Efficient Large-scale Parallel Stencil Computation on Multi-Core and Multi-GPU Accelerated Clusters

Zhengchun Liu[a,b,c], Kalyan S. Perumalla[a,*]

[a] *Oak Ridge National Laboratory, Oak Ridge, TN, USA*
[b] *Computer Architecture & Operating Systems, University Autonoma of Barcelona, Barcelona, Spain*
[c] *Mathematics and Computer Science Division, Argonne National Laboratory Lemont, IL, USA*

Abstract

In today's computing system, heterogeneous architectures predominate the modern high-end platforms. Stencil computations and agent based simulations offer significant potential of computational concurrency to exploit such heterogeneous accelerated systems. However, distributed memory across nodes needs new algorithmic or hardware optimization to overcome high memory-memory transfers. In this paper, we analyze expanded ghost layers to improve multi-node performance on large heterogeneous accelerated systems via algorithmic optimization, and direct memory-memory transfers for optimized execution on high-end hardware. We present performance evaluation on 3-D models, variable stencil neighborhoods, and multiple application benchmarks. An agent-based model and a scientific model that solves partial differential equations by using the finite difference method are used as case studies for our latency-hiding scheme a in hierarchical computer organization. We scaled the hierarchical implementations to 1,000 GPUs and 16,000 CPU cores of a supercomputing system. Test results on a supercomputer indicate the benefits of our latency-hiding scheme, delivering as much as 9× faster runtime performance than conventional implementation without latency-hiding scheme. Importantly, the algorithmic software-level enhancements are shown to fetch performance comparable to that of hardware-enhanced implementation.

Keywords: Computational hierarchy, Multi-core, Multi-GPU, Latency hiding, CUDA

1. Introduction

Computer simulation is one of the most generally applied approach to study natural systems in physics, chemistry and biology, and human systems in economics and social science. Efficient simulation methods enable researchers to study more scenarios and larger scale in short time frame. Iterative stencil loops are commonly used in many scientific programs across many disciplines to implement relaxation methods for numerical simulation and signal processing [1]. Stencil computation, which has lots of concurrency potential in the form of single instruction multiple data (SIMD) type of computation, is an important class of codes used in a variety of application domains ranging from image and video processing to simulation and computational science applied in several areas of natural science. Stencil codes are compute-intensive, in which data points arranged in large grids need to be recomputed repeatedly from the previous values of data points in a predefined neighborhood. This application-specified neighborhood pattern is called a stencil. Stencil computation is an attractive and powerful method to explore the challenges of massive parallelism in solving partial differential equations. As investigated by many researchers in the literature [1, 2, 3, 4, 5], tiling is a well-known technique to localize iterative stencil loops computation, which involves halo regions that need to be updated, exchanged and synchronized among different subdomain regions mapped to the processing elements in a parallel architecture.

☆Corresponding author at: Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA. Tel.: +1 (865) 241-1315; Fax: +1 (865) 576-0003.
This work was partially done while Zhengchun Liu was with Oak Ridge National Laboratory as a visiting scholar.
Email address: perumallaks@ornl.gov (Kalyan S. Perumalla)

1.1. Motivation

Computing technologies such as CPU, GPU, FPGA and Intel Xeon Phi processors, are becoming faster, which have historically improved at the rate of roughly 50% per year [6], and less expensive, according to Moore's law. However, memory latency has not improved as dramatically, which improved at a rate of only 7% per year [6], and memory access times are increasingly limiting system performance. This is a phenomenon known as the *Memory Wall* [7, 8]. Thus, the gap between CPU processing speed and memory access rate has been becoming bigger. To bridge this gap, efforts in both system architecture design and user application are required [5]. The main memory in a parallel computer is either shared among all processing elements in a single address space, or distributed, in which each processing element has its own local address space [9]. Distributed memory refers to the fact that the memory is logically distributed, but often implies that it is physically distributed as well.

Analogous to the memory wall, due to the large and growing gap between computation speed and inter-node communication speed in a grid environment, new methods need to be developed to balance computation with communication. Memory system performance is largely captured by two parameters, namely, latency and bandwidth. Latency is the time gap from the issue of a memory request to the time the data is available at the processor. The bandwidth, mostly determined by hardware specification, is the maximum rate at which data can be pumped to the processor by the memory system. This paper concentrates on hiding the latency of stencil computation models and agent-based models on multi-GPU and multi-Core hierarchical organization.

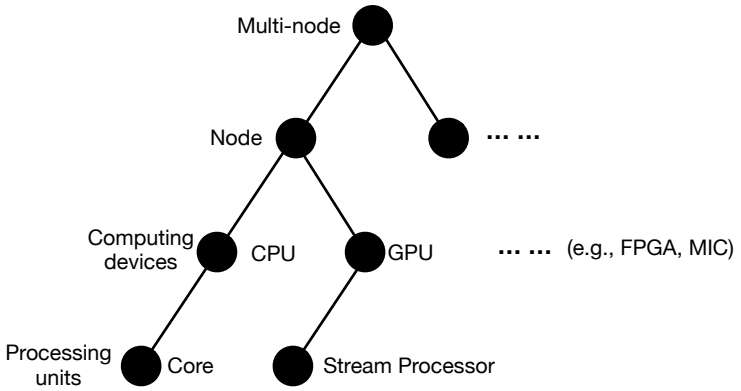


Figure 1: The state-of-the-art parallel computing platform architecture. Computing tasks are hierarchically decomposed. At one level, the compute capability of difference items are different.

The state-of-the-art parallel computing platforms are commonly composed of multiple compute nodes. These nodes are interconnected by high-speed interconnection, and each node contains multiple CPU cores and zero or more accelerators such as GPU and FPGA. Effective use of these components presents significant performance challenges in the form of communication-oriented and memory access-based bottlenecks. The question of how best to utilize these resources is an important issue. As shown in Figure 1 and Figure 2, a large-scale simulation problem is usually decomposed first to subdomains for compute nodes, then further decomposed to tasks for computing devices on each of the node, and finally tasks broken down for processing units according to the specific features of computing devices.

To deal with the large stencil computation model, data parallelism is commonly used. In particular, on a distributed system, each processor holds a subset of the problem domain, referred to as problem subdomains. Each subdomain is padded with one or several boundary layers, which are usually called ghost cells [5]. A thorough investigation on the effectiveness of these boundary layers in combination with the computational dynamics of the model is critical to help simulation developers to more efficiently utilize the heterogeneous, large-scale parallel computing platform.

1.2. Background

Our previous study [10] explored the computation versus communication trade-off continuum available with the deep computational and memory hierarchies of extant platforms, and presented a novel analytical model of the trade-off. However, it was restricted to 2D simulation, and only applicable to those models which update elements according to one-layer away neighborhoods (i.e., computing-dependency order equals one). Unfortunately, in many applications,

such as in climate modeling and seismic wave propagation model, updating elements according to three or even more layers are common in a 3D environment. Our new work in this paper relaxes and generalizes the computation versus communication trade-off in the 3D environment with different computing-dependency order. A new analytical model is also developed to determine the optimum computation-communication trade-off point.

1.3. Contributions

This study presents a latency-hiding mechanism to exploit the hierarchical organization and heterogeneity of the state-of-the-art high performance computing platforms. The latency hiding is based on the principle of computation versus communication tradeoff. In other words, it trades off the duplication of some computation to gain some concurrency to offset communication latency [10]. We call it the "*B+2R latency-hiding scheme*" as defined previously [10], or as *B2R* in short. Here $B = \{B_x, B_y, B_z\}$ refers to the size of subdomain at each direction, and R represents the size of ghost zone. Note that R is measured in units of the computing-dependency order of stencil in the application model. For example, if a model has computing-dependency order as δ, our B2R scheme will pad $R\delta$ layers of ghost cells to each dimension of the subdomain. These notations will be used throughout the entire article.

Our work extends previous work on the B2R latency-hiding scheme to 3D grid environment. In addition, this work also studies the effectiveness of B2R with different layers of data-dependence (referred by δ in this context) on updating the state of each element.

