Communication Avoiding 2D Stencil Implementations over PaRSEC Task-Based Runtime

Yu Pei*, Qinglei Cao*, George Bosilca*, Piotr Luszczek*, Victor Eijkhout†, and Jack Dongarra*
*Innovative Computing Laboratory, University of Tennessee, Knoxville, U.S.A.
†Texas Advanced Computing Center, University of Texas, Austin, U.S.A

***Abstract*—Stencil computation or general sparse matrix-vector product (SpMV) are key components in many algorithms like geometric multigrid or Krylov solvers. But their low arithmetic intensity means that memory bandwidth and network latency will be the performance limiting factors. The current architectural trend favors computations over bandwidth, worsening the already unfavorable imbalance. Previous work approached stencil kernel optimization either by improving memory bandwidth usage or by providing a Communication Avoiding (CA) scheme to minimize network latency in repeated sparse vector multiplication by replicating remote work in order to delay communications on the critical path.**

Focusing on minimizing communication bottleneck in distributed stencil computation, in this study we combine a CA scheme with the computation and communication overlapping that is inherent in a dataflow task-based runtime system such as PaRSEC to demonstrate their combined benefits. We implemented the 2D five point stencil (Jacobi iteration) in PETSc, and over PaRSEC in two flavors, full communications (base-PaRSEC) and CA-PaRSEC which operate directly on a 2D compute grid. Our results running on two clusters, NaCL and Stampede2 indicate that we can achieve 2X speedup over the standard SpMV solution implemented in PETSc, and in certain cases when kernel execution is not dominating the execution time, the CA-PaRSEC version achieved up to 57% and 33% speedup over base-PaRSEC implementation on NaCL and Stampede2 respectively.

***Index Terms*—2D stencil; communication avoiding; parallel programming models**

I. Introduction

Stencil computations are a common operator in a variety of scientific and engineering simulations based on partial differential equations (PDE), and they constitute a key component of many canonical algorithms from stationary iterative methods involving sparse linear algebra operations, for example Jacobi iteration [1], as well as non-stationary and projection methods employing geometric multigrid [2], [3] and Krylov solvers [4, pp. 241-313]. They are routinely used to solve problems that arise from the discretization of PDE [5]. Stencil codes can be characterized as having high regularity in terms of the data structures and the data dependency pattern. However, they also exhibit low arithmetic intensity and, as a consequence, the available memory bandwidth required for data movement is the limiting factor to their performance. To exacerbate these issues, the recent trends in the hardware architecture design have been skewed towards ever-increasing number of cores, widening data parallelism, heterogeneous accelerators, and a decreasing amount of per-core memory bandwidth [6]. The prior work on optimizing the stencil computations has mostly focused on techniques to improve the kernel performance within a particular domain such as cache oblivious algorithms, time skewing, wave-front optimizations, and overlapped tiling [7]. On modern systems, these algorithmic classes must be recast to overcome the geometrically growing gap between processor speed and memory/network parameters, in particular, CPU/GPU speeds have been improving at 59% per year while the main memory bandwidth at only 23%, and the main memory latency decreased at a mere 5.5% [8]. Given the widening gap between computation speed and network bandwidth, a systematic study of performance of stencils on the distributed memory machines is still relevant. Especially the optimization of communication is lacking.

In the recent years, a number of runtime systems and new programming models have been developed to facilitate application development by separating the domain science and the tuning of the performance, leveraging the respective strengths of domain scientist and runtimes. Some examples include: Legion [9], UPC++[10], StarPU [11], PaRSEC [12], and Charm++ [13]. In essence, the runtime system developers can optimize performance over the massively parallel and heterogeneous computing system, while the domain scientists express the algorithms as a Directly Acyclic Graph (DAG) of tasks. The hope is that the runtime system will be able to schedule the tasks on available resources, manage data transfers and be able to overlap the two efficiently.

In this study we adopted PaRSEC to abstract away the MPI communication across nodes, and experimented with communication-avoiding (CA) techniques to further reduce the communication overhead that is the limiting factor in stencil computation. We use the 2D five point stencil as the test case, and compared the performance of three implementations: PETSc, base-PaRSEC and CA-PaRSEC. We investigated extensively the interplay between memory bandwidth, computation speed and network latency/bandwidth on performance. We demonstrated that under some reasonable assumptions on workload and system configurations, we can achieve up to 57% and 33% improvements on the two machines we tested on when we add communication avoiding scheme into PaRSEC runtime.

In section II we cover the related works on stencil/sparse vector multiply and on runtime systems. Then in section III we briefly described the background of the problem, and the background on PaRSEC runtime and communication avoiding technique. In section IV we introduce the three implementa-

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00127

tions we used for evaluation. Sections V and VI describe our experiment setup and performance results and we conclude in section VII.

II. Related Work

Stencil research has mostly focused on optimizing the kernels [14] or domain specific systems that can generate efficient kernels automatically [6] and generating code that can utilize GPUs efficiently [15]. In [7] the authors not only optimized the kernel, but also implemented the communication avoiding technique directly within their compiler framework. Here instead of combining everything within one compiler system, we investigate delegating the internode communication to runtime system instead to combine communication hiding and communication avoiding at the runtime system level. Communication avoiding method (or s-steps method) itself is an old concept as stated in [16] and many Krylov solvers have been build with this idea [17]. The numerical properties of such approaches are out of the scope of this paper and we mainly focus on the feasibility and benefits of having CA ability implemented within a runtime infrastructure. Applications of communication avoiding technique to numerical linear algebra algorithms [18] [19] have also been studied and performance improvement demonstrated. In particular authors in [19] also studied the interaction between communication computation overlap with communication avoiding technique programmed with Unified Parallel C (UPC), a partitioned global address space (PGAS) model.

