

Communication savings with ghost cell expansion for domain decompositions of finite difference grids

C. Zambrana Rojas¹ and M. Hoemmen²

¹ Department of Civil Engineering, UCB, Berkeley CA 94704, USA *

² Computer Science Division, UCB, Berkeley CA 94704, USA †

May 14, 2004

Abstract

We investigate trade-offs between communications frequency and bandwidth, using the ghost cell expansion method (GCE) with domain decomposition for parallel evaluation of finite difference schemes for PDEs. GCE delays communication of boundary values between local domains for a small number of timesteps. This delay reduces communication pressure and aggregates messages into larger packets, thus optimizing for the high latency, high bandwidth interconnects common in modern parallel computing platforms. However, choosing the optimal ghost cell expansion level requires a careful balance between the cost of local computation per timestep, and network bandwidth (which may limit the effectiveness of increasing message size). Furthermore, for stencils wider than one cell on each side, GCE is an approximation which may impair numerical accuracy.

For now, we neglect numerical aspects of GCE. We use a LogP model with bandwidth to show that for the platforms tested, the technique only has potential for reducing the amortized cost of communication in the 1-D and 2-D cases. The model suggests a declining payoff for increasing expansion level in the 2-D case. Experimental results with a test problem verify some of these predictions.

1 Ghost cell expansion

Ghost cell expansion, proposed by Ding and He [6], is a form of domain decomposition with overlapping boundaries. Ding and He formulate it for finite difference methods used to solve PDEs with a timestep iteration. It accommodates either time-dependent PDEs (parabolic or hyperbolic) or elliptic problems

*E-Mail: zarcnek@newton.berkeley.edu

†E-Mail: mhoemmen@berkeley.edu

solved with a timestep-like iteration (e.g. Jacobi, Gauss-Seidel or ADI). For convenience’s sake, we refer to “timestep” in both cases.

Ding and He consider a regular rectangular mesh, partitioned into equal rectangles distributed one to a processor. (This description applies to 1-D and 3-D problems as well, though for clarity we explain the 2-D case.) Figure 4 illustrates one of these rectangular subdomains. The usual decomposition overlaps the subdomains by the width of one side of the finite difference stencil. For example, with a centered difference scheme, the stencil width is one cell, so the rectangles overlap by one. Each local domain also stores its *ghost cells*, the values of the neighbors’ adjoining cells which are in the numerical domain of dependence of the domain’s own cells. The width of this *ghost cell region* is that of one side of the stencil (in this case, one). Adjacent to the ghost cells in the local domain is a region of *mirror cells*, which correspond to the neighbors’ ghost cells. These regions are shown in Figure 5.

Communication is required after every timestep, when the mirror cells’ values change. In the communication phase, the local domain sends its mirror cells’ values to its neighbors’ ghost cells, and receives its neighbors’ mirror cells’ values into its own ghost cells. Figure 6 demonstrates this for a 1-D local domain, and Figures 7 and 8 for a 2-D local domain. (The 2-D and 3-D cases will be explained in detail in a later section.)

Ghost cell expansion widens the ghost cell regions (and thus the mirrors as well) by a small number of cells. This allows communication to be delayed over as many timesteps as the number of expansion cells. Boundary values are accumulated locally over more timesteps, and communicated in larger, but less frequent packets. The delay reduces the frequency of communication, at the cost of increasing the volume of data in a single transmission. This strategy exploits the relatively high overhead, high latency and high bandwidth networks that are common in recent parallel architectures. However, increasing the ghost cell region width, as well as the dimension of the PDE, will increase message volume, potentially taxing network bandwidth. We explore these compromises in the following.

1.1 The GCE algorithm

Ding and He err in their pseudocode description of GCE, although their diagram and subsequent descriptions indicate that they understand the intent of the technique. In particular, GCE should specify an expansion of the *update region*¹ to include the ghost cells. The update region then contracts by one cell on each side with every iteration, until at the last iteration before communication, it includes only the original update cells in the local domain. However, Ding and He’s pseudocode starts with the original update cells and moves inward from there. Since we implemented their pseudocode specification, our numerical results are tainted. However, Ding and He themselves point out that (for the 2-D case at least, which is the only case they analyze) the additional computation

¹Those cells in the local domain whose values are updated.

Listing 1: Schematic of GCE with an explicit time-dependent 1-D finite difference method

```

/* u[] array contains 2(L+e) + N elements */
start = L;
end = L + N + 2e;
t = 0;

for (n = 0; n < max_num_timesteps; n++)
{
    int gce_incr = n \% (e + 1);

    /* Communicate ghost cells once every e+1 timesteps */
    if (gce_incr == 0)
        exchange_ghost_cells (u);

    for (i = start - gce_incr; i < end + gce_incr; i++)
        /* Update value of u[i] using the stencil */
        do_stencil_update (u, i, delta_t, ...);

    /* Arbitrary timestep selection routine */
    delta_t = choose_timestep (delta_t, ...);
    t += delta_t;
}

```

required by the expansion of the update region is minimal. Thus, our comparison of computation and communication costs should be within reasonable bounds of accuracy. Furthermore, absolute communication costs are the same, regardless of which cells are updated.

For simplicity, we present the 1-D case. Suppose that the numerical domain of dependence of u_i^{n+1} includes u_{i-L}^n and u_{i+L}^n . We say then that the stencil width is L . Suppose the local domain includes N local compute cells. Without GCE, the local domain would need to store L ghost cells on each side, in an array of $N + 2L$ values. Assuming zero-based indexing, elements L to $N + L - 1$ would be updated at every timestep. With GCE, the array now stores $L + e$ ghost cells on each side, where e is the ghost cell expansion level. Ghost cell exchanges occur once every $e + 1$ timesteps. The program listing 1 shows an example application of GCE to an explicit time-dependent 1-D finite difference method.

1.2 GCE as a numerical approximation

If the stencil width is at most 1, GCE does not alter the numerical properties of the finite difference method. With every iteration, the numerical domain of dependence shrinks inward, thus avoiding computation with stale values. Every

$e+1$ timesteps, the stale values are updated with fresh ones. As long as the final timestep occurs immediately before this update, and the results are reported after the update, the output is identical to that of the original difference scheme.

However, if the stencil width L is 2 or more (on either side), updates near the boundary will use “stale” (not up-to-date) ghost cell values, since the numerical domain of dependence only moves inward by one cell every iteration. The domain of dependence must move inward by L cells per iteration, in order to avoid using stale ghost cells in the stencil update.²

Ding and He claim that GCE is a variant of additive Schwarz domain decomposition with overlapping local domains, where the amount of overlap is $e+1$. This statement is only completely accurate if their technique is applied to steady-state problems. Although they claim to be solving time-dependent problems, the authors only apply GCE to an elliptic test case. H. A. Schwarz’s original 1870 formulation of domain decomposition defined an iteration for elliptic PDEs, and subsequent domain decomposition techniques have focused on elliptic problems. However, variants for hyperbolic PDEs have recently emerged. Wu et al. present a variational formulation of the transport equation which is amenable to additive Schwarz (AS) domain decomposition [12]. They first discretize implicitly in time, and then use AS for the resulting spatial first-order PDE. Their PETSc-based[1] parallel implementation solves the transport equation.

GCE could be considered as a version of additive Schwarz in which only one iteration is taken at every timestep. In particular, it reduces any finite difference stencil to first order in space at the interfaces. Intuitively, the use of an implicit scheme would “spread out” the first-order error throughout the local domain, whereas explicit methods would limit the error’s propagation speed. More sophisticated techniques for limiting or preventing error propagation, such as wavefront tracking, are generally ill-suited to the regular grids and simpler problems for which GCE is intended.

Unfortunately, Ding and He do not quantify the numerical consistency and stability of their approach, or show experimental convergence results. The convergence slowdown resulting from the ghost cell method could potentially negate its performance gains. Unfortunately, time limitations prevent us from addressing this question in more detail.

2 Ghost cell exchange algorithms

The communication phase of GCE consists of each local domain updating its neighbors’ ghost cells with its corresponding boundary values. Palmer and Nieplocha present several update algorithms for regular, rectangular grids, for both shared- and distributed-memory parallel architectures [9]. Edge and corner cells present the biggest challenge for an efficient update. The naive method, which Palmer and Nieplocha call *Put*, simply sends mirrors to all neighboring ghost regions, in a single communication phase. A rectangular domain in the

²Ding and He do not discuss this straightforward extension of the GCE algorithm.

middle of a 2-D grid has eight neighbors; a separate message must be sent to each. In the general d -dimensional case, all but the boundary processors must send $3^d - 1$ messages. Furthermore, 2^d of these messages are for corners, which contain two orders' fewer cells.