To facilitate implementation and optimization, the following are developed: (a) an analytical model to determine the optimum B2R parameters, (b) a framework template [1] to update/synchronize arbitrarily defined subdomain boundaries, and (c) an auto-tuning balance scheme for heterogeneous computing devices in grid environment. In summary, the main contributions of this work are: (1) we extend the *B2R* scheme model from 2D to 3D; (2) we generalize from fixed stencil ratio $\delta = 1$ to arbitrary $\delta \geq 1$; (3) from computing via CPU or GPU to with both CPU and GPU, and design of an auto-tuning, self-balancing task-alignment scheme for CPU and GPU. Compared with other related works that focused either in grid environment with multi-core or in GPU with CUDA, this study explores the latency hiding mechanism to accommodate the hierarchical organization of heterogeneous computing platforms. Furthermore, a benchmark agent-based model, multiple implementations of heat diffusion model with different order of accuracy (vary δ) and a computationally expensive seismic wave propagation model are used to benchmark the proposed latency hiding scheme. The source code is available online at `https://github.com/lzhengchun/b2r`.

The rest of the paper is structured as follows: section 2 gives a literature review on related work. A detailed description of our latency hiding scheme is given in section 3, in which an analytical performance model is formulated for analyzing the potential optimum configuration. Section 4 describes the implementation details. A comprehensive performance study with several applications are presented in section 5. Finally, section 6 closes the article with our conclusions and potential future contributions.

2. Related work

Stencil codes are compute-intensive algorithm, in which data points arranged in a large grid are recomputed repeatedly from the values of data points in a predefined neighborhood. The fixed neighborhood pattern is called a stencil [3]. Stencil codes are widely used in computing the discrete solutions of partial differential equations and systems composed of such equations. Given this importance of application of stencil-based computations, how to efficiently execute stencil codes in the state-of-the-art parallel computing platforms is a question that has attracted many investigations. J. Meng et al. [2] thoroughly investigated the effectiveness of ghost zone in CUDA parallel computing platform. The authors proposed a framework template that automatically incorporates ghost zones to iterative stencil loops applications in normal CUDA code and optimize them with the selection of trapezoidal configurations. Furthermore, J. Meng et al. [2] also found that smaller stencils with smaller halo widths can benefit more from the trapezoid technique; they tested the performance with 1D, 2D and 3D model, and found that the 1-D PathFinder benefits the most, followed by 2-D HotSpot and Poisson model, but the 3-D Cell (Game of Life) does not benefit at all. They finally concluded that the benefit of ghost zones may be more significant for architectures that easily allow for larger

[1] available at `https://github.com/lzhengchun/b2r`

tile sizes. However, larger thread block size and larger shared memory to host bigger subdomain data and ghost zone data seems neither possible nor necessary on GPU-based accelerated systems.

In finite difference methods to solve partial difference equations on distributed memory computers, subdomain boundaries must be updated at each time step. This boundary updating process involves many messages of small sizes, and therefore results a in large communication overhead. D. Chris et al. [5] investigated the ghost cell layers padding scheme, which updates boundaries much less frequently, reducing total message volume via grouping small messages into bigger ones. However, their work only covered message-passing systems in a grid computing environment; it did not involve the state-of-the-art heterogeneous parallel computing platform like the one with GPUs. With respect to large-scale scientific simulations such as seismic wave propagation, which involve solving multiple partial differential equations on large 3D domain. Zhou et al. [11] presented an implementation and optimization investigation on GPU with CUDA, but they did not explore performance on multiple GPUs. Cui et al. [12] explored an implementation of earthquake wave propagation model on peta-scale supercomputer, and their implementation has been scaled to more than two hundred CPU-cores. However, they only considered ghost zones with a single layer and only execution on CPU. Multiple ghost zone layers achieve better tradeoff between communication and computation on large distributed memory system and, with multi-GPU, can further accelerate the simulation.

Although the increasing computational speed of GPUs make their use for stencil computations an interesting goal, achieving highly efficient implementations is often nontrivial [13]. Li et al. [1] presents a compiler framework to automatically tile the iterative stencil loops, with the objective of improving the cache performance. There is also some work on performance model-based auto-tuning of stencil computations. For example, Hu et al. [13] proposed an analytical performance model for stencil codes on GPUs. Lutz et al. [14] focused on abstracting the complexity of multi-GPU (restricted to one computing node) programming for stencil computation. They demonstrated that adaptation to a given PCI express configuration is a significant factor in achieving high performance. Datta et al. [15] developed a number of effective optimization strategies, and built an auto-tuning environment that searches optimization and their parameters to minimize runtime, while maximizing performance portability on multi-core architectures. They highlighted that auto-tuning is critically important for unlocking the performance potential across a diverse range of chip multiprocessors. Moreover, they targeted at keeping the programmer's effort low without a significant sacrifice in performance. Zhang et al. [16] developed and evaluated search and optimization techniques for auto generated, auto-tuning 3D stencil computations on GPUs.

3. Approach description

3.1. Task decomposition

Data parallelism is commonly used for efficiency to parallelize the large simulation problem to many computing nodes Using this kind of decomposition, the domain is decomposed into small subdomains and distributed among all computing nodes; therefore, each processor is assigned its own subdomain problems. Since each computing node has CPU and accelerating devices, the subdomain will be further decomposed for computing devices. Figure 2 shows the 3D domain decomposition process.

For decomposition on CPU- and GPU- based heterogeneous parallel computing platform, the 3D domain is partitioned into a number of subdomains, and each subdomain is reserved for computation on a computing node. Here we use X, Y, Z to represent columns, rows and depth, respectively. The memory for holding the subdomain is managed in row-major order, i.e., the first dimension X (column) changes the fastest and the last dimension Z (depth, in this case) changes the slowest.

As shown in Figure 2, on each computing node, the subdomain is partitioned into two parts (along Oz), N_{GPU} and N_{CPU} slices for GPU and CPU respectively. Suppose that the 3D subdomain is represented as (N_{bx}, N_{by}, N_{bz}) cells. Then CPUs hold $(N_{bx}, N_{by}, N_{CPU})$ cells, and each GPU holds $(N_{bx}, N_{by}, N_{GPU})$ cells, where $N_{bz} = N_{CPU} + N_{GPU}$. Lastly, we need to decompose the CPU domain for CPU cores. As the memory address varies the slowest in Oz direction, to optimize cache hits, the job is partitioned with 1D along Oz direction, i.e., one core is responsible for several neighboring Oxy slices. To partition for GPU streaming multiprocessors (SM), we choose 2D decomposition on the Oxy plane, so that each SM takes over the entire computation in Oz direction.

In this decomposition, it is never needed for any single node to host the total problem domain. This makes it scalable for very large-scale simulation domain. Thus its scalability for large problem size will be investigated in

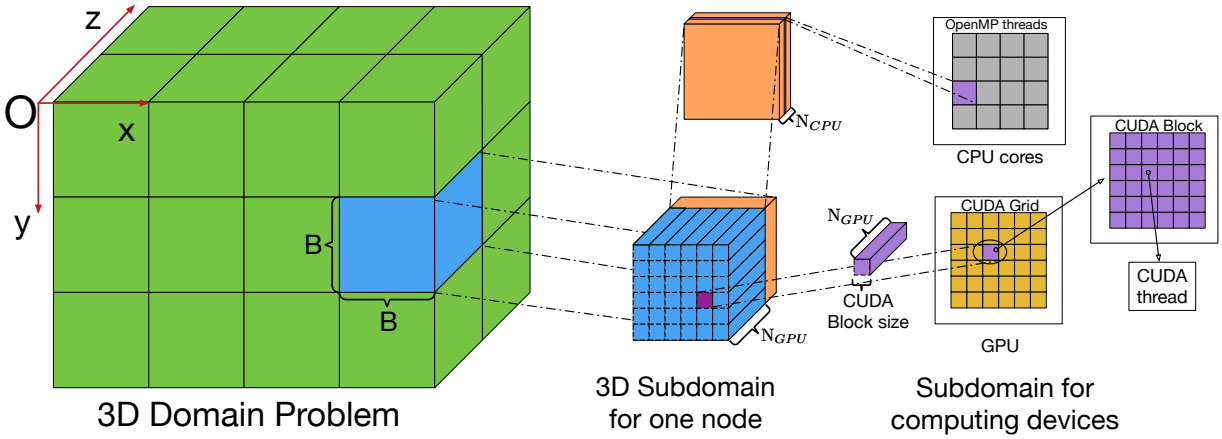


Figure 2: Process of 3D domain decomposition on CPU/GPU based multi-node heterogeneous computing platform: first decompose for nodes, then decompose for computing devices (GPU and CPU, along Oz direction, N_{GPU} and N_{CPU} respectively) on the node, then decompose for processors (CPU cores and GPU streaming multiprocessors).