In recent years, the increase complexity of programming parallel and distributed machines has sparked the development of many programming models and runtime systems that can utilize the machine more efficiently. Among them PaRSEC [12], Legion [9], StarPU [11] and Charm++ [13] are actively being developed. In a task-based programming environment, a vast amount of parallelism is exposed through expressing the algorithm as a set of successive, fine-grain tasks. The runtime system is then responsible for scheduling these tasks while satisfying the data dependencies between them. Similar to the compiler approach, it also allows the decoupling of algorithm specification and the underlying machine optimization by the runtime system.

In this study, we adopt the runtime approach and study extensively the benefits of runtime to provide better communication computation overlap, and the opportunity for further improvement via communication avoiding scheme for stencil computation and for SpMV in general [20] [21]. To the best of our knowledge this is the first time the combined approach is being done on a distributed system. Our goal is to demonstrate the feasibility of such a software infrastructure for a broad range of numerical algorithms.

III. Background

A. Stencil Problem Description

Scientific simulations in diverse areas such as diffusion, electromagnetics, and fluid dynamics use PDE solvers as the main computational component. These applications commonly employ discretization schemes such as finite-difference or finite-element techniques. During the solve, they sweep over a spatial grid, performing computations involving nearest-neighbor grid points. Such compute patterns are called stencils. In these operations, each of the regular grid points is updated with weighted contributions from a small subset of neighboring points in both time and space. The weights represent the coefficients of the PDE discretization for that data element. Depending on the solver, these coefficients may be the same across the entire grid or differ at each grid point. The former is a constant-coefficient stencil while the latter – a variable-coefficient stencil. The range of solvers that often employ stencil operations includes simple Jacobi iterations [1] to complex multigrid [3] and adaptive mesh refinement (AMR) methods [22].

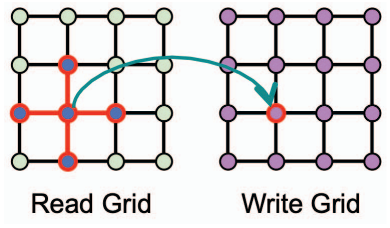

Fig. 1. Common illustration of the Jacobi update scheme [22].

Stencils can operate on different dimensions, having different iterations and coefficient types. In this work we use Jacobi iteration to solve the Laplace's equation, which means that we will have one read grid $X^{\ell-1}$ and one write grid X^{ℓ}, and the update will be in the form of:

$$\begin{aligned} x_{i,j}^{\ell} =& w_{0,0} \cdot x_{i,j}^{\ell-1} \\ &+ w_{0,-1} \cdot x_{i,j-1}^{\ell-1} + w_{0,1} \cdot x_{i,j+1}^{\ell-1} \\ &+ w_{-1,0} \cdot x_{i-1,j}^{\ell-1} + w_{1,0} \cdot x_{i+1,j}^{\ell-1} \end{aligned} \tag{1}$$

We used the more general form of weights which will give us the consistent FLOP/s count of $9n^2$ for all implementations (5 multiplications and 4 additions). A diagram of the Jacobi iteration scheme is shown in Figure 1.

B. Distributed Memory Dataflow with PaRSEC

PaRSEC [12] is a task-based runtime for distributed heterogeneous architectures and is capable of dynamically unfolding a description of a graph of tasks on a set of resources and satisfying all data dependencies by efficiently shepherding data between memory spaces (between nodes but also between different memories on different devices) and scheduling tasks across heterogeneous resources.

Domain-specific languages (DSLs) in PaRSEC help domain experts to focus only on their domain science by masking required computer science knowledge. The Parameterized Task Graph (PTG) [23] DSL uses a concise, parameterized, task-graph description known as Job Data Flow (JDF) to represent the dependencies between tasks. To enhance the productivity of the application developers, PaRSEC implicitly infers all communications from the expression of the tasks, supporting one-to-many and many-to-many types of communications. From a performance standpoint, algorithms described in PTG have been shown capable of delivering a significant percentage of the hardware peak performance on many hybrid distributed machines. Other DSLs, such as Dynamic Task Discovery (DTD) [24], are less science-domain oriented and provide alternative programming models to satisfy more generic needs by delivering an API that allows for sequential task insertion into the runtime.

C. Communication Avoiding Algorithms

With the increasingly widening gap between computation and communication, modern algorithms should try to minimize communication both within a local memory hierarchy and between processors. And this is especially true for SpMV, stencil operations that are memory system and network bound. The key idea in Demmel et al [16] is to perform some redundant work but can relief the bottleneck on communication latency. Two new algorithms were introduced, PA1 (depicted in figure 2) and PA2 as they described in the paper, where PA1 is the naive version while PA2 will minimize the redundant work but might limit the amount of overlap between computation and communication. Our implementation follows the PA1 algorithm.

As an example shown in figure 2, ghost region of *3*-layers are used to store remote data. This allows the local grid to perform Jacobi updates upto three time steps for the local data (white points) with replication of work from remote points (outer red points used to update inner red points). By performing redundant work, we reduce the frequency of communication thus the cost of network latency.

PaRSEC runtime system by design allows computation and communication overlap, by incorporating the communication avoiding scheme into the task-based implementation of stencil operations, we believe such an infrastructure can further improve its performance.

IV. Implementation

A. Standard Implementation with PETSc

PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations [26], [27], [28]. It provides many of the mechanisms needed within parallel application codes, such as simple parallel matrix and vector assembly routines that allow the overlap of communication and computation. In addition, PETSc includes support for parallel distributed arrays useful for finite difference methods.