Ding and He describe an optimization, which they call *diagonal communication elimination*³, for reducing the number of messages in the exchange operation. While it sends some edge and corner values redundantly, it reduces the total number of messages from $3^d - 1$ to $2d$. This is a natural optimization for high-latency, higher-bandwidth machines. Furthermore, it vastly simplifies the message-passing code: When using MPI, only $d - 1$ MPI datatypes are needed.

Diagonal communication elimination, which Palmer and Nieplocha classify as a “shift” scheme, separates communication into d stages. All processes must complete each stage before the next may begin. The two stages of the 2-D case are illustrated in Figures 7 and 8. First, each processor sends its north and south mirror regions to the corresponding neighboring local domains. When that step is complete, then each processor sends its west and east mirrors, along with the ghost cells in the same column (which it had received from its north and south neighbors), to its west or east neighbors, respectively. The effect is that corner ghost cells are communicated diagonally, even though no explicit communication with diagonal neighbors takes place. This technique extends in an obvious way to the d -dimensional case.

One disadvantage of “shift” methods is that they require some form of synchronization between neighbors. This is because communication of diagonally adjacent local domains depends on the correct ordering of messages. Synchronization may be implicit, as with the use of blocking sends and receives, which Palmer labels `M_P`. Alternately, it may involve some explicit synchronization operations, such as flags or global barriers. Palmer and Nieplocha compare several implementations, some optimized for shared memory, some for longer-latency distributed memory, and even hybrids that would be natural on clusters of SMPs. They also consider `Put`. The authors find that `M_P` tends to outperform any other technique. Even on a platform such as the Cray T3E, with a very low-latency network and hardware support for global synchronization, their experiments show that `M_P` performs nearly as well as `Put` (the best algorithm on that machine). On an earlier IBM SP, which like Seaborg is also a cluster of SMPs, `M_P` outdoes even the version specifically optimized for such clusters. We also chose to implement the exchange using `M_P`.

Despite the advantages of a “shift”-based ghost cell exchange, the required synchronization between stages (in the 2-D and 3-D cases) precludes the potential optimization of overlapping communication with computation. Palmer and Nieplocha do not consider this possibility. As long as mirrors are buffered and stencil updates begin on the portion of the grid outside the ghost cells' numerical domain of dependence, the updates could begin during the ghost cell exchange. In fact, given sufficient memory, several timesteps could be calcu-

³The technique was originally developed by Ding in [5].

lated. At each timestep, these computations would move inwards to avoid the ghost cells’ domain of dependence. However, this optimization would only pay off if the local domain has enough cells that computation costs are greater than communication. Otherwise, stencil updates would need to stop eventually and wait for the boundary values to arrive. The next stage of communication could begin only after the boundary values have been updated.

3 Test platforms

We had access to three different test platforms: “Alvarez,” “Citris” and “Seaborg.” Alvarez (see [8]) is cluster of 80 nodes connected with Myrinet 2000, which has two network ports per node. Each compute node is an IBM xSeries 330, which is a two-way SMP 866 MHz Pentium III with 1GB of SDRAM. It runs an SMP version of GNU/Linux kernel 2.4.20-24.7. Seaborg (see [7]) is an IBM SP RS/6000 cluster with 6080 processors, arranged as 16-way SMP nodes. Each processor is a Nighthawk-class 375 MHz POWER3. Nodes are linked together with an IBM Colony / GX switch that provides two ports per 16-way node. Citris (see [4]) is an Intel Itanium 2 cluster of 64 two-way SMP nodes, which run an SMP variant of GNU/Linux kernel 2.4.20. Some of the processors are McKinley class (900 MHz), and some are Madison class (1.3 GHz). Although a Myrinet 2000 network is available in hardware, the software does not currently work, so the cluster must rely on 100 Base-T Ethernet (with one port per node). This is the classical “Beowulf cluster” configuration. Ethernet has considerably longer end-to-end latency as well as receive overhead than either Myrinet or the Colony / GX switch. We suspect that this property will weight Citris to favor algorithms that reduce the total number of messages.

4 Communication model

The LogP model of communication (see [3]) uses four parameters to predict communication costs in a message-passing paradigm: the network latency L , the processor overhead o (sometimes split into send overhead o_{send} and receive overhead o_{recv}) of sending or receiving a message, the “gap” g (minimum time between initiations of a send or receive) and the number of processors P .

Using the M_P method of exchanging ghost cells (“shift” with blocking send and receive), each dimension requires two messages. In the LogP model, a processor requires o_{send} overhead to send the first message. Besides some index calculation for the next send, there is no useful computation it can perform between sends, so it must wait for the entire gap g to pass before it can send the second message. So the second message is sent at time $g + o_{send}$. If we assume $o_{recv} \leq o_{send}$ and L is sufficiently long, then the second message is received at $g + o_{send} + L + o_{recv}$, and the recipient processor is ready for the next stage of communication at time $2g + o_{send} + L$.

Alternately, we may combine o_{send} , L and o_{recv} together as “end-to-end la-

Dimension	Number of stages	Estimated cost
1	1	$g + EEL$
2	2	$3g + 2EEL - o_{recv}$
3	3	$5g + 3EEL - 2o_{recv}$

Table 1: Communication costs for each dimension, estimated using the LogP model.

Platform	Network type	EEL (μs)	Gap (μs)	o_{recv}	BW (MB/s)
Alvarez	Myrinet 2000	12	17.8	0	88
Citris	100 Mb/s Ethernet	30-90	?	?	12-20+
Seaborg	Colony/GX	19.5	7.6	5.4	242

Table 2: LogP parameters for each network type, as measured experimentally by Bell et al. [2] Note: Myrinet 2000 parameters were measured on the Millennium x86 network, which has a configuration similar to Alvarez. Benchmarking on Seaborg used MPI (rather than the native LAPI library), and on the Myrinet 2000 platform, the GM-based implementation of MPICH was used. While Citris is equipped with Myrinet 2000 hardware, the network is currently non-functional. 100 Mb/s Fast Ethernet performance depends heavily upon the software layer, so we give a range of values found by measurements on other systems.

tency” (EEL), which more accurately reflects some networks’ ability to overlap send and receive overheads (so that $EEL < o_{send} + L + o_{recv}$). Bell et al. advocate this strategy in their evaluation of current parallel architectures [2]. In their model, the recipient processor finishes receiving the second message at $g + EEL$ and is ready for communication by $2g + EEL - o_{recv}$. Table 1 reviews the estimated costs of communication in the 1-D, 2-D and 3-D cases.

We use a simple “ $\alpha + n\beta$ ” model to estimate end-to-end latency. α is the short-message end-to-end latency EEL , β is the inverse of the bandwidth and n is the number of units (bytes or words, depending upon the units of β) in the message. Bell et al. measure these quantities experimentally, and their values are reported in Table 2. The receive overhead for 100 Base-T Ethernet tends to be high, but without more detailed data, we must limit our predictions to the other two platforms.

4.1 Communication volume and possible GCE benefit

Message size scales roughly with $N^{d-1}(L + e)$, where N is the side length of a square local domain, L the stencil width and e the GCE expansion level. Formulae for exact volume of communication per message for each of the 1-D, 2-D and 3-D cases are shown in Table 3. Combining this information with the LogP estimates in Table 1 and multiplying by the average number of exchanges

Dimension	Message size (words)
1	$D(L + e)$
2	$D(2(L + e) + N)(L + e)$
3	$D^2(2(L + e) + N)^2(L + e)$

Table 3: Volume of communication per message, in bytes. D is the number of bytes per data word, L is the stencil width, and e is the ghost cell expansion level. We assume that each local domain has a side length of N cells, for each problem dimension.

per timestep ($1/(e + 1)$) results in an estimate of the amortized communication time per timestep. Using these estimates, we can project the potential benefits of different levels of ghost cell expansion. Figures 1, 2 and 3 show the 1-D, 2-D and 3-D cases, respectively.

For 1-D, we see that an expansion level of 8 improves communication time tenfold. However, timings for a level larger than one are within the end-to-end latencies of all the networks tested. This suggests that noise factors such as variance in network performance (a common and complex phenomenon) may be of the same magnitude as performance improvements. The 2-D performance estimates suggest that the gains due to increasing expansion level drop off quickly. Still, a GCE level of 1 or 2 may save more per timestep than the cost of a single small message. These savings should accumulate into significant gains over all the time iterations.