§5.5. We emphasize that the domain and subdomain are not essential to be cubic, can be cuboid or even a flat 2D plane ($N_{bz} = 1$). Other decomposition methods for specific problem, such as the two-layer decomposition [4] where each subdomain takes over the entire computation in the Oz direction, can be achieved by setting one computing node in Oz direction.

3.2. Ghost expansion

In stencil computations or agent based models, updating one cell at time t requires the current state of its neighbors. The range of neighbors depends upon the specific model (in stencil computing, this range is determined by the order of numerical accuracy required). In this article, we use δ to represent range (i.e., at a Chebyshev distance [17] of δ) of neighborhood. Thus, each cell has $(2\delta + 1)^n - 1$ and $2\delta n$ cells in an n-Dimension environment ($n = \{1, 2, 3\}$) for Moore neighborhood (e.g., agent-based model), and Von Neumann neighborhood (e.g., finite difference method), respectively. As we aim to synchronize less often with bigger message sizes in order to reduce communication latency, $R\delta$ layers will be padded to subdomains as depicted in detail in Figure 2 at every level (node, accelerator devices, multi-core processing units). Computation on local subdomain can then increase to R iterations before having to re-synchronize with neighbors. Since this will result in more computation, there will be a trade-off between extra computation and reduced communication latency. The feasibility of this idea and an in-depth analysis of the method will be given in §3.4.

3.3. Boundaries updating

As discussed in §3.1 and §3.2, ghost padding must be synchronized after every R local iterations for consistency. Table 1 lists the synchronization process in a 3D decomposition using this methods. This diagonal communication elimination technique was first implemented [18] in a slightly different form and later explored in [5] in 2D scenarios. It requires only communications with von Neumann neighbors, and thus can reduce the number of messages from 8 to 4 in 2D, and 26 to 6 in 3D scenario. To be clear, as illustrated in Figure 3, in this technique the corner blocks are moved twice to reach their final destinations in 2D, and the cubic diagonal blocks are moved 3 times to reach their final destinations. Since this technique requires step-by-step synchronizing, according to our experiments, it performs better for small message size than synchronizing directly with all Moore neighbors concurrently. No significant difference was found with big sized messages. Its performance also depends on network bandwidth and topology. Since messages exchanged between diagonal subdomains are usually small, $(R\delta)^n$ in nD ($n = \{1, 2, 3\}$) domains, this diagonal communication elimination technique was used in this study. It is worth noting that if a subdomain spans the entire computation in any of the Ox, Oy, Oz direction, these directions will not need synchronization. For example, as shown in Figure 2, the CPU/GPU subdomain on the node was partitioned only in Oz (spanning entire Ox and

Oy) direction. Therefore, it only needs the third step for updating ghost padding along Oz direction. Similarly, the decomposition for GPU SMs needs synchronization in Ox and Oy direction.

Table 1: Boundary updating steps and message size.

Step	Direction	Valid before updating	Updating size	Valid after updating
1	Ox (Left, Right)	$B_x \times B_y \times B_z$	$R\delta \times B_y \times B_z$	$(B_x + 2R\delta) \times B_y \times B_z$
2	Oy (Up, Down)	$(B_x + 2R\delta) \times B_y \times B_z$	$(B_x + 2R\delta) \times R\delta \times B_z$	$(B_x + 2R\delta) \times (B_y + 2R\delta) \times B_z$
3	Oz (Front, Back)	$(B_x+2R\delta)\times(B_y+2R\delta)\times B_z$	$(B_x+2R\delta)\times(B_y+2R\delta)\times R\delta$	$(B_x+2R\delta)\times(B_y+2R\delta)\times(B_z+R\delta)$

More specifically, in Table 1, consider the compute-node level (hierarchy in Figure 1) synchronization as an example and assume it decomposes in all three directions. There are $B_x \times B_y \times B_z$ valid domain blocks left after R local iterations. $R\delta$ layers of data surrounding the valid domain need to be refilled with state information from neighboring subdomains. As shown in Table 1, it requires three consecutive steps to update all boundaries from its neighbors. After synchronization, execution can continue for another R iterations. An illustration of the three-step synchronization is shown in Figure 3.

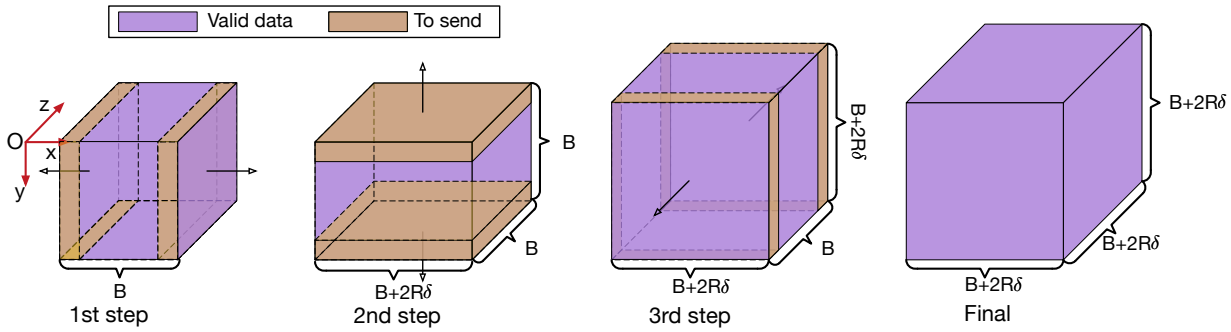


Figure 3: Illustration of updating ghost cells in 3D decomposition. 6 ghost cells blocks are from immediate neighbors.

Figure 3 illustrates the steps to send data to neighbors. Note that all the receiving processes are the same. Each node will receive the same amount of data from its neighbors, and the received data will be padded to the corresponding ghost zones. The non-blocking communication functions of Message Passing Interface (MPI) (*MPI_Isend* / *MPI_Irecv*) were used for this synchronization process.

3.4. Analytical model

In the this section, we examine the feasibility of the proposal described in §3.1, §3.2 and §3.3, and give an in-depth analysis of the method. Our framework template supports both cubic and non-cubic subdomains. Here, to simplify the model description, we assume that all the subdomains are cubic. All the computation and communication time models are formulated as

$$y = a_{(c,m)}^{(C,G)} x + b_{(c,m)}^{(C,G)} \tag{1}$$

where, the subscript c and m of the parameters denote computation and communication respectively, the superscripts C and G represent CPU and GPU respectively, x represents operation (compute or communicate) size, and y is the operation time.

