

# A Plug-and-Play Model for Evaluating Wavefront Computations on Parallel Architectures\*

Gihan R. Mudalige<sup>§</sup> Mary K. Vernon<sup>‡</sup> and Stephen A. Jarvis<sup>§</sup>

<sup>§</sup>Dept. of Computer Science  
University of Warwick  
Coventry, CV4 7AL, U.K

{g.r.mudalige,saj}@dcs.warwick.ac.uk

<sup>‡</sup>Dept. of Computer Sciences  
University of Wisconsin-Madison  
Madison, WI 53706-1685

vernon@cs.wisc.edu

## Abstract

*This paper develops a plug-and-play reusable LogGP model that can be used to predict the runtime and scaling behavior of different MPI-based pipelined wavefront applications running on modern parallel platforms with multi-core nodes. A key new feature of the model is that it requires only a few simple input parameters to project performance for wavefront codes with different structure to the sweeps in each iteration as well as different behavior during each wavefront computation and/or between iterations. We apply the model to three key benchmark applications that are used in high performance computing procurement, illustrating that the model parameters yield insight into the key differences among the codes. We also develop new, simple and highly accurate models of MPI send, receive, and group communication primitives on the dual-core Cray XT system. We validate the reusable model applied to each benchmark on up to 8192 processors on the XT3/XT4. Results show excellent accuracy for all high performance application and platform configurations that we were able to measure. Finally we use the model to assess application and hardware configurations, develop new metrics for procurement and configuration, identify bottlenecks, and assess new application design modifications that, to our knowledge, have not previously been explored.*

## 1 Introduction

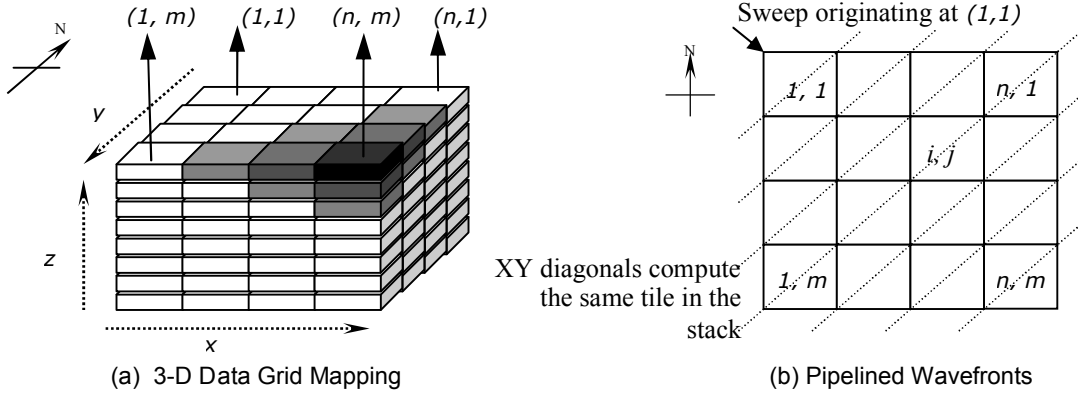
Particle transport and other parallel wavefront applications form up to 50-80% of high performance scientific computing load [1] at institutions such as the Los Alamos National Laboratories (LANL) and the Atomic Weapons Establishment (AWE) in the U.K. For this reason, benchmark codes that are representative of this

class of applications – such as the codes known as Sweep3D from LANL and Chimaera from AWE – are used to evaluate new high performance architectures during both design and procurement. The communication primitives used in these parallel benchmark codes are the blocking send, receive, and group communication primitives in the Message Passing Interface (MPI) [2].

To aid design and procurement decisions, considerable recent research has developed and applied accurate parametrized analytic models of Sweep3D on high performance architectures [1, 3, 4, 5, 6]. These and a similar model of LU [7], sum the critical path computation and communication times to obtain the total execution time on a given parallel architecture. Several of the previous papers [1, 3, 4, 8] have demonstrated that these abstract models can be highly accurate. On the other hand, each previous application model requires significant and unspecified modifications in order to apply it to other wavefront codes, such as Chimaera or a production code of interest to a given organization. Hoisie et. al [1] provide a model of a single sweep in any wavefront code, but that model also requires significant customization to represent message contention, the structure of the sweeps, and other operations in an actual benchmark or production code. A key point is that the model equations must be developed from scratch for each new code.