The 3-D case confirms the intuition that ghost cell regions in 3-D scale too fast with the problem size for GCE to pay off. In fact, according to the model, the local domain size must be reduced to $8 \times 8 \times 8$ cells for communication time on Seaborg to benefit at all from GCE, and it must be reduced to 2×2 cells for timings on Alvarez to improve. In both cases, only an expansion level of one improves LogP-modeled communication time.

5 Model problem and finite difference scheme

We choose a hyperbolic model problem, Problem 1. Ding and He already examine the elliptic case, and finite difference schemes tend to be applied to hyperbolic PDEs more often than to elliptic or parabolic PDEs. Multigrid smoothers are a notable exception. For our model, we select $f(x) = 0$, $c = 1$ and $\Omega = (0, 1)^d$ (where d is the problem dimension), and discretize the domain uniformly in each dimension: $x_j = j\Delta x$.

Problem 1. *Two-sided linear wave equation in d dimensions, with homogeneous Dirichlet boundaries.*

$$\begin{aligned}
 \dot{u} &= c^2 u_{,ii} + f(x) && \text{in } \Omega, \\
 u|_{t=0} &= g(x) && \text{on } \partial\Omega, \\
 g|_{\partial\Omega} &= 0.
 \end{aligned}$$

Dimension	Cells per local domain	Number of timesteps
1	2^{10}	100
1	2^{20}	100
2	$2^{10} \times 2^{10}$	100
3	$2^6 \times 2^6 \times 2^6$	25
3	$2^6 \times 2^6 \times 2^6$	100

Table 4: Test problem sizes for each dimension.

For convenience, we require that the initial state $g(x)$ satisfies the Dirichlet boundary condition also.

Problem 1 admits an explicit centered-in-time, centered-in-space (CTCS) discretization (5.1) (shown in the 1-D case for simplicity). Although a more sophisticated scheme might be used in practice, CTCS represents the minimal amount of local computation that a reasonable explicit difference method for hyperbolic problems would perform. We choose the timestep size Δt to satisfy the CFL condition $R \leq 1$.

$$\begin{aligned}
 u_i^{n+1} &= 2u_i^n - u_i^{n-1} + R(u_{i+1}^n - 2u_i^n + u_{i-1}^n) + \Delta t f_i \\
 R &= \frac{c^2 \Delta t^2}{\Delta x^2}.
 \end{aligned}
 \tag{5.1}$$

5.1 Test cases

Model problem sizes for each dimension are shown in Table 4. Concern for overloading a busy queue guided the limited size of the 3-D case. Nevertheless, the 3-D solver requires a seven-point stencil, as compared to a two-point stencil in the 1-D case, so the total amount of computation is comparable. For 2-D and 3-D, we arranged the rectangular or cubical local domains, respectively, in order to minimize inter-node communication, while keeping the numbers of local domains in each direction within a power of 2 of one another. For example, the 32-domain grid in 3-D has 4 domains in the x and y directions, and 2 in the z direction.

6 Experimental results

It appears that GCE does decrease overall communication time in some cases. On Seaborg, we observed a clear trend for 1-D and 2-D that communications costs decline as the GCE expansion level increases. See Figures 15 and 16. The 1-D case on the Citris platform also benefits from GCE (see Figure 12), though the 2-D case shows only a slight improvement. Alvarez timings were so variable that it is difficult to extrapolate a trend; if anything, communication time appears to increase with expansion level, for 1-D, 2-D and 3-D. In general, as the LogP model predicts, GCE is not beneficial in 3-D.

6.1 1-D details

As predicted by the LogP model, adding more ghost cell layers tends to improve communication time. However, the payoff is rarely monotonic. For example, on Alvarez with 8 or 32 processors (see Figure 18), no GCE is better than using 1 or 2 layers. On Citris (see Figure 19), if we throw out the 2 and 4 processor cases, there is little benefit from any expansion. A level of 8 on Seaborg tends to be the best (see Figure 20), but the relationship is highly nonlinear. For example, on 32 processors (two 16-way nodes), using no GCE improves communication time by a factor of 8 over an expansion of 1 cell, but expanding the ghost layer by 8 cells improves the time by a factor of 8 over the no GCE case.

The lack of a monotonic relationship means that achieving any communications savings at all may require a possibly unacceptable loss of numerical accuracy. This may limit the use of GCE to width-1 stencils, where delaying communication causes no accuracy loss.

6.2 2-D details

GCE benefits for the 2-D case tend to support the LogP model's prediction, that some expansion pays off, but the benefits may decrease as the expansion level increases. However, variation in results hinders reliable prediction. For example, on Seaborg (see Figure 23, at least one of the GCE expansion levels tested outperforms the no GCE case, for any number of processors tested. However, for four and eight processors, a level of 1 outdoes the others, whereas for 16 or 32 processors (the latter case involving remote communication), a level of 8 is best. For Citris (see Figure 22, a level of 8 is always the best or nearly the best, for all numbers of processors examined. Timings varied significantly on Alvarez (see Figure 21, but some amount of expansion between 1 and 4 generally improves communication time. An expansion of 8 causes little benefit, or even increases communication time in the 32 processor case.

Although these results are less severely non-monotonic than in the 1-D case, there is no obviously best expansion level, even for a particular platform. Still, depending upon the application, the possible savings justify a look at GCE.

6.3 3-D details

Although the model predicts no benefit from GCE, timings on Alvarez show surprising speedups. On 8 processors, using an expansion level of 8 gives a sixteen-fold reduction in total communication time, over the level-0 case (see Figure 24). Using some level of expansion generally pays off, even though there is no clear monotonic relationship between the optimal amount of expansion and the number of processors. In contrast, timings on Seaborg and Citris generally do not benefit from GCE, as Figures 25 and 26 show.

Colleagues have pointed out that timings on Alvarez tend to be quite variable, as our experiments also suggest. The difference between results tends to be much higher than on the other two platforms, even though the effects of op-

timizations should stand out much more on the much higher-latency Ethernet-connected Citris machine, or the higher-bandwidth Seaborg cluster.

7 Conclusions and future work

Determining an optimal GCE level is more difficult than simply observing an improvement. The optimal level depends on static and dynamic network parameters, as well as the stencil width and the number of local compute cells. Timings vary significantly from one run to the next. Nevertheless, our experiments suggest that for 1-D and 2-D problems, GCE may improve communication times significantly. In 1-D, if we ignore the potentially suspect Alvarez timings, an expansion level of at least 8 is generally the best, or in the worst case does not significantly increase communication time. More investigation is needed in the 2-D case. In 3-D, as the model predicts, GCE is generally not beneficial.

For stencils of width 1, expanding the ghost layer has no effect on numerical accuracy. For wider stencils, the required expansion level to achieve the same benefits must be scaled by the stencil width. Alternately, one may choose to retain the original GCE algorithm as defined by Ding and He. In the time-dependent case, this reduces the finite difference method to first order at the local domain interfaces. For elliptic problems, ghost cell expansion is a form of additive Schwarz domain decomposition. If one is using a stencil-based iteration (like Jacobi or Gauss-Seidel) to solve the elliptic problem, convergence will be slow anyway, and using additive Schwarz will probably degrade it further. However, one usually uses Jacobi-like iterations as a smoother for multigrid. There, the convergence rate of the smoother is less significant, and some of the iterations used have stencil width 1.

In the parallel multigrid context, techniques similar to GCE fall under the category of “sparse tiling,” and are an area of active investigation; see for example [10]. In the compiler literature, the method is called “time skewing”: see e.g. [11]. Combining Ding and He’s original idea with these more mature and generalized optimizations could benefit both multigrid, and finite difference methods for time-dependent problems. We plan to investigate these relationships further.

8 Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. Some of the computational work for this project was done on the Lawrence Berkeley National Laboratory (LBL) / NERSC Alvarez Cluster (described in this document). We would especially like to thank David Skinner, who was administrating the cluster during our project.

The Itanium 2 cluster also used in our work is part of the CITRIS (Center for Information Technology Research in the Interest of Society) Project, a collabo-

ration of several University of California campuses. The cluster is managed by the Electrical Engineering and Computer Science department of the University of California, Berkeley.

We would like to thank Professors Katherine Yelick and James Demmel for suggesting this research topic and pointing out its relationship to tiling techniques in multigrid.