Since only the stencil operations within the valid tile are performed in each iteration when computing on CPU (i.e., after one iteration, 2δ layers will lose validity in each dimension), the total computation time for updating R simulation steps on CPU can be formulated as:

$$F_c^{CPU} = a_c^C \sum_{i=1}^{R} [B + 2(R-i)\delta]^3 + b_c^C = a_c^C[2\delta^3R^4 + (4B\delta^2 - 4\delta^3)R^3 + (3B^2\delta - 6B\delta^2 + 2\delta^3)R^2 + (B^3 - 3B^2\delta + 2B\delta^2)R] + b_c^C \tag{2}$$

where, a_c^C is the unit time for updating one cell in the grid. Equation 2 can be easily extended to non-cubic grid (different $B \in \{B_x, B_y, B_z\}$ for each domain). However, on GPU platforms, although the subdomain size decreases 2δ after each iteration in each dimension, as previously investigated [2], the changing-boundary conditional if-testing increases the amount of computation and leads to more control-flow divergence within GPU warps, which undermines SIMD performance. Therefore, the full domain can be computed as if its tile size does not change along with the loops. At the end, only those elements that fall within the boundary of the shrunk tile are committed to the device memory. Computing in this manner results in the following analytical computing time model:

$$F_c^{GPU} = R\{a_c^G[(B + 2R\delta)^3] + b_c^G\} = a_c^G[8\delta^3R^4 + 12B\delta^2R^3 + 6B^2\delta R^2 + B^3R] + b_c^G R \tag{3}$$

To analyze the communication time, F_m, we assume it can be approximated by a simple model based on (*message−volume*)/*bandwidth* + *latency*. i.e.,

$$F_m = a_m x + b_m \tag{4}$$

where a_m denotes the inverse of bandwidth (time per byte), and b_m represents the latency. Both a_m and b_m are platform-specific communication overhead constants. Thus the network communication (F_m^{MPI}) and host-to-device($H2D$) and device-to-host($D2H$) memory transferring time ($F_m^{H2D/D2H}$) for one synchronization operation (R local iterations) can be expressed as:

$$F_m^{MPI} = a_m^M[(B + 2\delta R)^3 - B^3] + b_m^M = a_m^M[8\delta^3R^3 + 12B\delta^2R^2 + 6B^2\delta R] + b_m^M \tag{5}$$

$$F_m^{H2D/D2H} = a_m^G[(B + 2R\delta)^3 + B(B + 2R\delta)^2] + b_m^G = a_m^G[8\delta^3R^3 + 16B\delta^2R^2 + 10B^2\delta R + 2B^3] + b_m^G \tag{6}$$

Note that MPI communications are assumed to be in a full-duplex model because the send/receive data size are the same and done simultaneously. However, the host-to-device and device-to-host communication are done in sequence, the first and last $R\delta$ layers (only convenient along Oz direction because Z changes the slowest) do not need to be copied to host since they are invalid data after R local iterations. Therefore, the runtime on pushing one simulation time-step ahead can be formulated as:

$$F_{step} = (F_c + F_m^{MPI} + F_m^{H2D/D2H})/R = (a_3R^3 + a_2R^2 + a_1R + a_0) + (b_m^M + b_m^G + 2a_m^G B^3)/R \tag{7}$$

where, the computation time F_c depends upon the type of computing device, i.e., CPU or/and GPU. If both CPU and GPU are used, the task decomposition scheme should balance their task size (because $F_c = min\left(F_c^{GPU}, F_c^{CPU}\right)$), that is, $F_c = F_c^{GPU} = F_c^{CPU}$. This scheme will be further discussed in §3.5. Here, in Equation 7, we use $F_c = F_c^{GPU}$ as an example. In Equation 7, $a_0 = a_c^G B^3 + b_c^G + (6a_m^M + 10a_m^G)B^2\delta$, $a_1 = 6a_c^G B^2\delta + (12a_m^M + 16a_m^G)B\delta^2$, $a_2 = 12a_c^G B\delta^2 + 8(a_m^M + a_m^G)\delta^3$, $a_3 = 8a_c^G\delta^3$. Consider Equation 7 as a function of R. It has two parts: a polynomial of degree 3 and a term with the inverse of R. Since a_1, a_2 and a_3 are very small number ($\leq 10^{-9}$) but greater than zero, the polynomial will increase with R ($R \in \mathbf{Z}^+$). However, the second part is a non-negative, monotonically decreasing function. Therefore, the second part with a_0 dominates F_{step} when R is small. The first part becomes dominant when R is greater than some value dependent on the computing platform.

This indicates two traits. The first is that the runtime shall experience a significant decrease as R increases. The second is that there will also exist an R value at which the platform no longer favors computation over communication; in other words, there will be a fixed R for a given B and δ for which optimal performance is achieved. Later, in the performance study (§5), we in fact observe the fall and rise of runtime with R, as predicted by the analytical model (actual time and analytical model prediction are compared in Figure 6). These inferences are in fact in line with the observation and empirical findings in stencil-based computations as well, although our model is more general in nature.

3.5. Heterogeneous balance

Multi-node configuration is a common architecture in current state-of-the-art parallel computing platforms. On each of the nodes, there are multiple CPUs with multiple cores, multiple GPUs and other parallel computing devices such as field-programmable gate array (FPGA), and digital signal processor (DSP). As shown in Figure 1, the compute capability of each item (node, device, core) is widely different from others.

With hybrid acceleration, there are different resources working on different tasks or parts of the work. When load is not balanced, some resources will be idling. Therefore, it is crucial to consider hierarchical load balancing. An ideal task assignment should make all the computing devices in all hierarchies accomplish their job within the same time frame. In this study, we designed an automatic scheme to balance the jobs on CPU and GPU. The scheme will break down the job for GPU and CPU according to their computing capability. Their capability is estimated online by actually time instrumenting the code for each R iterations. Users of our scheme only need to assign an initial ratio for the decomposition of CPU and GPU; the scheme will then tune itself to balance the load of CPU and GPU.

As shown in Equation 2, for any given R and δ, the runtime is cubic to the subdomain size B. Considering that the 3D grid data was organized in an 1D array, in which z-direction varies slowest, it is fairly easy to split the computational tasks along the Oz direction, i.e., the unit of task will be a slice in Oxy plane. Thus, the runtime will be linear in the number of slices. With the goal of assigning tasks for GPU and CPU to obtain similar runtime per simulation time-step iteration, the task assignment scheme can be described by the following equation group 8:

$$\begin{cases} \frac{T_C}{N_C} x_C - \frac{T_G}{N_G} x_G = 0 \\ x_C + x_G = N \end{cases} \tag{8}$$

where, T_C and T_G are the CPU time and GPU time respectively, and N_C and N_G are tasks (number of slices) assigned to CPU and GPU. Values for these four arguments are from previous iteration, x_C and x_G are variables that represent the problem size of CPU and GPU respectively for the next iteration. Our experiments, discussed later in Figure 5a, show that the balance scheme can converge in less than five iterations.

4. Implementation

4.1. Inter-node synchronization

The subdomain boundary synchronization includes two parts: inter-node updating via MPI, and host / device exchanging via accelerator API (*cudaMemcpy* provided by CUDA in this case). The inter-node updating process was illustrated in Table 1 and Figure 3. After inter-node boundaries are updated and after CPU/GPU task decomposition (detailed in §3.5), OpenMP threads (equals with number of cores) are invoked for CPU computing and host operation of GPU computing. Specifically, one OpenMP thread is used to perform CUDA related host operation, such as memory transfer and thread synchronization. Other OpenMP threads perform the tasks assigned to CPU. Given that there are $R\delta$ layers padded in each dimension of the subdomain, the GPU and CPU task can iterate R time-steps without synchronizing with neighboring subdomains, i.e., there are only $maxIter/R$ subdomain boundaries synchronization needed for *maxIter* simulation time-steps. The overall flow is shown in Figure 4.

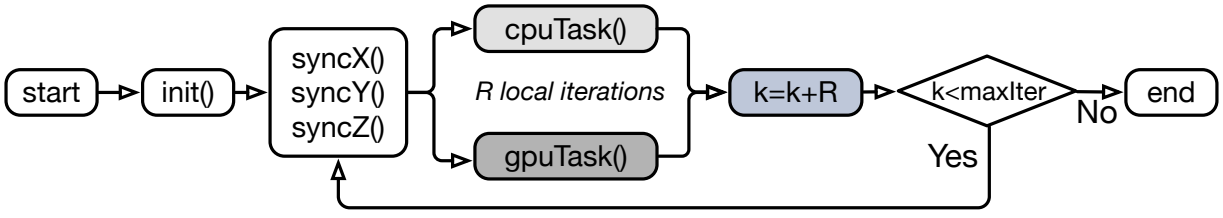


Figure 4: Flow chart of synchronization and computation. As illustrated in Table 1 and Figure 3, *syncX()*, *syncY()* and *syncZ()* are executed in sequence. There are multiple threads in *cpuTask()* for data parallelism, and one thread for *gpuTask()*.