In this paper we investigate the open question of whether a plug-and-play re-usable model can be developed, such that the user only needs to specify a few input parameters to obtain a model of any given wavefront application on a parallel architecture, or to assess various possible design changes in the wavefront application or parallel architecture. We develop such a model, validate it on a large Cray XT4, and apply the model to assess example system procurement and configuration questions as well as several key application and architecture design alternatives. Specifically, we make the following contributions:

\*This work was partially sponsored by the NSF under Grant CNS-0435437 and by the United Kingdom Atomic Weapons Establishment under grants CDK0660 and CDK0724. Gihan Mudalige was funded by a Warwick-Wisconsin Madison Research Exchange Fellowship.



**Figure 1: Data Mapping and Sweep Wavefronts**

- We modify and extend a previous LogGP model of Sweep3D [3] to create a plug-and-play model that reflects the functional behavior of existing and imaginable wavefront computations that use MPI and operate on a regular orthogonal grid of data. Only a small number of input parameters are needed to specify the particular behavior of any given wavefront benchmark or production code. Other new features of the model include (a) extensions to model the application performance on multicore (CMP) nodes, and (b) a more precise model of message contention in the multicore nodes than previous work.
- The new input parameters capture the key structural and behavioral differences among different wavefront application codes, providing insight into alternative wavefront application designs.
- We develop and validate new models of off-chip and on-chip MPI send, receive, and all-reduce operations on the Cray XT4. Results quantify XT4 communication software overhead and hardware latencies, showing that the communication software and hardware is highly optimized.
- We validate the new plug-and-play application model using parameters for LU, Sweep3D and Chimaera computations on the Cray XT4, which has dual-core nodes. The model predicts execution time on up to 8192 processors with less than 5% error for LU and less than 10% error for all high performance configurations of the particle transport benchmarks. These accuracies, comparable to previous models, are sufficient for platform and application design assessments as illustrated in Section 5. The validated model is also the first analytic model for AWE’s Chimaera benchmark.
- Model results show that synchronization costs, which were previously found to be significant on the SP/2, are a negligible fraction of the total execution time for up to 8192 processors on the XT4.
- Applications of the model in Section 5 provide new approaches to procurement and configuration assessments,

as well as new wavefront application optimizations, and new insight into model utility.

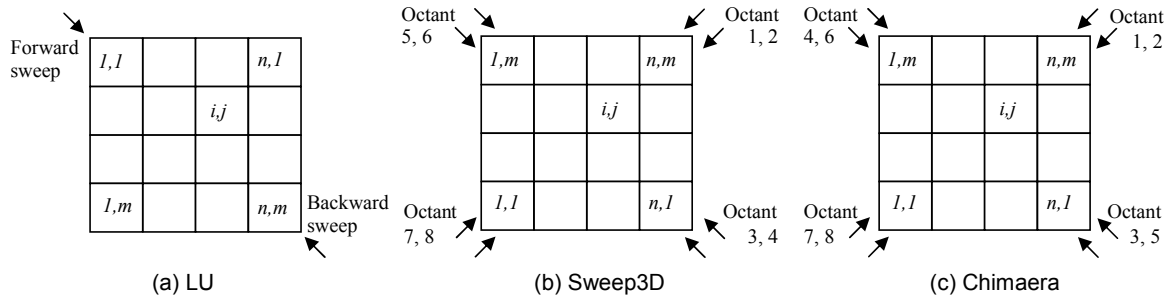
Section 2 provides background on pipelined wavefront applications - LU, Sweep3D and Chimaera benchmarks, and previous models. Section 3 develops and validates the models of MPI primitives for the Cray XT3/XT4 system. This section also provides new measures of XT4 communication performance, in terms of  $L$ ,  $\alpha$ , and  $G$ . Section 4 develops the plug-and-play LogGP performance model that reflects the operation of alternative wavefront codes, including the input parameters for each of the three benchmarks. Section 5 applies the new models of Chimaera and Sweep3D to assess the quantitative impact of various application and platform configuration and design changes on the execution time of large particle transport simulations. Section 6 summarizes the results and concludes the paper.