LogP-estimated effects of GCE expansion on 1-D communication time

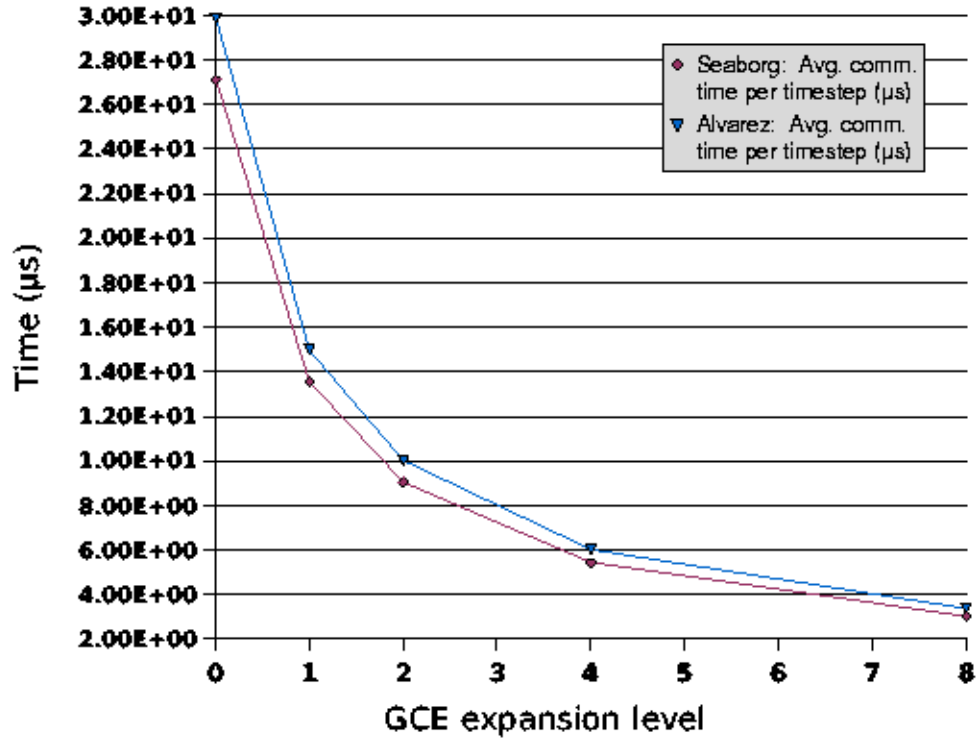


Figure 1: LogP-based estimate of amortized communication time (μs) per timestep, as a function of the ghost cell expansion level. We assume a 1-D local domain with 2^{20} cells.

LogP-estimated effects of GCE expansion on 2-D communication time

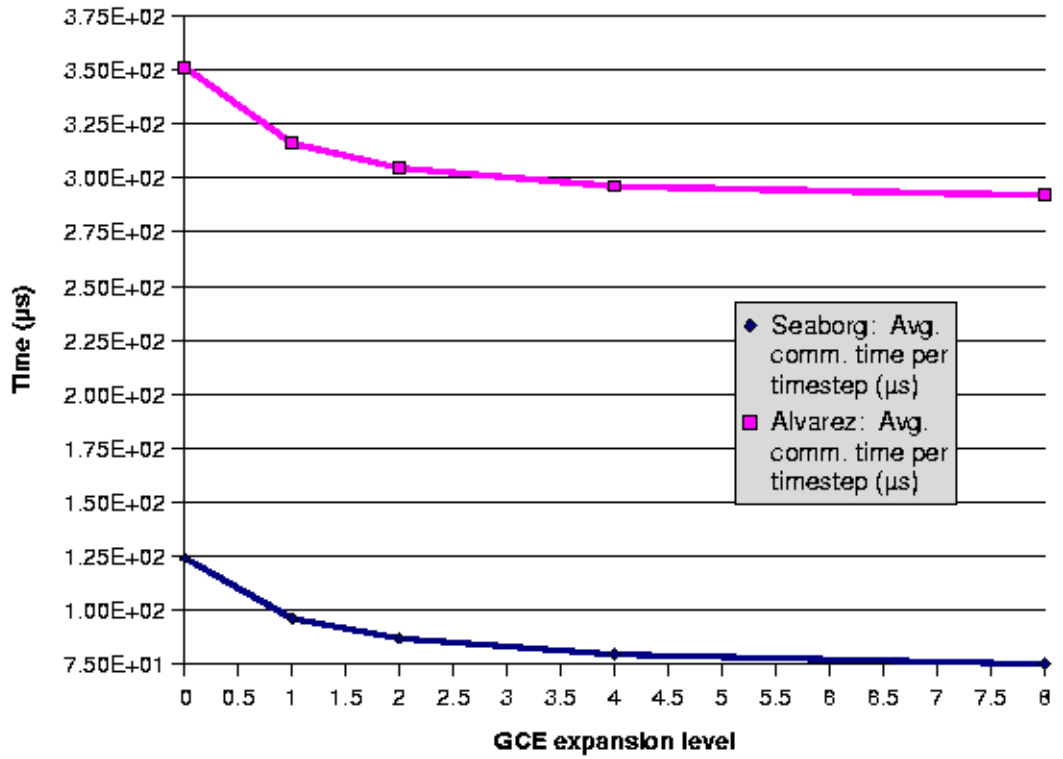


Figure 2: LogP-based estimate of amortized communication time (μs) per timestep, as a function of the ghost cell expansion level. We assume a 2-D local domain with $2^{10} \times 2^{10}$ cells.

LogP-estimated effects of GCE expansion on 3-D communication time

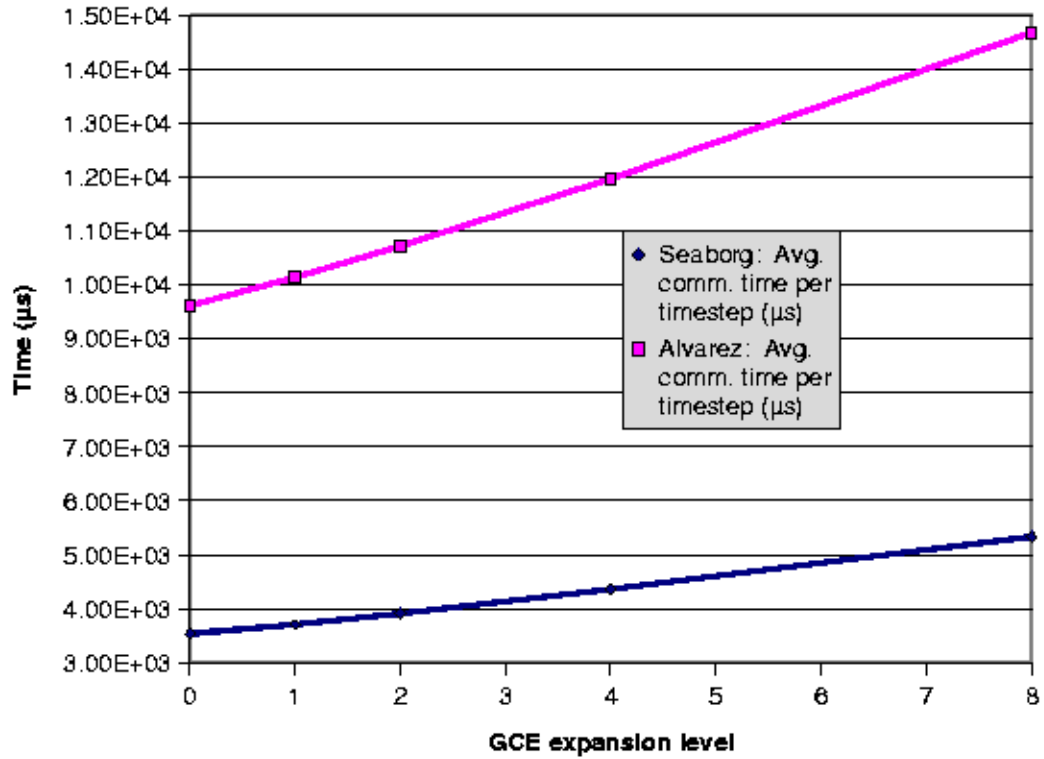


Figure 3: LogP-based estimate of amortized communication time (μ s) per timestep, as a function of the ghost cell expansion level. We assume a 3-D local domain with $64 \times 64 \times 64$ cells.