Implementing Jacobi iteration in PETSc, we simply expand the 2D compute grid points into 1D solution vector, and the corresponding 5 points stencil update expresses as a sparse matrix. PETSc by default will partition the sparse matrix by rows with each process having a block of matrix rows. To perform the updates, we have two solution vectors that will swap within the for loop up to a specified iteration count.

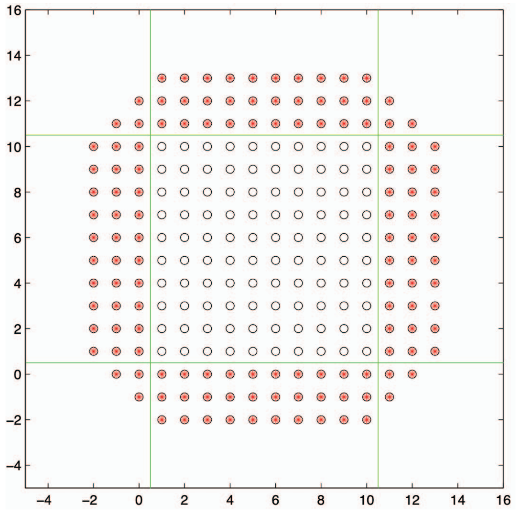

Fig. 2. The 2D five-point stencil operation using PA1 algorithm on a 10-by-10 grid, having a step size of 3 as illustrated in the original report [25]. For a single processor with the projected view. Red asterisks indicate remote values that need to be communicated.

Since PETSc is a mature and widely used package, the result will serve as the baseline for our PaRSEC performance.

B. Task-based Implementation in PaRSEC

Task-based runtime system offers a unified view of the underlying hardware and let the developer focus on the algorithm, described as Directed Acyclic Graph (DAG) of tasks. The runtime system will then manage all data transfers and synchronizations between computing devices and the scheduling of tasks among available computing resources. Hence these frameworks allow for the separation of major concerns in

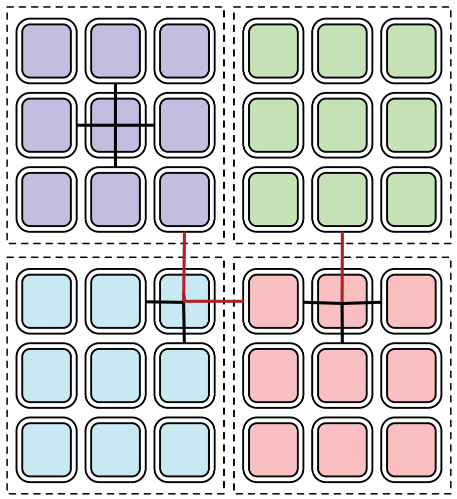

Fig. 3. Diagram of the base version PaRSEC implementation. Three possible task locations and their data dependencies are shown. Black line indicates within node data copy while red line indicates remote communication.

HPC: design of the algorithm, creating a data distribution and developing computational kernels [29].

1) Base PaRSEC Version: The first version follows the formulation of the Jacobi iteration, with data partitioned into 2D blocks over the 2D computation grids. Then the data on each node are divided into tiles that each task will operate on. By providing this extra level of decomposition, only the tasks that have neighbor tiles on a remote node will incur communication, while the inner tasks can still be processed with the remaining workers. Each tile will have an extra ghost region used for data exchange between tasks.

Figure 3 provides the diagram that depicts the implementation of the base stencil. The dashed line outlines the node boundary, the 2D blocked data distribution ensures that the surface to volume ratio is minimized and we have minimal remote communications. Each tile will have the same size and also each will have an extra ghost region for copying neighbor tiles' value. Since we are doing a 5-point stencil, the figure indicates that we will have three possible dependencies cases, for the interior tasks, all the neighbors are local to the task and we can simply copy the memory into the ghost region. For the tiles on the boundaries or corners, one or two remote data transfers will be needed. The computation kernel itself is very straightforward as we simply loop over the elements within a tile and apply the updates.

2) Communication Avoiding PaRSEC Version: The second version we adopted the communication avoiding scheme where we trade more computation for less frequent communication. Figure 4 has the overall structure very similar to the base version, and the interior tasks have the same task dependency as in the base version. But for boundary tiles, in addition to the four neighbors, we need to buffer additional data from the four corner neighbors due to the additional

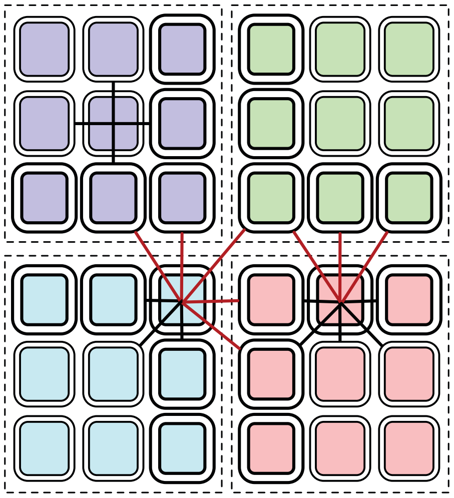

Fig. 4. Diagram of the communication avoiding version PaRSEC implementation. Three possible task locations and their data dependencies are shown. Black line indicates within node data copy while red line indicates remote communication. The boundary tiles will have a bigger ghost region to accommodate the extra layers of remote data.

steps of remote computation that we need to replicate locally. Since we still don't have the support at the runtime level, we implemented the logic directly as a proof of concept and it is a problem specific solution. Conditions are provided to test whether the task is operating on a boundary tile, and whether we need to communicate at this iteration. And we will have the corresponding logic in the body of the task to decide on the data we need to copy in and out, and which kernel we should call. Similar to the base version, the tiles that have all its neighbors being local to the node will have one layer ghost region for data exchange since they don't need remote communication. But the boundary tiles will have ghost region of *steps*-layers as specified for the extra amount of data exchanged, as a result, this version will use slightly more memory.