## 2 Background

### 2.1 Pipelined Wavefront Sweeps

A pipelined wavefront sweep operates on a three dimensional discretized grid of data cells, depicted in Figure 1(a). The dimensions of the data are denoted by  $x$ ,  $y$ , and  $z$ , with the total number of cells given by  $N_x \times N_y \times N_z$ . The 3-D data grid is partitioned and mapped onto a two-dimensional  $m \times n$  array of processors, such that each processor is assigned a stack of data cells of size  $N_x/n \times N_y/m \times N_z$  as also depicted in the figure. A processor is indexed as  $(i, j)$ , where  $i$  is the horizontal position (column number) and  $j$  is the vertical position (row number) respectively. Each partition of data cells assigned to a processor can be viewed as a stack of tiles, with each tile being one or more grid points high in the  $z$ -dimension.

A sweep starts with one of the corner processors computing over the cells in its top or bottom tile. Consider the case that processor  $(1, 1)$  in Figure 1(a) begins by computing the results for its bottom-most tile. At the end of that calculation, it sends the respective boundary values (i.e., the new values for the data cells at the edge of the tile) to processors



**Figure 2: Structure of the Sweeps per Iteration in Key Benchmark Codes**

(1, 2) and (2, 1). After receiving those values, processors (1, 2) and (2, 1) compute values for each of their bottom tiles, while processor (1, 1) computes values for the next tile in its stack. Each processor sends boundary values to its east and south neighbors and then computes new values for the next tile in its stack, and so forth, until all the tiles have been processed. This creates a series of wavefronts, depicted in Figure 1(b), since the processors along each  $x - y$  diagonal are all processing the tile at the same position in their respective stacks. The sweep ends when the processor  $(n, m)$  at the opposite corner finishes processing its top-most tile – that is, the tile at the opposite corner of the 3-D grid. The shaded tiles in Figure 1(a) depict the tiles that are processed during the final three wavefronts – light gray, then medium gray, then dark gray, ending with the top-most tile on  $(n, m)$ . Such pipelined sweeps are the bulk of the execution time in wavefront computations such as particle transport simulations.

## 2.2 LU, Sweep3D, and Chimaera

As noted in Section 1, LU, Sweep3D and Chimaera are important scientific benchmark codes. LU is a NAS benchmark that represents a compressible Navier-Stokes equation solver used in computational fluid dynamics. Sweep3D is an ASC benchmark developed by LANL to represent particle transport applications that make up 50-80% of the computations that run on their high performance systems [1]. Chimaera is a particle transport benchmark from the Atomic Weapons Establishment (AWE) in the U.K., representative of a major portion of their workload and also used for the procurement of their high performance systems.

Figures 2(a) and (b) show the starting points and ordering of the pipelined sweeps that are performed in each iteration of LU and Sweep3D, respectively. (Note that processor indexing in the 2-D array is different in each of these codes.) An iteration in LU consists of two sweeps [7], one sweep starting from the bottom most tile on processor (1,1) to the top-most tile on processor  $(n, m)$ , followed by a sweep in the opposite direction. In contrast, Sweep3D has eight sweeps, one originating from each corner (or octant) of the 3-dimensional grid [3], as shown in the figure.

To our knowledge, the structure of the sweeps in Chimaera has not previously been reported, but from the code

we determined the structure shown in Figure 2(c). Note that in both Sweep3D and Chimaera, processor  $(n, m)$  begins the first tile of the second sweep immediately after it finishes the last tile of the first sweep, and the third sweep begins when processor  $(n, 1)$  has completed its stack of tiles in the second sweep. In Sweep3D, the fourth sweep begins as soon as processor  $(n, 1)$  has finished its tiles in the third sweep, but in Chimaera the fourth sweep does not begin until processor  $(1, m)$  at the opposite corner finishes the third sweep. These are some of the similarities and differences that need to be captured in the input parameters for a plug-and-play re-usable model.

Additional structural and behavioral differences among these three benchmarks and other wavefront codes also need to be captured in a re-usable model. To our knowledge, there are no previous papers detailing those differences nor detailing the operation of the Chimaera code. In Section 4 we outline the key further differences obtained from the source codes, to motivate the design decisions in the new re-usable wavefront application model.