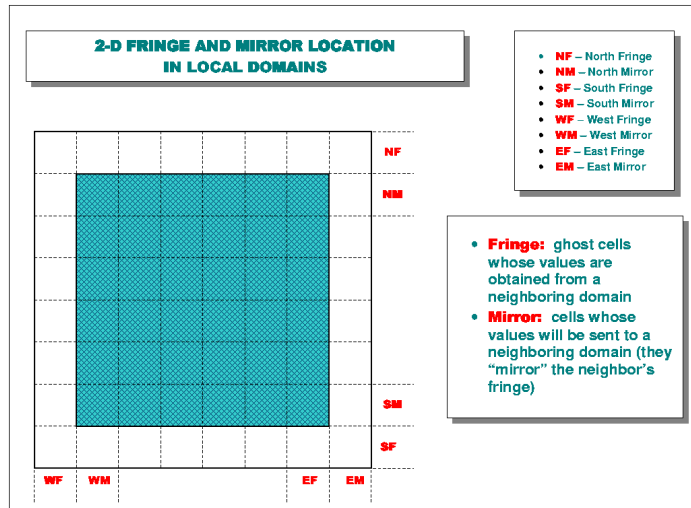


Figure 4: Illustration of a 2-D local domain. For simplicity, it is shown as a square, though our implementation allows general rectangles.

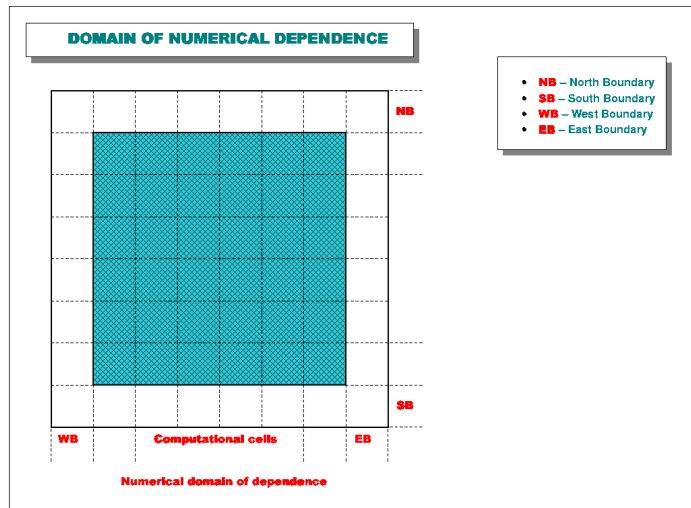


Figure 5: Ghost and mirror regions of a 2-D local domain.

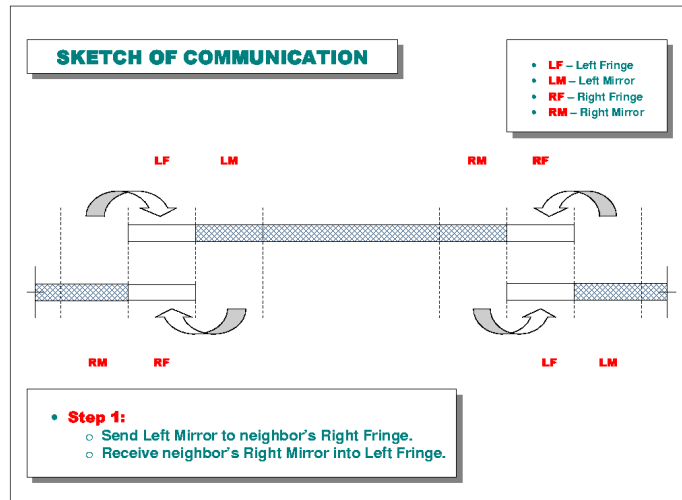


Figure 6: Communication of ghost and mirror regions with neighboring 1-D local domains.

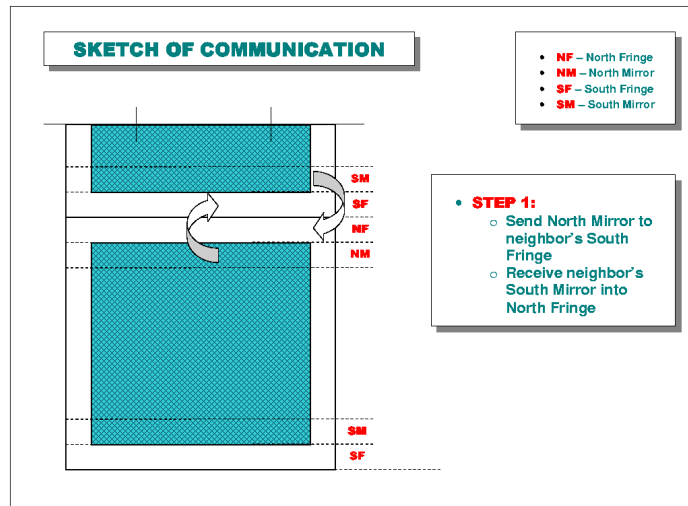


Figure 7: Communication of ghost and mirror regions with neighboring 2-D local domains: Stage 1.

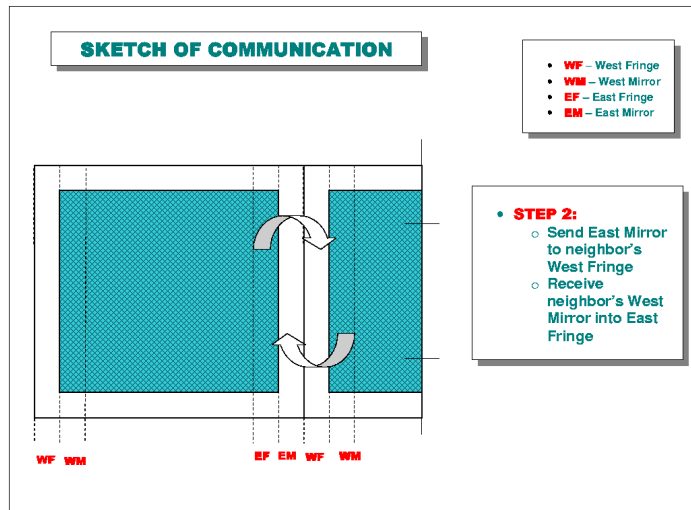


Figure 8: Communication of ghost and mirror regions with neighboring 2-D local domains: Stage 2.

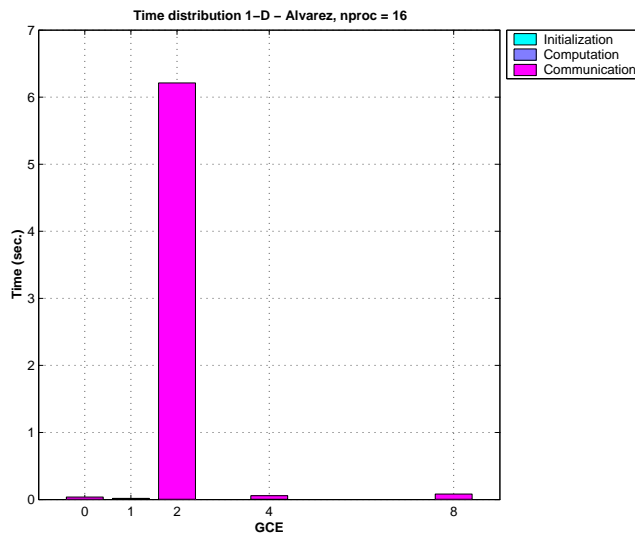


Figure 9: Time distribution for 16-processor 1-D test run on Alvarez. Each local domain consists of 1024 cells.

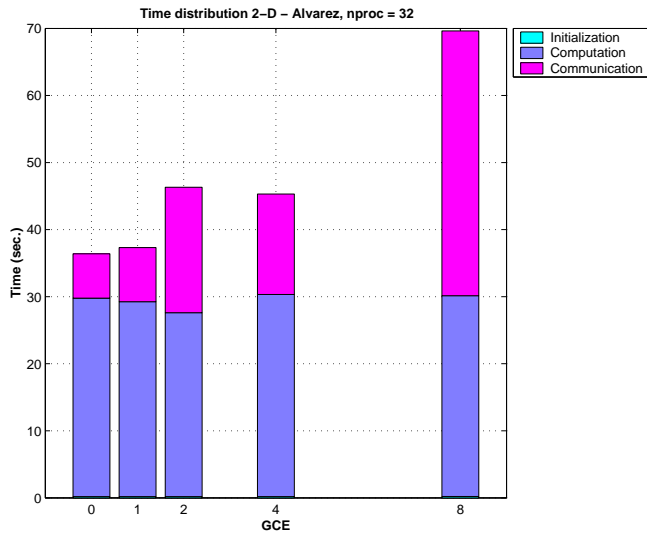


Figure 10: Time distribution for 32-processor 2-D test run on Alvarez. Each local domain consists of 1024×1024 cells.