4.2. Benchmark application

With the goal of reducing communication latency at the cost of slightly increased computation, three models (six applications) with different computation complexity and memory access intensities were used to benchmark our *B2R* latency hiding scheme. The first one is a commonly used scientific model as a benchmark, namely, a heat diffusion model. This model has been implemented by using the finite difference method with different orders of accuracy (2nd, 4th, 6th, 8th -order accuracy separately). The second benchmark is a well known agent-based model, namely, John Conway's Game of Life [19] which we extended to 3D. The third is a Seismic wave propagation model that can simulate earthquakes. Details on models are discussed as follows.

Heat diffusion. This is a parabolic partial differential equation that describes the distribution of heat in a given region over time. More generally, it is to solve a partial differential equation:

$$\partial\mu/\partial t - \alpha\Delta^2\mu = 0 \tag{9}$$

where, $\mu(x, y, z, t)$ is the current temperature of time variable t at point (x, y, z), α is the thermal diffusivity, and Δ^2 denotes the Laplace operator. Equation 9 was solved with the finite difference method, which essentially uses a weighted summation of function values at neighboring points to approximate the derivative at a particular point.

3D Game of life. This is a 3D extension of the Game of Life (devised by John Horton Conway in 1970 [19]). In the extended model, any element with only 1 or fewer neighbors (Moore neighborhood, 26-cell cubic neighborhood total in 3D) dies, as if by loneliness; if 5 elements surround an empty element, they breed and fill it, and if a cell has 8 or more neighbors, it dies as if from overcrowding. Therefore, the state of each element is computed as a function of 26 neighboring elements.

Seismic wave propagation. In 3D Cartesian coordinates (x, y, z) for an isotropic and elastic medium this model can be formulated by equations in velocity stress form [12, 20]:

$$\partial_t v = \frac{1}{\rho}(\nabla \cdot \sigma + f) \tag{10}$$

$$\partial_t \sigma = \lambda\,(\nabla \cdot v)\, I + \mu\left(\nabla v + \nabla v^T\right) \tag{11}$$

where, $f = \{f_x, f_y, f_z\}$ are the body force components in three directions (i.e., force from oceanic plates colliding), λ and μ are the Lamé coefficient, ρ is the constant density, $v = \{v_x, v_y, v_z\}$ denotes the particle velocity vector and σ represents the symmetric stress tensor. Expanding Equation 10 and Equation 11 leads to three equations for velocity and six equations for the stress tensor components [20].

In the implementation of the three models, CUDA threads are mapped in Oxy plane and computation iterates along z direction (i.e., compute slice by slice along z direction in a CUDA thread block). For the heat diffusion model, a central finite difference method was used for solving Equation 9 that describes the physical laws of heat transfer. It essentially uses a weighted summation of function values at neighboring points to approximate the derivative at a particular point. In a 3D problem domain, with a (2δ)th-order accuracy, the next data value of each grid point is calculated based on the current data values of $\pm\delta$ in Ox, Oz, Oz directions and itself, i.e., $(6\delta + 1)$-stencil computation. We implemented with $\delta \in \{1, 2, 3, 4\}$ in space, i.e., 7-, 13-, 19- and 25-stencil computation to benchmark our latency hiding scheme in multi-GPU and multi-CPU cores environment. In the CUDA based implementation, as described in [21], shared memory (structured by a 2D array with size: $(blockDim.x + 2\delta) * (blockDim.x + 2\delta)$) was used to buffer an entire slice, and registers are used to buffer δ points in-front and δ points behind in each thread. The current state values loaded into registers and shared memory are shifted ahead at each iteration in the Oz direction. This dramatically saves access to global memory.

Although δ equals 1 in the 3D game of life model, its memory access pattern is quite different with the $\delta = 1$ (7-stencil) special case of heat diffusion model. In the 3D game of life model, updating one cell requires the state values of 26 neighbors. Given this, three 2D array (with size $(blockDim.x + 2) * (blockDim.x + 2)$) in shared memory were used to buffer the current layer (in Oxy plane), one layer in-front, and one layer behind. Then it is iterated along

the Oz direction and shifted three array slots ahead. In summary, the heat diffusion solution with $\delta = 1$, in comparison with the 3D game of life model, requires less computational but more memory access.

In order to solve nine coupled partial differential equations of the seismic wave propagation model, a finite difference method with fourth-order accurate in space and second-order accurate in time was used for solving Equation 10 and Equation 11. This model is a memory-intensive application and twelve 3D variable arrays (three velocity components, three displacement components for visualization, and six symmetric stress tensor components) are involved in the main computation loop. We designed three CUDA kernel functions for computing v_x, v_y and v_z separately, and one kernel function for the six stress tensor components. Similar to solving Equation 9, shared memory was used to reduce device memory access. Compared to the above two models, the seismic wave propagation model is both computationally expensive and memory access intensive. That is to say, with the same subdomain synchronization, the seismic wave propagation model needs more time on computation. This is verified later shown in Figure 7 and Figure 9a, showing that the time ratio of $sync/computation$ is smaller.

5. Performance study

As detailed in §4.2, an agent-based model and two stencil computation models, both programmed with CUDA, were implemented to benchmark the overall performance of the B2R scheme. The heat diffusion model was implemented with different order of accuracy. Since different δ values ($delta \in \{1, 2, 3, 4\}$) result in quite different computational complexity (formulated in Equation 2 and Equation 3) and memory access intensities, there are six distinct models in fact. This section will discuss the benefit of B2R latency hiding with different models and different scales.

5.1. Hardware & Software

All the experiments have been run on the Titan supercomputer at the Oak Ridge National Laboratory. Each computing node is equipped with one 16-core AMD Opteron 6274 processor and one NVIDIA Tesla K20X GPU accelerator with 6 GB GDDR5 ECC memory. The CPU has access to 32 gigabytes of DDR3 memory, and is connected to a Cray Gemini interconnect through a hyper-transport 3 interface. Two computing nodes share one Cray Gemini high-speed interconnect router.

In the implementation, we used the NVIDIA Toolkit and SDK (nvcc compiler) for the GPGPU part. Concurrent execution on multi-core is achieved using OpenMP supported by PGI compiler. The inter-node communication for multi-CPU and multi-GPU is handled through MPI. Host code, including MPI and OpenMP libraries, was compiled with PGI compiler. All runs were conducted on a 64-bit Linux environment.

5.2. CPU/GPU load balance

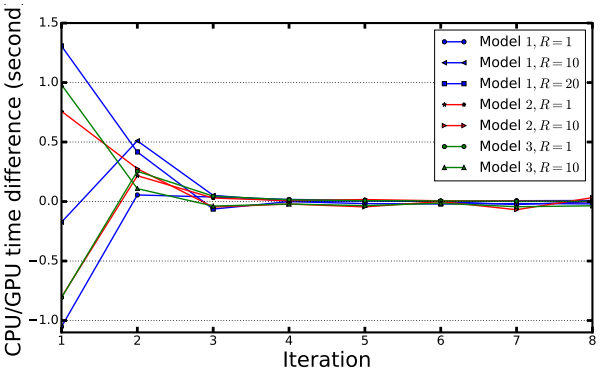
The heterogeneous balance scheme described in §3.5 has been tested with different models and different R. The total problem size is: $B_x = B_y = 200$, $B_z = 8000$ for all the models. As described in §3.5, tasks for CPU and GPU are split in the Oz direction. Figure 5a shows the convergence of time difference between CPU and GPU.

It is clear from Figure 5 that the balance scheme converges in less than 5 steps. Since the initial task ratio for CPU was set as 0.1, the runtime difference at the first step varies in different cases. As shown in Figure 5b, the CPU+GPU only benefits a little because the performance difference between CPU and GPU in our test computing node is pretty big (i.e., CPU undertakes very few tasks at the end). We believe that further benefit can be obtained on other supercomputing systems made of computing node with more powerful CPU.