## 2.3 Related Work

In an early paper, Yarrow et. al [7] develop models of two different versions of LU that elucidate the differences in communication structure. They find a maximum of 30% error compared with measured LU execution time on an SP system. Sundararam-Stukel and Vernon [3] develop a LogGP model of Sweep3D on the SP/2 that includes synchronization overheads, and obtain a high degree of accuracy for up to 128 processors. Their results include optimized application configuration parameters and number of processors for key problem sizes. They also show that synchronization cost is significant on the SP/2. Mathis et. al [6] use less precise communication costs in a model of a single sweep in Sweep3D and apply the model to explore two possible alternative data decompositions. Kerbyson et. al [4] provide a Sweep3D model that is derived from the Hoisie et. al [1] single-sweep model, and use the model to project performance for the Earth Simulator. Validations of their model on BlueGene/L, ASC Purple at Lawrence Livermore and the Cray RedStorm at Sandia are presented in [8]. Their model does not quantify communication software overhead ( $o$ ) and latency ( $L$ ). They include terms for

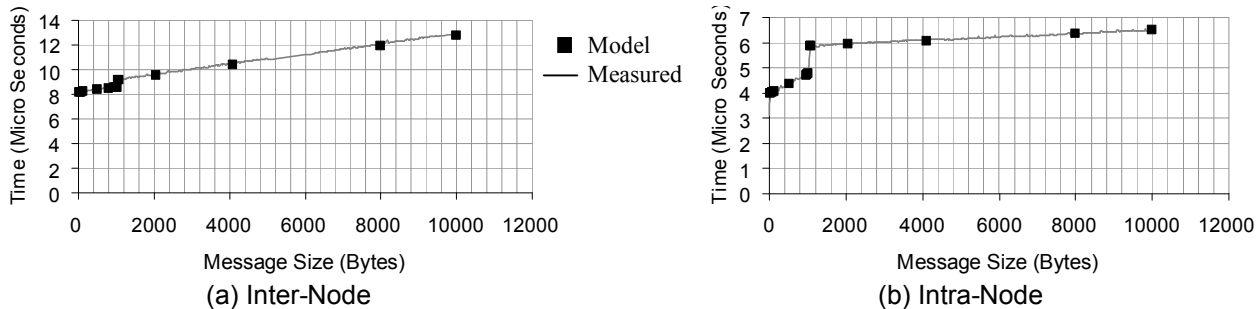


Figure 3: Measured and Modeled MPI End-to-End Communication Times

multi-core contention on the outgoing communication link. The model validates well on dual-core single-link nodes but total communication time (rather than contention) reduces to zero as the number of links per node increases.

Each of these previous analytic models of wavefront applications are specific to LU or Sweep3D and require significant (unspecified) restructuring to apply to other benchmarks or production codes of interest. This paper derives a plug-and-play LogGP model for a wide variety of wavefront codes, derives XT4 communication overhead, latency, and bandwidth, models contention more precisely, and provides new applications of the models for system design and configuration. The LogGP model and its parameters are described in [9].

### 3 MPI Communication on the XT4

A key component of a LogGP application model is the set of sub-models that define the execution time of the communication primitives on the platform of interest. Thus, the plug-and-play model of wavefront codes developed in Section 4 requires LogGP sub-models of MPI send and receive, and the MPI group all-reduce primitive.

The platform of interest here is the high performance Cray XT3/XT4 which we have access to through the PEAC Project at ORNL. Each node in the XT4 is a dual core 2.6GHz Opteron [10]. The interconnection between nodes is a 3-D torus network, which facilitates efficient mapping of wavefront applications and implies near-neighbor send/receive operations.

To our knowledge, XT4 MPI communication models have not previously been reported on in the literature. Hence, we derive these models below. Note that we confirmed the basic operation of the XT4 MPI implementations with XT4 system architects. To our knowledge, the models in Section 3.2 are also the first validated LogGP models of *on-chip* MPI send/receive for any platform.

The LogGP communication models derived below can be used for any application that uses MPI primitives. They also yield insights into the implementation as well as quantitative values of the end-to-end communication latency ( $L$ ), the processing overhead ( $o$ ) at the sender and receiver, and

the per-byte transmission cost ( $G$ ). Thus, these models are valuable in their own right.