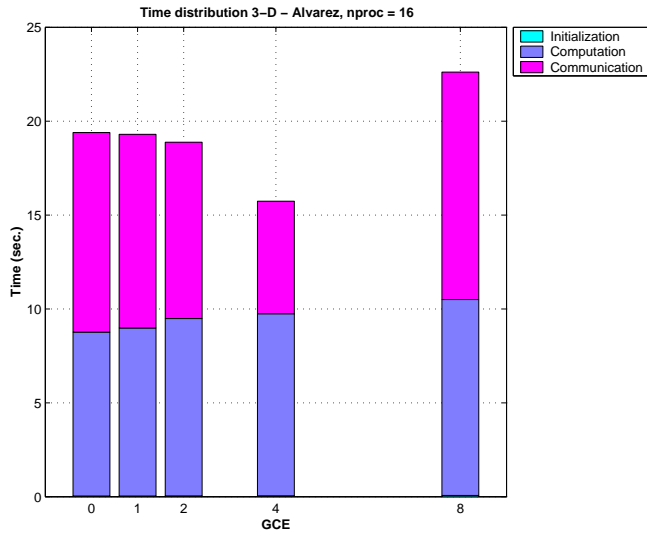


Figure 11: Time distribution for 16-processor 3-D test run on Alvarez. Each local domain consists of $64 \times 64 \times 64$ cells. Timings taken for 25 timesteps.

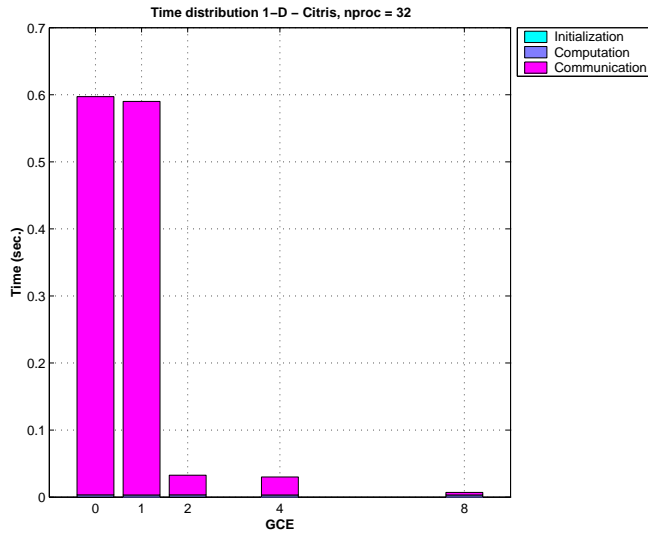


Figure 12: Time distribution for 32-processor 1-D test run on Citris. Each local domain consists of 1024 cells.

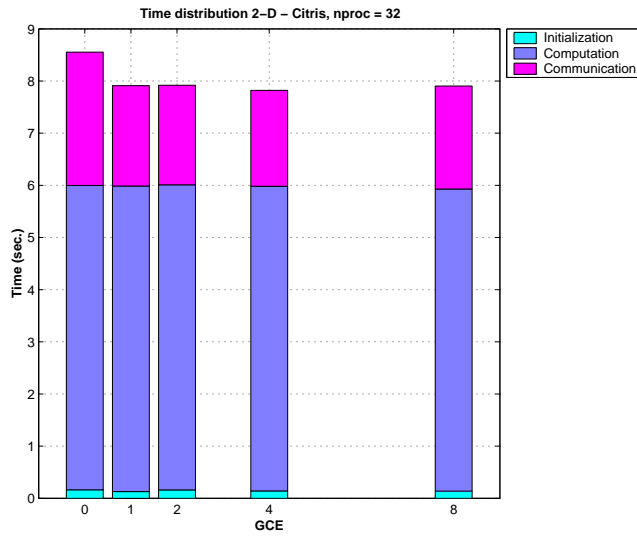


Figure 13: Time distribution for 32-processor 2-D test run on Citris. Each local domain consists of 1024×1024 cells.

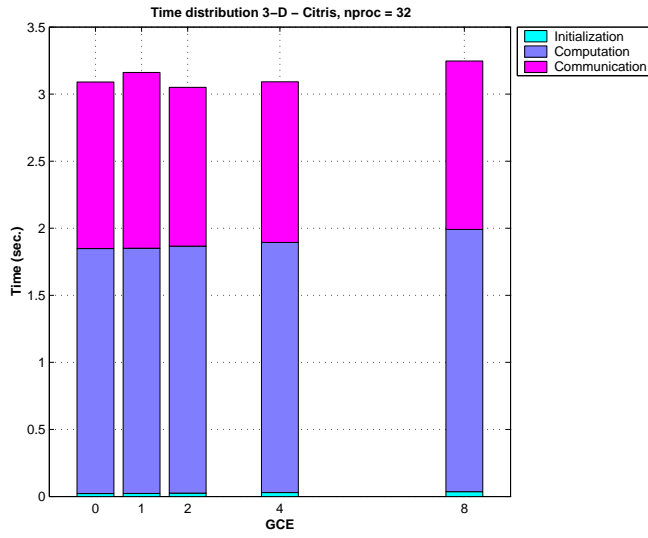


Figure 14: Time distribution for 32-processor 3-D test run on Citris. Each local domain consists of $64 \times 64 \times 64$ cells. Timings taken for 25 timesteps.

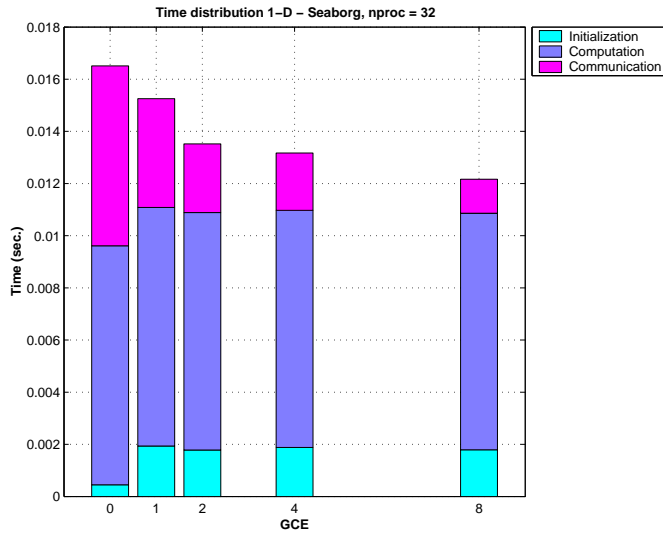


Figure 15: Time distribution for 32-processor 1-D test run on Seaborg. Each local domain consists of 1024 cells.

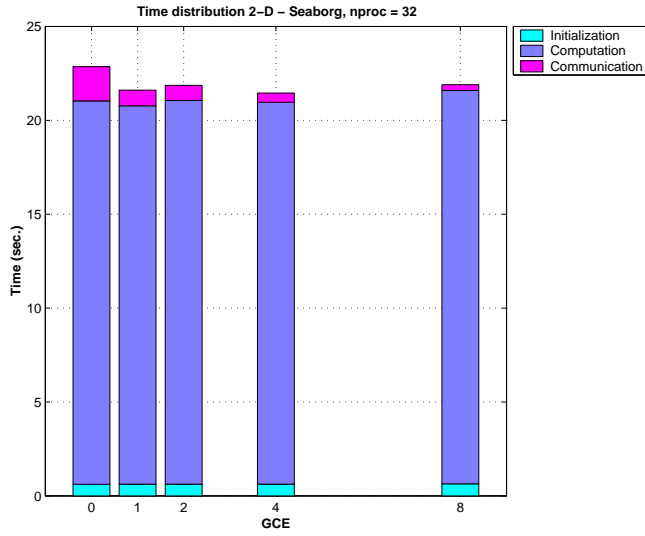


Figure 16: Time distribution for 32-processor 2-D test run on Seaborg. Each local domain consists of 1024×1024 cells.

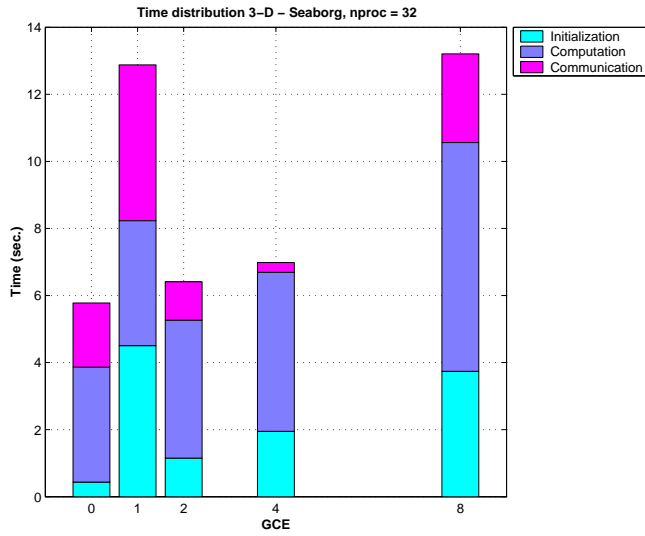


Figure 17: Time distribution for 32-processor 3-D test run on Seaborg. Each local domain consists of $64 \times 64 \times 64$ cells. Timings taken for 25 timesteps.