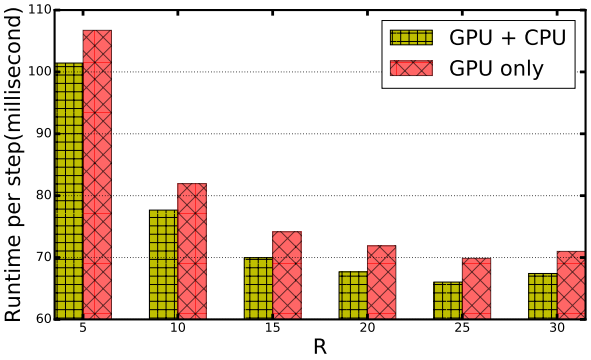
5.3. Analytical model validation

To validate our analytical model formulated in Equation 2 – Equation 6, we carried out six experiments with the heat diffusion model. The problem size are the same, namely, $1200 \times 1200 \times 1200$. The heat diffusion model with $\delta = \{1, 2, 3\}$ were run with different number of nodes: (a) 27 nodes, 3 in each dimension, and (b) 64 nodes, 4 in each dimension. Thus, subdomain sizes are different between (a) and (b). Results from experiments with $\delta = 2$ in (b) are used to fit the analytical model described in Equation 7. Then, with the estimated parameters, actual runtime and analytical model prediction were compared in Figure 6 on different experiment scenarios.

It is clear to see from Figure 6 that, our analytical model is not only general in nature, but also it is highly effective and extremely accurate to describe and predict the runtime complexity.

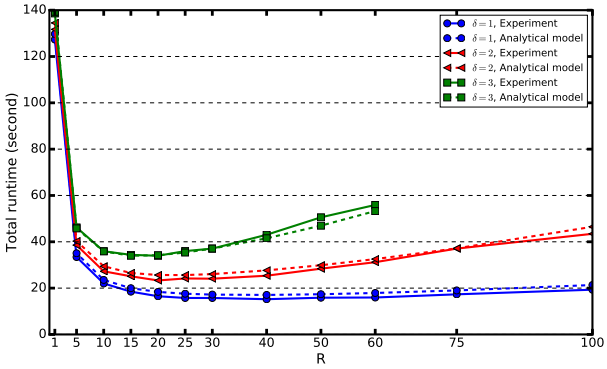


(a) Performance test of the automatic load balance scheme.

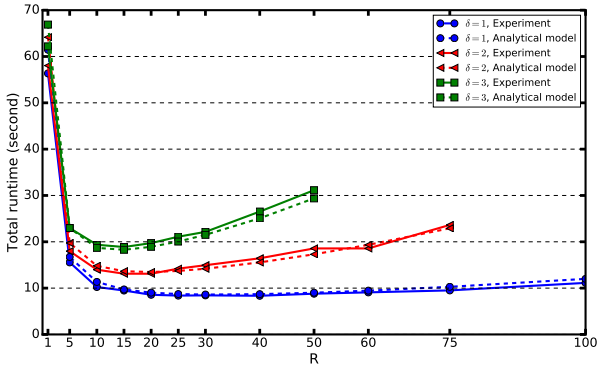


(b) Average runtime, GPU-only versus CPU+GPU.

Figure 5: Performance test of the automatic load balance scheme on different model. Different model represents different order of finite difference method for solving heat diffusion equation. In Figure 5a, the iteration is at node-level (Figure 1), i.e., one iteration denotes R simulation time-step. In Figure 5b, the vertical Y-axis represents runtime for updating one simulation times-step.



(a) Node = 27 ($3 \times 3 \times 3$).



(b) Node = 64 ($4 \times 4 \times 4$).

Figure 6: Total runtime, experiment results versus analytical model. The data in case with $\delta = 2$, Node = $4 \times 4 \times 4$ was used to fit parameter values. Problem size is $1200 \times 1200 \times 1200$, and executed 600 iterations for each run.

5.4. Speedup benefit

The runtime benefit of B2R is clear as shown in Figure 6. This section details the distribution of the total runtime and the speedup relative to $R = 1$ (which corresponds to an conventional implementation without our latency hiding scheme). We roughly divide the total runtime into three parts: inter-node communication for updating boundaries, memory transfer between host and device, and computation in the GPU device. The game of life model runtime distribution with different R was illustrated in Figure 7.

As shown in Figure 7, a dramatic reduction in communication time is observed even with a small increase in R. Although this results in a little longer computation time within a node relative to sequential execution, the total runtime was reduced significantly. However, there is an optimal R, after which the increased computing time dominates the communication time, i.e., B2R can not profit indefinitely.

To explore the benefit of B2R with different models, the speedup relative to $R = 1$ (no latency hiding scheme) was used as the performance metric. The speedup obtained with different R for heat diffusion model is shown in Figure 8. Here, the total problem size is $1200 \times 1200 \times 1200$, iterated 600 times in all testing scenarios. Figure 8a shows the speedup benefit in solving the heat diffusion model with different accuracy order of the finite difference method. Because of the condition that $R\delta \leq B$, $B = 400$ and 300 in 27 and 64 nodes scenario respectively, the range of R is restricted in big δ scenarios. Figure 8b shows the speedup benefit in solving the 3D game of life model.

It is clear that models with smaller δ (results in less computation but less accurate) can benefit more from B2R

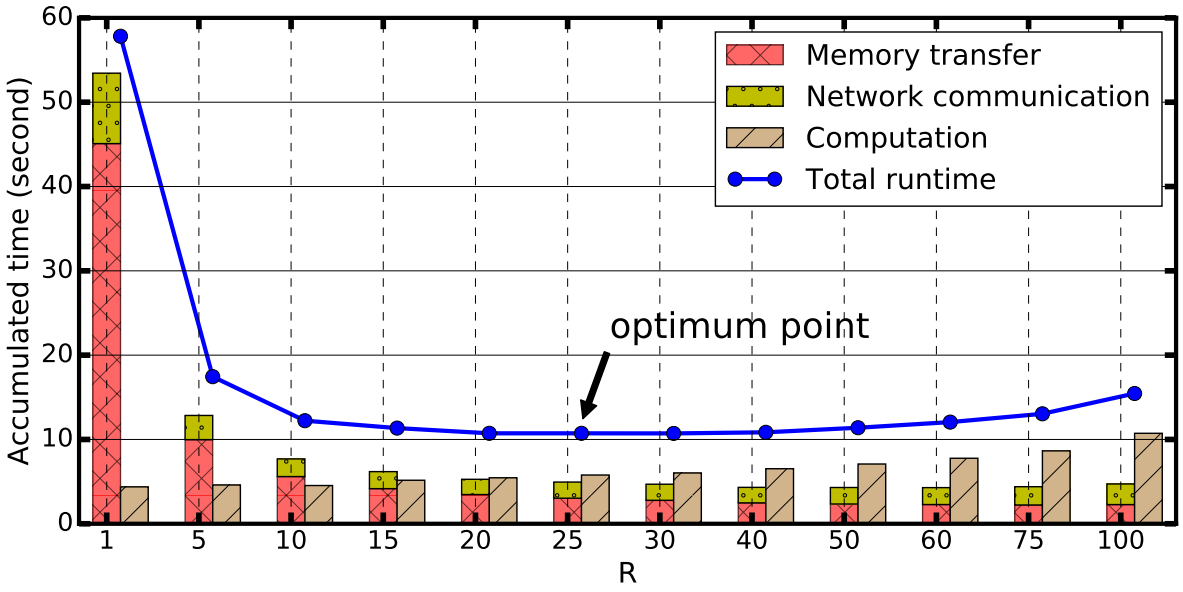
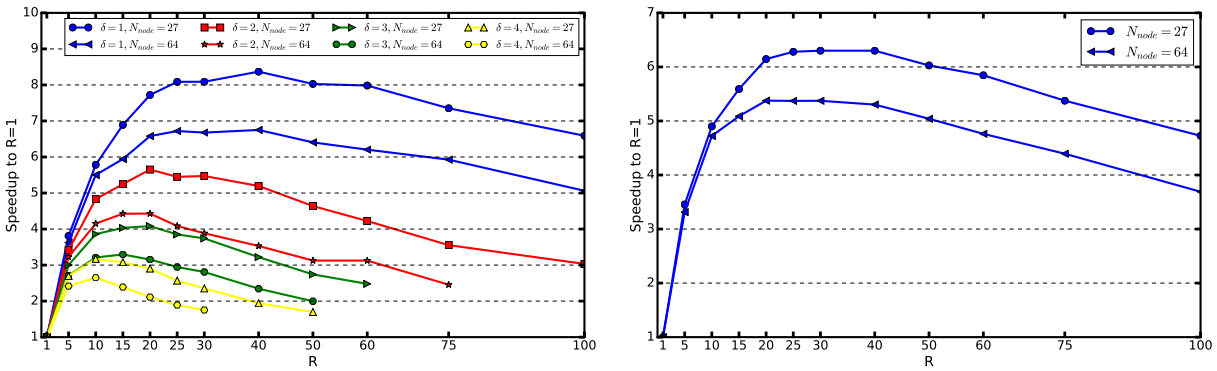