V. Experiment Setup and Evaluation

To evaluate the benefits of implementing stencil operations with a runtime system and additionally the benefits of incorporating communication avoiding scheme, we consider the following properties of the problem and the characteristics of the machines:

1) Number of arithmetic operations and memory accesses per task;
2) The maximum achievable network bandwidth of the cluster and the memory bandwidth of a compute node;
3) Number of floating-point numbers communicated per processor, and the number of messages sent per processor.

Since we formulate the problem in the more generic version which performs 9 floating point operations per update and need to transfer 16 to 24 Bytes (read and write of double floating point numbers) of data depending on the size of tiles, we will use the range of 0.37 to 0.56 as our arithmetic intensity. To measure the peak network bandwidth performance, we used the NetPIPE benchmark [30] and for memory bandwidth performance, we used the STREAM benchmark [31].

As mentioned here in before, there are three versions of implementation, one in PETSc and two in PaRSEC, with normal communication pattern and CA scheme respectively. We first compare the strong scaling performance of the three versions using PETSc as the baseline in order to have a better understanding of PaRSEC versions' performance. Then we move on to adjust the step sizes and tune the execution time of the kernel (simulate memory bandwidth utilization rate) to investigate the interplay between memory bandwidth and network communication on the overall performance. As the computer architectures continue to evolve, our result should provide a guidance for future performance improvements that can be expected on stencil operations.

VI. Experiment Results

The experiments are run on two systems. First is an in-house cluster called NaCL that has the total of 64 nodes, each with two Intel Xeon X5660 (Westmere) CPUs, with a total of 12 cores spread across two sockets and 23 GB of memory

TABLE I
STREAM BENCHMARK RESULTS (MB/S) FOR NACL AND STAMPEDE2.

System	Scale	COPY	SCALE	ADD	TRIAD
NaCL	1-core	9814.2	10080.3	10289.3	10271.6
NaCL	1-node	40091.3	26335.8	28992.0	28547.2
Stampede2	1-core	10632.6	10772.0	13427.1	13440.0
Stampede2	1-node	176701.1	178718.7	192560.3	193216.3

per node. The network switch and network cards are Infiniband QDR with a peak network rate of 32 Gb/s. The second system is Stampede2 system located at TACC: each node is equipped with two Intel Xeon Platinum 8160 (Skylake) CPUs with a total of 48 cores across two sockets, and 192GB of on-node memory. The interconnect is a 100 Gb/sec Intel Omni-Path network.

We used PaRSEC master branch from commit faf0872052, and PETSc release version 3.12. On NaCL, we compiled with gcc 8.3.0 and using Intel MPI 2019.3.199. On Stampede2, we compiled with Intel compiler 18.0.2 and MVAPICH2 version 2.3.1. PETSc was compiled with all the optimization enabled and using 64-bit integers. PETSc runs had one MPI process per core. For PaRSEC runs, we configured to have one process per node, with one thread dedicated for communication while the remaining ones for computation. The nodes during runs were arranged into square compute grid and the data tiles were allocated in a 2D block fashion to exploit the surface-to-volume ratio effect.

A. *Network and Memory bandwidth Benchmark*

STREAM benchmark is run on both systems utilizing all the cores on a compute node since, as the results show, a single core cannot saturate the memory interface. The results are shown in Table I. The different STREAM modes vary in their arithmetic intensity: bytes transferred per FLOP computed. For simplicity, in the following we use the results from COPY operation as our achieved memory bandwidth.

The achieved bandwidth NaCL and Stampede2 were 39.1 GB/s and 172.5 GB/s, respectively. Our estimated arithmetic intensity is between 0.37 to 0.56 depending on data availability in cache. We expect the effective peak performance between 14.5 to 21.9 GFLOP/s and 63.8 to 96.6 GFLOP/s for our memory-bound stencil kernels under the assumptions of the roofline model [32].

We test the network interconnect on both systems with the NetPIPE benchmark and obtain the following performance results (also plotted in Figure 5). The effective peak network bandwidth on NaCL is about 27 Gb/s while on Stampede2 we can achieve up to 86 Gb/s. Given the size of our stencil tiles, we will likely not be able to reach that peak bandwidth shown in Figure 5. The latency of the network is around 1 microseconds.

B. *Tuning of Tile Size for PaRSEC Performance*

Next, we measure the actual performance results of the base implementation on top of PaRSEC that runs on a single node (no network communication) with different tile sizes across all available cores. The results allow us to select a reasonable tile size for local computation. They also tell us the gap between the performance of the native kernel and the peak memory bandwidth performance to provide us with a reference point for distributed runs.

We can see in Figure 6 there is a certain range of tile sizes that allows us to obtain reasonable performance levels. For the NaCL system, the tile sizes of 200 to 300 will result in 11 GFLOP/s while on Stampede2 the tile sizes 400 to 2000 will yield close to 43.5 GFLOP/s. Given the fact that we did not optimize the kernel, the obtained result is acceptable for the circumstances but is still not close to the peak memory bandwidth level indicated in the previous section. Therefore, in the following experiments, we will run PaRSEC versions with the tile sizes in the optimal range obtained from the local-only runs.