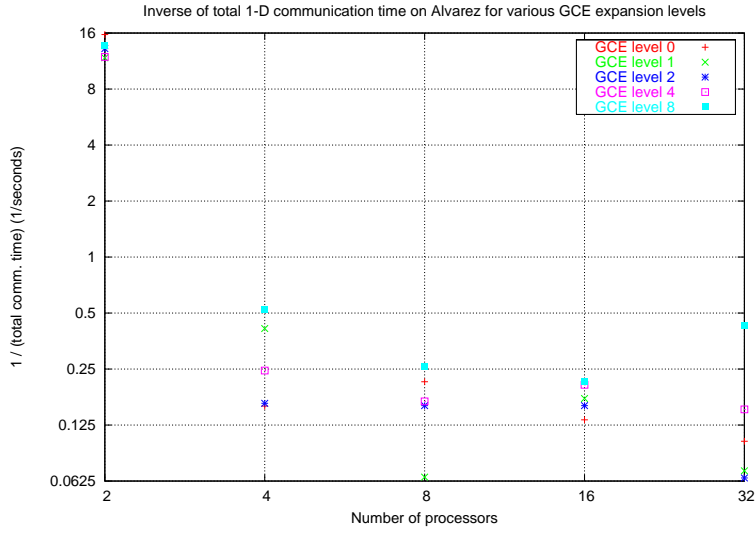


Figure 18: Inverse of total 1-D communication time on Alvarez for various GCE expansion levels. Each local domain consists of 2^{20} cells.

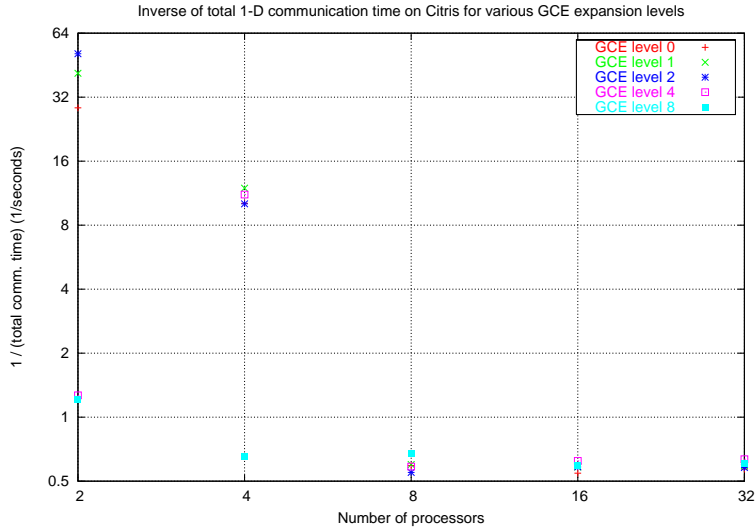


Figure 19: Inverse of total 1-D communication time on Citris for various GCE expansion levels. Each local domain consists of 2^{20} cells.

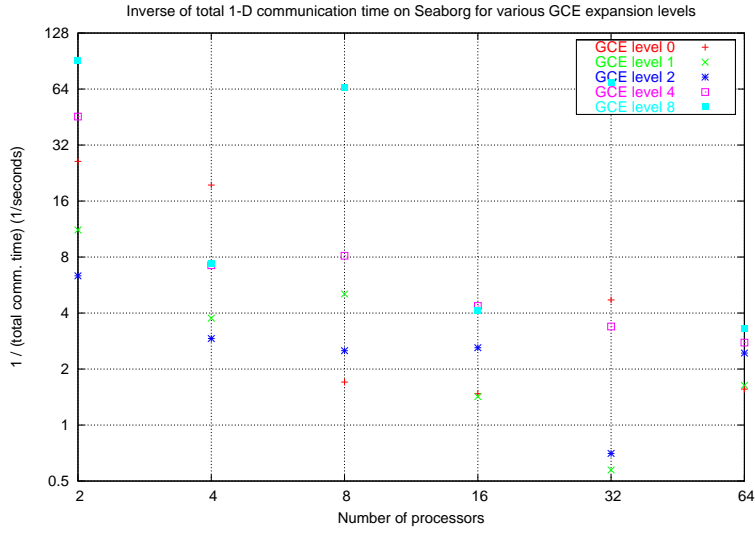


Figure 20: Inverse of total 1-D communication time on Seaborg for various GCE expansion levels. Each local domain consists of 2^{20} cells.

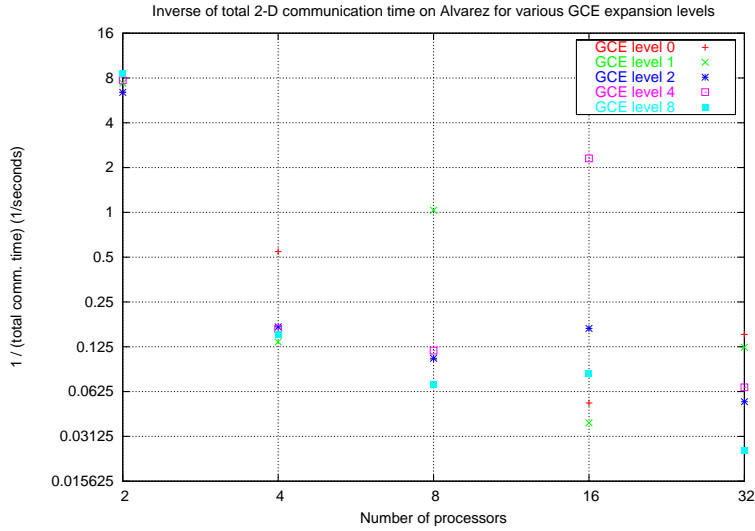


Figure 21: Inverse of total 2-D communication time on Alvarez for various GCE expansion levels. Each local domain consists of $2^{10} \times 2^{10}$ cells.

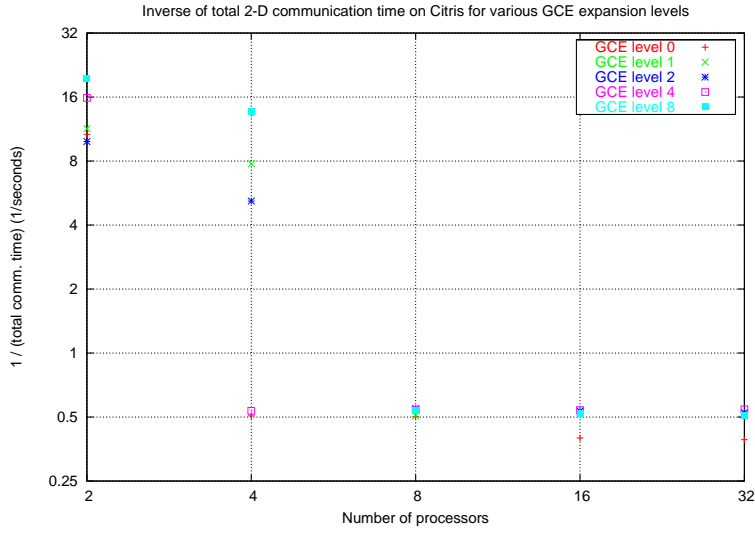


Figure 22: Inverse of total 2-D communication time on Citris for various GCE expansion levels. Each local domain consists of $2^{10} \times 2^{10}$ cells.