Figure 7: The proportion of communication time and computation time with different R.

latency hiding scheme, and each scenario has a significant optimum points (i.e., the best tradeoff between computation and communication). The speedup in Figure 8 clearly highlights the effectiveness of the B2R scheme.

For a given model, it is beneficial to decompose to more nodes to result in smaller subdomain size. Thus, when comparing the benefit between experiments with the same δ but different number of computing node, it is evident that bigger subdomain can benefit with more speedup from the latency hiding because each computing node holds a larger portion of data in local memory. However, when comparing Figure 8b with $\delta = 1$ in Figure 8a, although both scenarios require current state of neighbors within one layer, the game of life model benefits less. As analyzed in §4.2, this is because the game of life requires more memory access, which is more time-consuming than arithmetic operations [22]. That is to say the time ratio of *sync*/*computation* is smaller than that of the heat diffusion model. Figure 9 illustrates the runtime proportion and speedup benefit on solving the seismic wave propagation model.

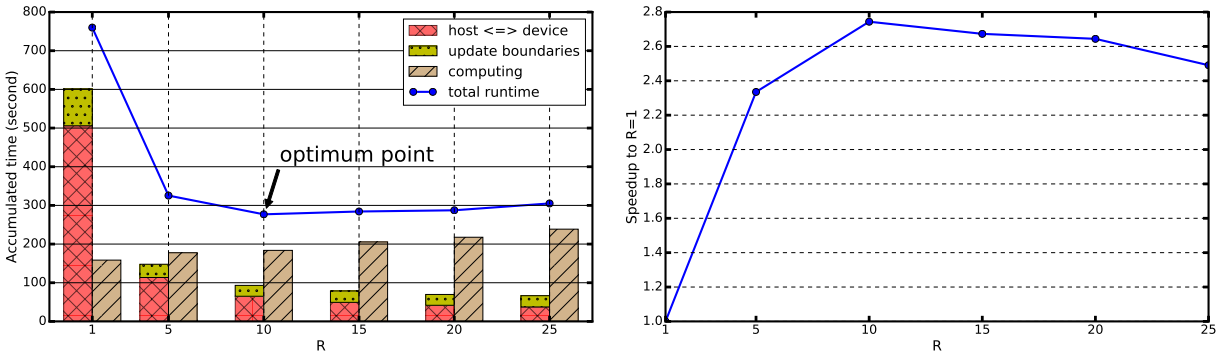
Comparing Figure 9a with Figure 7, it is clear that the communication to computation ratio of seismic wave propagation model is higher than that of the other two models. Although our B2R scheme is able to save a large



(a) Speedup versus R on solving heat diffusion model.

(b) Speedup versus R on 3D game of life model simulation.

Figure 8: Speedup versus R with different δ (Figure 8a) and subdomain size. The global grid size of the two models are $1200 \times 1200 \times 1200$, iterated for 600 times.



(a) Communication time versus computation time.

(b) Speedup versus R.

Figure 9: The proportion of communication time and computation time with different R and the speedup benefit versus R on solving seismic wave propagation model. The problem size is $3000 \times 200 \times 3000$ stencil points, it was decomposed to $25 = 5 \times 1 \times 5$ computing node.

amount of synchronization time by performing a little more computation, the final speedup benefit is not as good as the other two models. However, about 2.8x speedup without adding any hardware is still appealing on large-scale simulation.

In summary, the B2R scheme is able to save total runtime by synchronizing less frequently. The actual benefit is affected by the communication to computation ratio, i.e., time ratio of *sync*/*computation*. Relatively speaking, the computationally expensive models benefit slightly less from B2R than models with fine-grained grid cell computations.

5.5. *Large scale execution – strong & weak scaling*

To test the scalability of the B2R scheme, we conducted the following tests: (1) strong scaling test, which is defined as how the solution time varies with the number of processors for a fixed total problem size; (2) weak scaling test, which is defined as how the solution time varies with the number of processors for a fixed problem size per node. The 3D game of life model was utilized for the two tests. Figure 10 shows the results of strong scaling test.

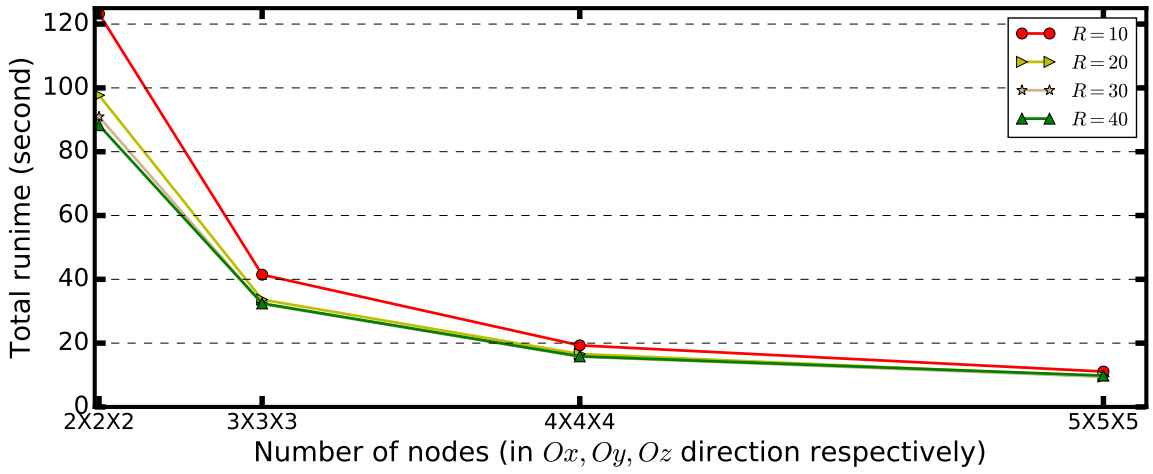
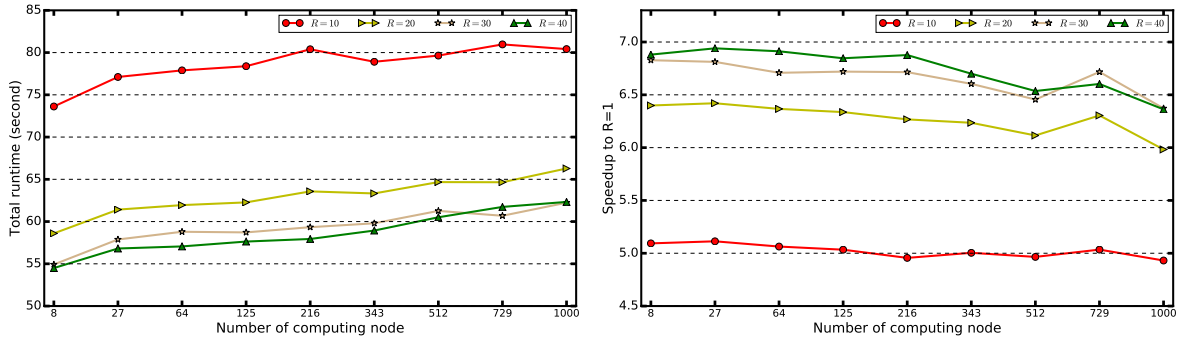


Figure 10: Strong scaling test of the B2R scheme. Total problem size is $1440 \times 1440 \times 1440$.