C. *Comparing Strong Scaling Performance*

Figure 7 shows the strong scaling speed up of the three implementations when using optimal single node performance as baseline. We can see that all three maintain good scalability levels, and PaRSEC versions can achieve twice the performance of PETSc. This performance advantage can be partly explained by the SpMV formulation used by PETSc, since instead of having the weight matrix be represented with only 5 numbers, the update will involve both sparse matrix indices and the corresponding values. This, at the very least, doubles the number of memory loads (64-bit integers) that are needed for the same amount of floating point operations (64-bit floating-point.) Finally, we notice that the two PaRSEC versions are almost indistinguishable from each other, indicating that the communication avoiding approach is not very helpful for 2D 5-point stencils on the tested machines as long as the kernel is bound by the local memory bandwidth instead of the network bandwidth.

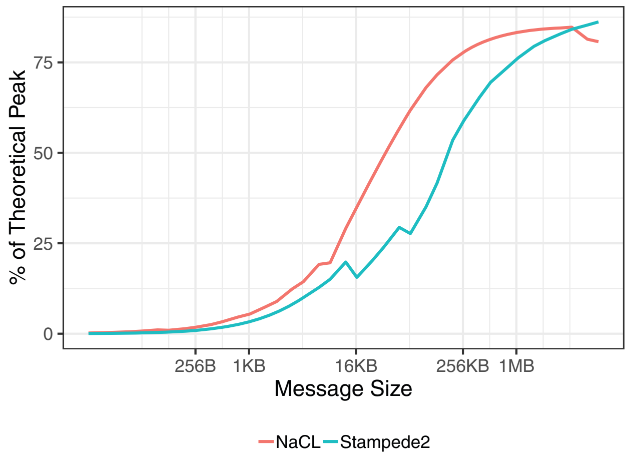

Fig. 5. Network Performance from NetPIPE on NaCL and Stampede2 with theoretical peak of 32Gb/s and 100 Gb/s, respectively.

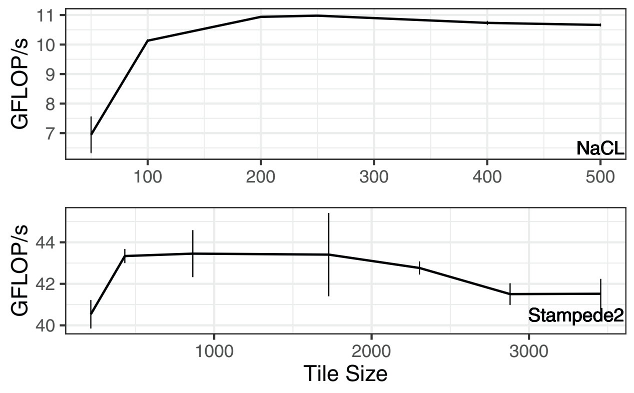

Fig. 6. Shared memory PaRSEC base version performance for a given tile size, top) NaCL with problem size 20K, bottom) Stampede2 with problem size 27K.

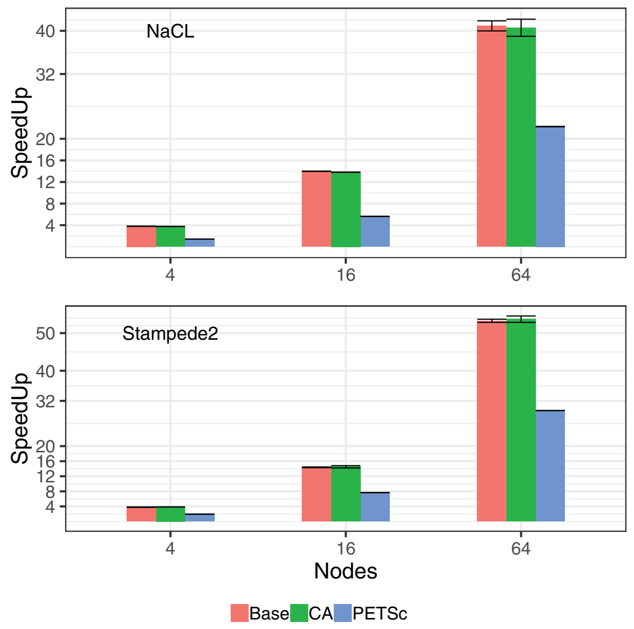

Fig. 7. Strong scaling speed up over single node baseline PaRSEC. top) NaCL result with problem size 23k, tile size 288; bottom) Stampede2 result with problem size 55k, tile size 864, running for 100 iterations. Steps size of 15 is used for CA version.

D. *Tuning of Kernel Time and Performance Impact of Communication Avoiding Scheme*

To further investigate the potential benefits of communication avoiding schemes on a distributed problem, we test in the case where the memory system is much faster or our computation kernel has been optimized to utilized the memory bandwidth better (a local communication avoiding scheme that reduces slow memory accesses for example). To simulate this, we set a ratio parameter, so that only $(ratio \times mb) \times (ratio \times nb)$ portion of the tile will get updated, which effectively reduce the memory access thus

