 10.1016/j.softx.2017.07.002. URL: https://www.sciencedirect.com/science/article/pii/S2352711017300250.

[19] OE Bronson Messer et al. “MiniApps derived from production HPC applications using multiple programing models”. In: *The International Journal of High Performance Computing Applications* 32.4 (2018), pp. 582–593.

[20] Tom Lin. *PBF-SPH: A 3D real-time SPH mini-app implemented in OpenCL, OpenMP, SYCL, and SYCL2020*. 2021. URL: https://github.com/UoB-HPC/pbf-sph. Accessed October 2023.

[21] L. I. Sedov. *Similarity and dimensional methods in mechanics*. English. NASA/ADS, 1959.

[22] August E. Evrard. “Beyond N-body: 3D cosmological gas dynamics”. In: *Monthly Notices of the Royal Astronomical Society* 235.3 (Dec. 1988), pp. 911–934. ISSN: 0035-8711. DOI: 10.1093/mnras/235.3.911. URL: https://doi.org/10.1093/mnras/235.3.911.

[23] R. J. Thacker et al. “Smoothed Particle Hydrodynamics in cosmology: a comparative study of implementations”. In: *Monthly Notices of the Royal Astronomical Society* 319.2 (Dec. 2000), pp. 619–648. ISSN: 0035-8711. DOI: 10.1111/j.1365-8711.2000.03927.x. eprint: https://academic.oup.com/mnras/article-pdf/319/2/619/4125070/319-2-619.pdf. URL: https://doi.org/10.1111/j.1365-8711.2000.03927.x.

[24] *The HDF5® Library & File Format*. 2023. URL: https://www.hdfgroup.org/solutions/hdf5/.

[25] OE Bronson Messer et al. “MiniApps derived from production HPC applications using multiple programing models”. In: *The International Journal of High Performance Computing Applications* 32.4 (2018), pp. 582–593. DOI: 10.1177/1094342016668241. URL: https://journals.sagepub.com/doi/10.1177/1094342016668241.

[26] *Intel® Intrinsics Guide*. 2023. URL: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

[27] *OpenMP*. 2023. URL: https://www.openmp.org/.

[28] *Open MPI: Open Source High Performance Computing*. 2023. URL: https://www.open-mpi.org/.

[29] *NVIDIA CUDA Compiler Driver NVCC*. 2023. URL: https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html.

[30] *SYCL Overview - The Khronos Group Inc*. 2023. URL: https://www.khronos.org/sycl/.

[31] *Jinja*. 2023. URL: https://jinja.palletsprojects.com/en/3.1.x/.

[32] *The LLVM Compiler Infrastructure Project*. 2023. URL: https://llvm.org/.

[33] *Silo | File-based scientific data exchange and software interoperability*. 2023. URL: http://software.llnl.gov/Silo/.

[34] *VisIt Home*. 2023. URL: https://visit-dav.github.io/visit-website/.

[35] *University of Warwick Scientific Computing RTP*. 2023. URL: https://warwick.ac.uk/research/rtp/sc/. Accessed October 2023.

[36] *GoogleTest - Google Testing and Mocking Framework*. 2023. URL: https://github.com/google/googletest.

[37] Corey Ladas. *Scrumban - Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi lean series. Modus Cooperandi Press, 2009. ISBN: 9780578002149. URL: https://books.google.co.uk/books?id=SQFdAgAAQBAJ.

[38] Scrum.org. *What is Scrum?* 2023. URL: https://www.scrum.org/resources/what-scrum-module. Accessed October 2023.

[39] Agile Alliance. *What is Kanban?* 2023. URL: https://www.agilealliance.org/glossary/kanban/. Accessed October 2023.

[40] PMI, ed. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. 5th ed. Newtown Square, PA: Project Management Institute, 2013. ISBN: 978-1-935589-67-9.

[41] David Hinde. *PRINCE2 study guide*. English. Second – 2017 update. Indianapolis: Sybex, a Wiley brand, 2018. ISBN: 9781119420859;9781119549185;1119549183;1119420857;