The strong scaling test was carried out with total problem size of $1440 \times 1440 \times 1440$, and with different R. As shown in Figure 10, it is clear that the total runtime decreased dramatically with increasing number of nodes, but not

as ideally as (theoretical) linearly to the number of nodes. This is because, as discussed in the analytical model and verified in Figure 8, the benefit of B2R also depends on subdomain size (B); less benefit is obtained with smaller subdomain size. The weak scaling test results are shown in Figure 11.



(a) Total runtime versus problem size.

(b) Speedup benefit versus problem size.

Figure 11: Weak scaling study of the B2R scheme. The problem size per computing node is $600 \times 600 \times 600$.

The weak scaling was conducted with a fixed problem size of $600 \times 600 \times 600$ per node. So, more computing nodes can carry out bigger total problem size. The last case scales to 1000 nodes (totaling 1,000 GPUs and 16,000 CPU cores) that simulates 216 billion cells. As total runtime shown in Figure 11a, due to synchronization between nodes, the runtime increased slightly with bigger total problem size. In addition to the speedup benefit of B2R, as shown in Figure 11b, the weak scaling also reaps the benefits of B2R scheme.

6. Summary and Future Work

Heterogeneous architectures are predominant in modern high-end computing platforms. Stencil computations and agent based simulations offer a lot of SIMD potential to exploit such architectures, but distributed memory across nodes needs algorithmic or hardware optimization to overcome high memory-memory transfers. In this paper, we analyzed a relaxation of the ghost layer technique to significantly improve multi-node performance via algorithmic optimization, and direct memory-memory transfers for optimized execution on specialized hardware. An analytical performance model was built for providing theoretical support to explain, predict and justify the optimal size of ghost zone and also to automatically split the task for CPU and GPU. The latency hiding scheme has been tested on multi-CPU and multi-GPU cluster, with a performance evaluation on 3-D models, variable stencil neighborhoods, and multiple application benchmarks. The implementations have been scaled to 1,000 GPUs and 16,000 CPU cores of a supercomputing system.

With only a few expectations, significant runtime benefit is observed from the generalized B2R scheme. Algorithmic speedup without the cost of adding custom hardware resources is attractive for simulation users because of reduction in runtime for large scientific simulations from several days to a single day.

ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research has been partially supported by the MINECO Spain, under contract TIN2014-53172-P, and a grant from the China Scholarship Council under reference number: 201306290023.

References

[1] Z. Li, Y. Song, Automatic Tiling of Iterative Stencil Loops, ACM Trans. Program. Lang. Syst. 26 (6) (2004) 975–1028. doi:10.1145/1034774.1034777.

[2] J. Meng, K. Skadron, Performance Modeling and Automatic Ghost Zone Optimization for Iterative Stencil Loops on GPUs, in: Proceedings of the 23rd International Conference on Supercomputing, ICS '09, ACM, New York, NY, USA, 2009, pp. 256–265. doi:10.1145/1542275.1542313.

[3] C. Lengauer, M. Bolten, R. D. Falgout, O. Schenk, Advanced Stencil-Code Engineering (Dagstuhl Seminar 15161), Dagstuhl Reports 5 (4) (2015) 56–75. doi:10.4230/DagRep.5.4.56.

[4] J. Zhou, Y. Cui, E. Poyraz, D. J. Choi, C. C. Guest, Multi-gpu implementation of a 3d finite difference time domain earthquake code on heterogeneous supercomputers, Procedia Computer Science 18 (2013) 1255 – 1264, 2013 International Conference on Computational Science. doi:10.1016/j.procs.2013.05.292.

[5] C. Ding, Y. He, A ghost cell expansion method for reducing communications in solving PDE problems, in: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, SC '01, ACM, New York, NY, USA, 2001, pp. 50–50. doi:10.1145/582034.582084.

[6] S. L. Lu, T. Karnik, G. Srinivasa, K. Y. Chao, D. Carmean, J. Held, Scaling the memory wall, in: Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on, 2012, pp. 271–272.

[7] A. Saulsbury, F. Pong, A. Nowatzyk, Missing the memory wall: The case for processor/memory integration, SIGARCH Comput. Archit. News 24 (2) (1996) 90–101. doi:10.1145/232974.232984.

[8] M. V. Wilkes, The memory wall and the cmos end-point, SIGARCH Comput. Archit. News 23 (4) (1995) 4–6. doi:10.1145/218864.218865.

[9] J. L. Hennessy, D. A. Patterson, Computer Architecture, Fifth Edition: A Quantitative Approach, 5th Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[10] B. G. Aaby, K. S. Perumalla, S. K. Seal, Efficient simulation of agent-based models on multi-gpu and multi-core clusters, in: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, SIMUTools 10, 2010, pp. 29:1–29:10. doi:10.4108/ICST.SIMUTOOLS2010.8822.

[11] J. Zhou, D. Unat, D. J. Choi, C. C. Guest, Y. Cui, Hands-on performance tuning of 3d finite difference earthquake simulation on {GPU} fermi chipset, Vol. 9, 2012, pp. 976 – 985, proceedings of the International Conference on Computational Science, {ICCS} 2012. doi:10.1016/j.procs.2012.04.104.

[12] Y. Cui, K. B. Olsen, T. H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. K. Panda, A. Chourasia, J. Levesque, S. M. Day, P. Maechling, Scalable earthquake simulation on petascale supercomputers, in: 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2010, pp. 1–20. doi:10.1109/SC.2010.45.

[13] Y. Hu, D. M. Koppelman, S. R. Brandt, F. Löffler, Model-driven auto-tuning of stencil computations on GPUs, in: A. Größlinger, H. Köstler (Eds.), Proceedings of the 2nd International Workshop on High-Performance Stencil Computations, Amsterdam, The Netherlands, 2015, pp. 1–8.

[14] T. Lutz, C. Fensch, M. Cole, Partans: An autotuning framework for stencil computation on multi-gpu systems, ACM Trans. Archit. Code Optim. 9 (4) (2013) 59:1–59:24. doi:10.1145/2400682.2400718.

[15] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K. Yelick, Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, in: High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, 2008, pp. 1–12. doi:10.1109/SC.2008.5222004.

[16] 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, ACM, New York, NY, USA, 2012, pp. 155–164. doi:10.1145/2259016.2259037.

[17] C. D. Cantrell, Modern mathematical methods for physicists and engineers, Cambridge University Press, 2000.

[18] H.-Q. Ding, Simulating lattice QCD on a caltech/JPL hypercube, International Journal of High Performance Computing Applications 5 (2) (1991) 74–81. doi:10.1177/109434209100500205.

[19] M. Gardner, Mathematical games: The fantastic combinations of john conway's new solitaire game "life", Scientific American 223 (4) (1970) 120–123.

[20] Y. Qin, Y. Wang, H. Takenaka, X. Zhang, Seismic ground motion amplification in a 3D sedimentary basin: the effect of the vertical velocity gradient, Journal of Geophysics and Engineering 9 (6) (2012) 761. doi:10.1088/1742-2132/9/6/761.

[21] P. Micikevicius, 3D finite difference computation on gpus using cuda, in: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-2, ACM, New York, NY, USA, 2009, pp. 79–84. doi:10.1145/1513895.1513905.

[22] C. Nvidia, Nvidia cuda c programming guide, Nvidia Corporation 1 (2015) 2–261.
URL http://docs.nvidia.com/cuda/cuda-c-programming-guide