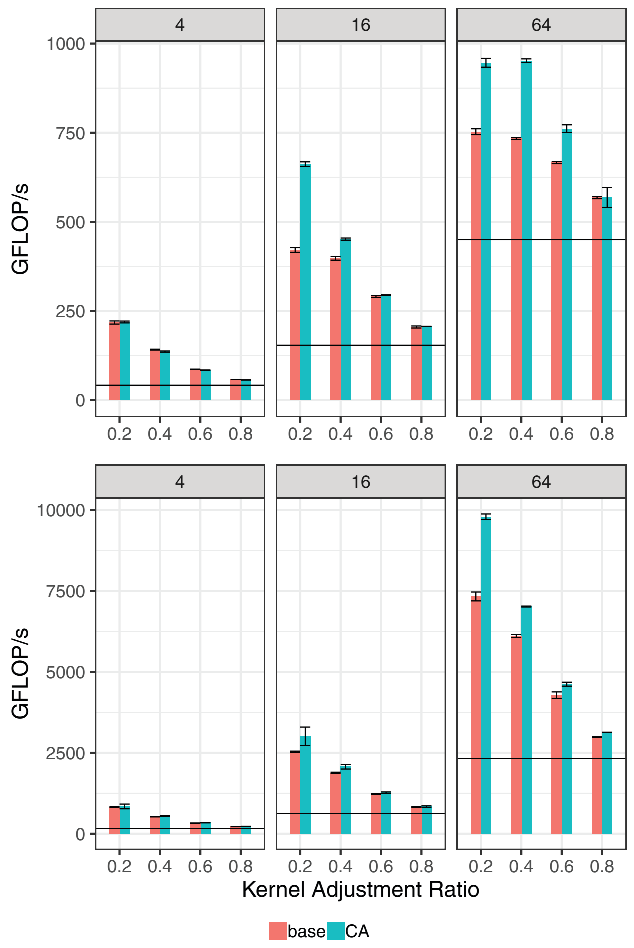

Fig. 8. Tuned kernel performance top) NaCL result with problem size 23k, tile size 288; bottom) Stampede2 result with problem size 55k, tile size 864, running for 100 iterations. Steps size of 15 is used for CA version. Running on 4, 16 and 64 nodes with squared compute grid. The ratio indicates the ratio of mb and nb of tile being operated on, namely $ratio^2$ of the original number of points in a tile. Black lines indicate the base PaRSEC with original kernels' result.

speedup the kernel execution. mb and nb are the rows and columns number of a tile respectively. Figure 8 shows that in such case, communication avoiding can provide a decent amount of improvements, for example the NaCL 16 nodes case we can see a 57% improvement if the kernel time is small. While on 16 Stampede2 nodes, a moderate 18% improvement can be observed in that case. The fast kernel times we assume here is quite realistic as well. Based on STREAM memory bandwidth test result, 0.6 ratio kernel performance is similar to reaching around 80% of STREAM bound. According to recent study [14], it is an efficiency level achieved with optimized kernel.

The step size affects how often the boundary tiles will communicate with each other, the size of the message and the amount of available tasks can be enabled in this interval. Although in our implementation, it will have no impact on the

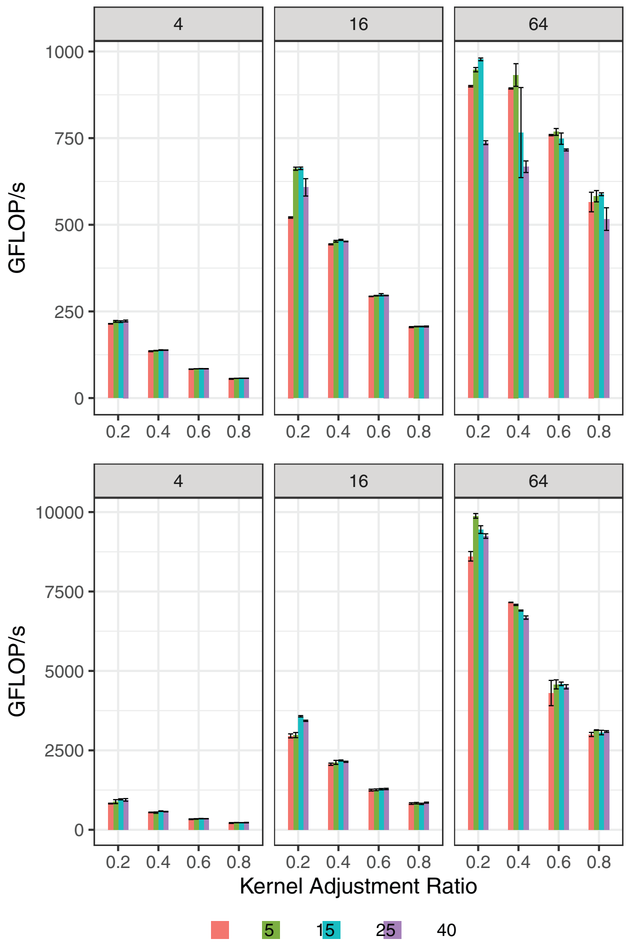

Fig. 9. Tuned step size performance top) NaCL results with problem size 23k, tile size 288; bottom) Stampede2 results with problem size 55k, tile size 864, running for 100 iterations. Step sizes of 5, 15, 25 and 40 are used

boundary tasks' execution time since we simulates the kernel time without the extra computation. The interplay between step size and kernel execution time is complicated, but the optimal step size can be searched via experiment runs. Figure 9 indicates that if communication avoiding scheme can improve performance over the base version, the step size needs to be tuned to get the best possible speedup.

E. *PaRSEC Profiling of the Two Versions*

To validate that the communication avoiding versions' indeed reduces the network latency thus reducing the cores idling time, we used PaRSEC's profiling system to record the execution trace of the tasks to generate Figure 10. The result from figure 8 shows that for tuned ratio of 0.4 running on 16 nodes on NaCL, we get a 14% performance improvement. From the execution trace we can see that indeed with the help of the CA approach, more tasks can get executed while network messages are exchanged and we generally have higher CPU occupancy. And the faster execution is achieved despite the fact that the base version has median kernel time of 136 milliseconds while CA version has median kernel time of 153 milliseconds due to the extra copies in the body.