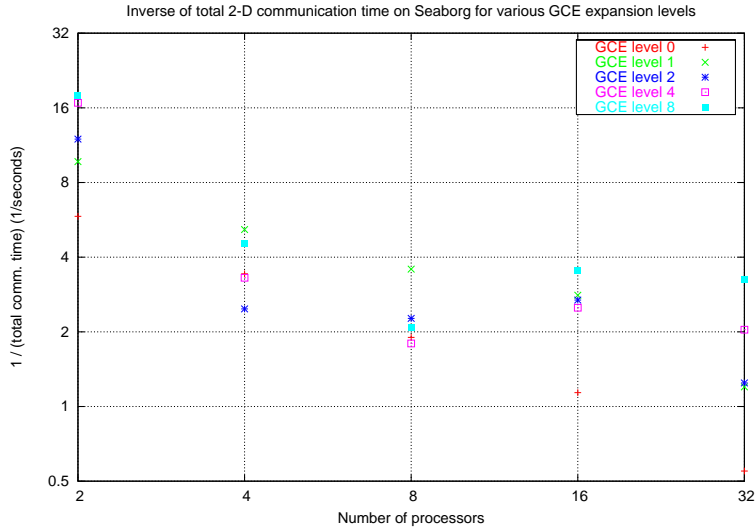


Figure 23: Inverse of total 2-D communication time on Seaborg for various GCE expansion levels. Each local domain consists of $2^{10} \times 2^{10}$ cells.

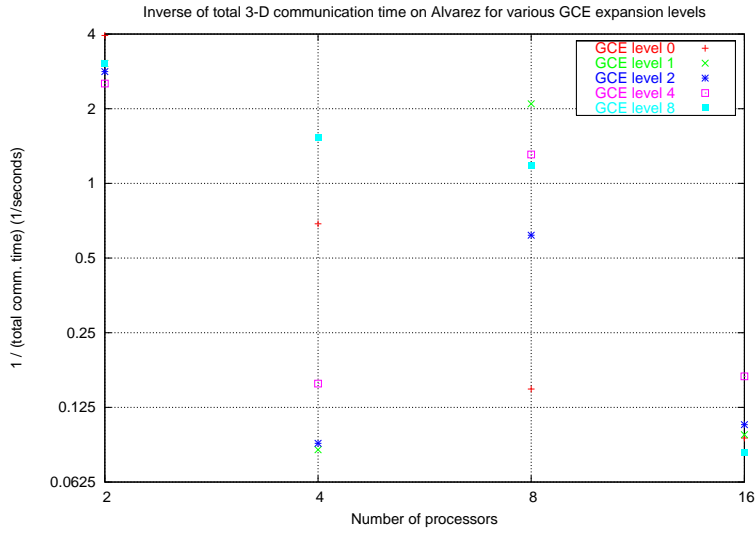


Figure 24: Inverse of total 3-D communication time on Alvarez for various GCE expansion levels. Each local domain consists of $64 \times 64 \times 64$ cells, and 100 timesteps were used.

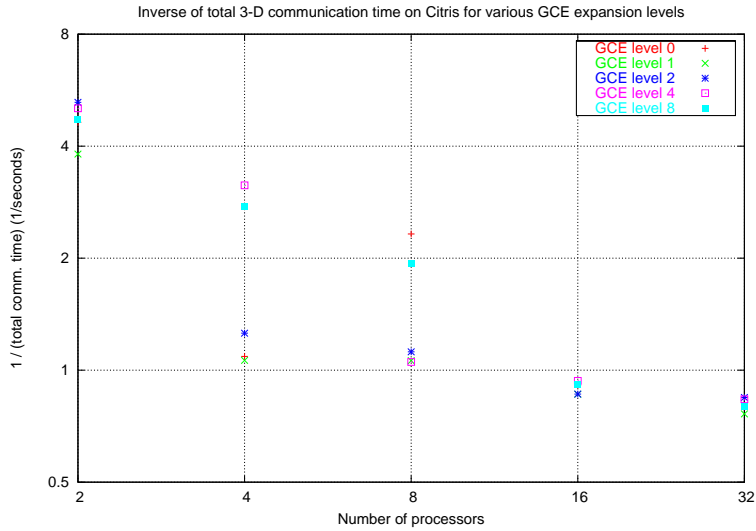


Figure 25: Inverse of total 3-D communication time on Citris for various GCE expansion levels. Each local domain consists of $64 \times 64 \times 64$ cells, and 100 timesteps were used.

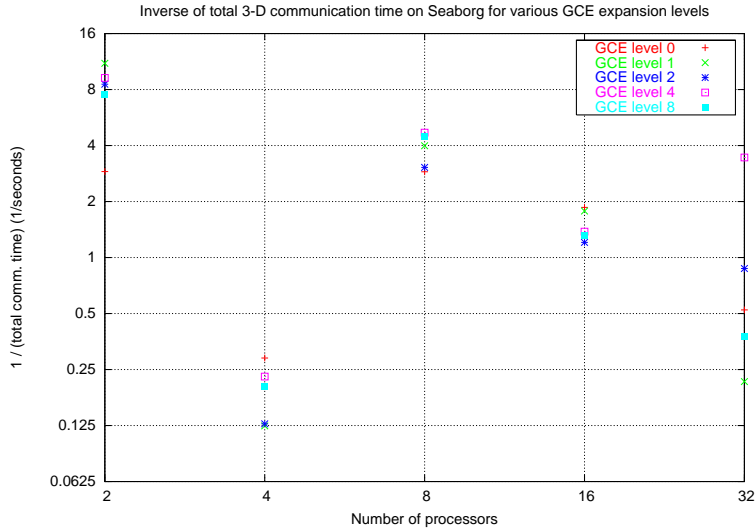


Figure 26: Inverse of total 3-D communication time on Seaborg for various GCE expansion levels. Each local domain consists of $64 \times 64 \times 64$ cells, and 100 timesteps were used.

Platform	Processor	Worst recorded timer resolution
Alvarez	Intel Pentium III	$2.15 \times 10^{-6} s$
Citris	Intel Itanium 2	$1.19 \times 10^{-6} s$
Seaborg	IBM POWER3	$4.05 \times 10^{-6} s$

Table 5: Resolution of `gettimeofday()` on test platforms.

9 Timer resolution

We used the POSIX function `gettimeofday()` for timings, since early test runs showed that `MPI_Wtime()` had an unacceptably low resolution (approximately $10^{-3} s$ on Itanium 2 platforms). To ensure accuracy, during several large (16 or 32 processor) test runs, we measured the resolution of each system’s `gettimeofday()` implementation, and recorded the worst over all the processors. Results are shown in Table 5. Given that timings were taken over 100 timesteps, a measurement of 0.01 seconds has at least three significant figures. This does not apply to the initialization timings, which measure a once-per-run event.

References

- [1] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith, *The portable, extensible toolkit for scientific computing, version 2.0.22*, <http://www.mcs.>

anl.gov/petsc/, 1998.

- [2] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, and K. Yelick, *An evaluation of current high-performance networks*, 17th International Parallel and Distributed Processing Symposium, 2003.
- [3] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken, *LogP: Towards a realistic model of parallel computation*, Principles Practice of Parallel Programming, 1993, pp. 1–12.
- [4] UC Berkeley EECS Department, *UC Berkeley Clustered Computing*, <http://www.millennium.berkeley.edu/citris/index.htm>, March 2004.
- [5] C.H.Q. Ding, *Simulating Lattice QCD on a Caltech/JPL Hypercube*, Int'l J. Supercomp. Appl. **5** (1991), 73–80.
- [6] C.H.Q. Ding and Y. He, *A ghost cell expansion method for reducing communications in solving pde problems*, Proceedings of the 2001 ACM/IEEE conference on Supercomputing (New York, NY, USA), ACM Press, November 2001.
- [7] National Energy Research Scientific Computing Center (NERSC), *Seaborg home page*, <http://hpcf.nersc.gov/computers/SP/>, October 2003.
- [8] ———, *Alvarez home page*, <http://www.nersc.gov/alvarez/>, April 2004.
- [9] Bruce Palmer and Jarek Nieplocha, *Efficient algorithms for ghost cell updates in two classes of MPP architectures*, Proceedings of the Fifteenth International Conference on Parallel and Distributed Computing Systems (PDCS) (Calgary, Canada), ACTA Press, 2002.
- [10] Michelle Mills Strout, Larry Carter, Jeanne Ferrante, Jonathan Freeman, and Barbara Kreaseck, *Combining performance aspects of irregular gauss-seidel via sparse tiling*, 15th Workshop on Languages and Compilers for Parallel Computing (LCPC) (College Park, Maryland), July 2002.
- [11] David Wonnacott, *Time skewing for parallel computers*, Languages and Compilers for Parallel Computing, 1999, pp. 477–480.
- [12] Yunhai Wu, Xiao-Chuan Cai, and David E. Keyes, *Additive Schwarz methods for hyperbolic equations*, Tenth International Conference on Domain Decomposition Methods (J. Mandel, C. Farhat, and X.-C. Cai, eds.), AMS, Contemporary Mathematics 218, 1998, pp. 513–521.