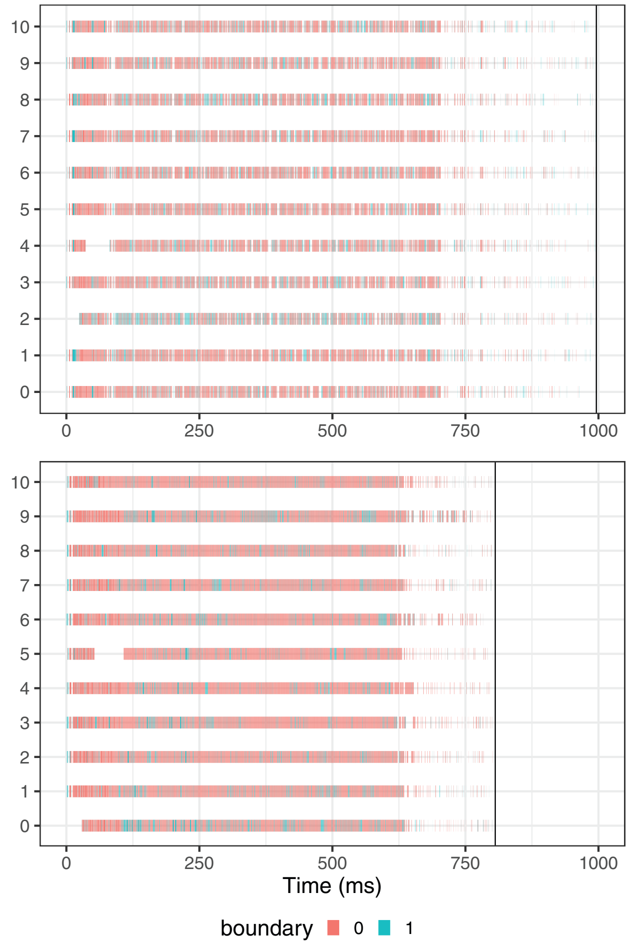

Fig. 10. One node's profiling result, running on NaCL with 16 nodes, tuned ratio of 0.4, 11 computation threads on a node. top) Base PaRSEC bottom) CA PaRSEC. Boundary indicates the tiles that need to exchange data with remote nodes.

VII. Conclusions and Future Work

In this work, we described, implemented, and analyzed a 2D stencil and its communication-avoiding (CA) variant on top of the PaRSEC runtime system. In particular, we proposed three implementations of a 5-point stencil as our test cases. We showed performance results on two distinct systems: NaCL and Stampede2; and compared three versions: PETSc, base PaRSEC and CA PaRSEC. The approaches based on a tasking runtime show good performance results, with minimal distinction between the two approaches in all compute-intensive scenarios. By artificially reducing the kernel execution time, we highlight the case where the CA variant on top of PaRSEC is able to outperform the others in the strong scaling regime with up to 57% and 33% improvements on both the NaCL and Stampede2 systems.

On the current state-of-the-art high performance computing system such as Department of Energy's Summit at Oak Ridge National Laboratory, each node has 6 GPUs and 900 GB/s memory bandwidth per GPU and showed a network latency of about 1 microsecond [33]. Exascale systems are expected to be delivered over the next few years, and some information about their architecture has been made public. The memory bandwidth is expected to have around 50% improvement, but the improvement of network latency will remain modest – a well established trend [8]. Thus, if the workload on each node can efficiently utilize the full memory bandwidth then it would become, in all likelihood, network-bound and the implementation variant based on communication-avoiding approach shows a distinct advantage. However, increasing the arithmetic intensity of the algorithms, or increasing workload on each node could also provide effective ways to mitigate the network inefficiencies.

Another way to look at the benefits of the communication avoiding approach is how it aggregates the data across several iteration steps. This reduces the communication frequency to counteract the latency overhead and thus transforming a latency-bound algorithm into a bandwidth-bound one. This also allows us to more efficiently use the network due to communicating larger messages that allow increased bandwidth efficiency from 20% percent to 70% of peak network bandwidth as shown in the NetPIPE results in Figure 5. By performing redundant computations, we delayed the network latency penalty by strong-scaling to larger node counts.

As a potential future work, we plan to investigate the possibility of providing a more generic communication avoiding framework that would be built directly into the runtime system. This approach will include automatic data replication across the stencil grid neighbors, i.e., the nodes that share a frontier region. Under such a design, the generation and the scheduling of the redundant tasks become transparent to the users and thus make the advantages of this approach widely available to codes that have a high threshold to non-trivial algorithmic changes.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant No. 1740250, and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, under UT Battelle subaward 4000153505.

REFERENCES

[1] G. Golub, J. Ortega, Scientific Computing, an introduction with Parallel Computing, Academic Press, 1993.

[2] W. Hackbusch, Multigrid Methods and Applications, Springer Series in Computational Mathematics Vol. 4, Springer-Verlag, Berlin, 1985.

[3] U. Trottenberg, C. W. Oosterlee, A. Schüller, Multigrid, Academic Press, London NW1 7BY, UK, 2001.

[4] L. N. Trefethen, D. Bau, Numerical Linear Algebra, SIAM, Philadelphia, PA, 1997.

[5] P. Ghysels, W. Vanroose, Modeling the performance of geometric multigrid stencils on multicore computer architectures, SIAM Journal on Scientific Computing 37 (2) (2015) C194–C216. doi:10.1137/130935781.

[6] S. Williams, D. D. Kalamkar, A. Singh, A. M. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, L. Oliker, Optimization of geometric multigrid for emerging multi- and manycore processors, in: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–11. doi:10.1109/SC.2012.85.

[7] P. Basu, A. Venkat, M. Hall, S. Williams, B. Van Straalen, L. Oliker, Compiler generation and autotuning of communication-avoiding operators for geometric multigrid, in: 20th Annual International Conference on High Performance Computing, 2013, pp. 452–461. doi:10.1109/HiPC.2013.6799131.

[8] S. L. Graham, M. Snir, C. A. Patterson, Getting up to speed, the future of supercomputing, The National Academies Press.

[9] M. Bauer, S. Treichler, E. Slaughter, A. Aiken, Legion: Expressing Locality and Independence with Logical Regions, in: International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2012. doi:10.1109/SC.2012.71.

[10] J. Bachan, S. B. Baden, S. Hofmeyr, M. Jacquelin, A. Kamil, D. Bonachea, P. H. Hargrove, H. Ahmed, UPC++: A high-performance communication framework for asynchronous computation, in: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019, pp. 963–973. doi:10.1109/IPDPS.2019.00104.

[11] C. Augonnet, S. Thibault, R. Namyst, P. Wacrenier, StarPU: A unified platform for task scheduling on heterogeneous multicore architectures, Concurrency Computat. Pract. Exper. 23 (2011) 187–198.

[12] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, J. J. Dongarra, PaRSEC: Exploiting Heterogeneity to Enhance Scalability, Computing in Science Engineering 15 (6) (2013) 36–45.

[13] L. V. Kale, S. Krishnan, CHARM++: A portable concurrent object oriented system based on C++, in: Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA '93, Association for Computing Machinery, New York, NY, USA, 1993, p. 91–108. doi:10.1145/165854.165874.

[14] T. Zhao, S. Williams, M. Hall, H. Johansen, Delivering performance-portable stencil computations on cpus and gpus using bricks, in: 2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2018, pp. 59–70. doi:10.1109/P3HPC.2018.00009.

[15] Y. Zhang, F. Mueller, Auto-generation and auto-tuning of 3D stencil codes on GPU clusters, in: Proceedings of the Tenth International Symposium on Code Generation and Optimization, CGO '12, Association for Computing Machinery, New York, NY, USA, 2012, p. 155–164. doi:10.1145/2259016.2259037.

[16] J. Demmel, M. Hoemmen, M. Mohiyuddin, K. Yelick, Avoiding communication in sparse matrix computations, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–12. doi:10.1109/IPDPS.2008.4536305.

[17] M. Hoemmen, Communication-avoiding Krylov subspace methods, Ph.D. thesis, USA, aAI3413388 (2010).

[18] E. Solomonik, J. Demmel, Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms, in: E. Jeannot, R. Namyst, J. Roman (Eds.), Euro-Par 2011 Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 90–109.

[19] E. Georganas, J. Gonzalez-Dominguez, E. Solomonik, Y. Zheng, J. Tourino, K. Yelick, Communication avoiding and overlapping for numerical linear algebra, in: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–11. doi:10.1109/SC.2012.32.

[20] E. Agullo, L. Giraud, A. Guermouche, S. Nakov, J. Roman, Pipelining the CG Solver Over a Runtime System, in: GPU Technology Conference, NVIIDA, San Jose, United States, 2013.
URL https://hal.inria.fr/hal-00934948

[21] I. Yamazaki, M. Hoemmen, P. Luszczek, J. Dongarra, Improving performance of gmres by reducing communication and pipelining global collectives, in: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1118–1127.

[22] K. Datta, Auto-tuning stencil codes for cache-based multicore platforms, Ph.D. thesis, USA (2009).

[23] A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, J. Dongarra, Ptg: An abstraction for unhindered parallelism, in: 2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing, 2014, pp. 21–30. doi:10.1109/WOLFHPC.2014.8.

[24] R. Hoque, T. Herault, G. Bosilca, J. Dongarra, Dynamic Task Discovery in PaRSEC: A Data-flow Task-based Runtime, in: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '17, ACM, New York, NY, USA, pp. 6:1–6:8.

[25] J. Demmel, M. F. Hoemmen, M. Mohiyuddin, K. A. Yelick, Avoiding communication in computing Krylov subspaces, Tech. Rep. UCB/EECS-2007-123, EECS Department, University of California, Berkeley.

[26] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Web page (2019).

[27] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.12, Argonne National Laboratory (2019).
URL https://www.mcs.anl.gov/petsc

[28] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.

[29] S. Moustafa, W. Kirschenmann, F. Dupros, H. Aochi, Task-based programming on emerging parallel architectures for finite-differences seismic numerical kernel, in: M. Aldinucci, L. Padovani, M. Torquati (Eds.), Euro-Par 2018: Parallel Processing, Springer International Publishing, Cham, 2018, pp. 764–777.

[30] D. Turner, A. Oline, X. Chen, T. Benjegerdes, Integrating new capabilities into netpipe, in: J. Dongarra, D. Laforenza, S. Orlando (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 37–44.

[31] J. D. McCalpin, STREAM: Sustainable memory bandwidth in high performance computers, Tech. rep., University of Virginia, Charlottesville, Virginia, a continually updated technical report. http://www.cs.virginia.edu/stream/ (1991-2007).
URL http://www.cs.virginia.edu/stream/

[32] S. Williams, A. Watterman, D. Patterson, Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures, Communications of the ACM.

[33] S. S. Vazhkudai, B. R. de Supinski, A. S. Bland, A. Geist, J. Sexton, J. Kahle, C. J. Zimmer, S. Atchley, S. Oral, D. E. Maxwell, et al., The design, deployment, and evaluation of the coral pre-exascale systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18, IEEE Press.