Note that in modern architectures, a node can transmit a new message as soon as a previous message transmission is complete, and thus the gap parameter,  $g$ , is equal to zero.

#### 3.1 MPI Send and Receive: Off-node

Figure 3(a) plots one half of the round-trip time for a ping-pong message exchange between two nearest neighbor nodes in the XT4, as a function of the size of the message that is transmitted back and forth. Note that each node posts a receive immediately after completing a send, and thus there is very low variance in the measured round-trip times. The solid line connects the measured values, while the points in black are the highly accurate values predicted by equations (1) and (2) in Table 1.

The slopes of the curves before and after the 1024 byte message size are equal. This slope is the per-byte transmission cost or Gap per byte ( $G$ ), given in Table 2.  $G$  is the sum of the per-byte costs for each of the copy operations at the sender and receiver. Note that  $1/G$  yields an XT4 inter-node bandwidth of 2.5 Gigabytes/second.

For messages smaller than 1025 bytes, equation (1) models the total time to send a message, including the message processing time ( $o$ ) at each end and the latency ( $L$ ) between the two processors. For all messages larger than 1024 bytes a handshake is performed. That is, the sender first sends a short message requesting a reply when the message receive has been posted, and waits for a reply before sending the message. Let  $h$  denote the total time for the handshake, obtained as the difference in transmission times for 1025 and 1024 bytes. Let  $o = o_{init} + o_{c2NIC}$  where  $o_{init}$  denotes the overhead for the copy between application and kernel, and  $o_{c2NIC}$  denotes the time to set up a (DMA or other) copy of the message data between kernel memory and the NIC and to prepare or process the message header. Using these processing overheads, (2) models the total time to send a message larger than 1024 bytes, including the handshake time.

Assuming that  $o_h$  is negligible we solve equations (1) and (2) for given respective message sizes simultaneously to derive the values of  $o$  and  $L$ , given in Table 2. These

**Table 1: LogGP Model of XT4 MPI Communication**

<b>(a) Off-Node Communication Model</b>	
$Total\_Comm_{\leq 1KB,offchip} = o + Message\_size \times G + L + o$	(1)
$Total\_Comm_{>1KB,offchip} = o + h + o + Message\_size \times G + L + o$ where $h = L + o_h + L + o_h$	(2)
$Send_{\leq 1KB,offchip} = o, Receive_{\leq 1KB,offchip} = o$	(3)
$Send_{>1KB,offchip} = o + h$	(4a)
$Receive_{>1KB,offchip} = L + o + Message\_size \times G + L + o$	(4b)
<b>(b) On-Chip Communication Model</b>	
$Total\_Comm_{\leq 1KB,onchip} = o_{copy} + Message\_size \times G_{copy} + o_{copy}$	(5)
$Total\_Comm_{>1KB,onchip} = o + Message\_size \times G_{dma} + o_{copy}$	(6)
$Send_{\leq 1KB,onchip} = o_{copy}, Receive_{\leq 1KB,onchip} = o_{copy}$	(7)
$Send_{>1KB,onchip} = o = o_{copy} + o_{dma}$	(8a)
$Receive_{>1KB,offchip} = Message\_size \times G_{dma} + o_{copy}$	(8b)

values provide the predicted communication times plotted in Figure 3(a). Note that the model is highly accurate. Furthermore, the off-node parameters in Table 2 are one to two orders of magnitude lower than the values for the SP/2 [3], which are:  $G = 0.07 \mu\text{sec}/\text{byte}$ ,  $L = 23 \mu\text{sec}$ , and  $o = 23 \mu\text{sec}$ . Thus, the Cray XT4 communication hardware and software are highly optimized.

LogGP application models also require sub-models of the time to execute the Send and Receive code. These are easily derived from the total time to transmit a message, and are given in equations (3),(4a) and (4b) in the Table 1.

### 3.2 MPI Send and Receive: On-chip

Figure 3(b) provides half the round-trip in the ping-pong MPI communication benchmark as a function of message size when the sender and receiver are on the same dual-core chip. We again observe a significant increase in transmission time for message size equal to 1025 bytes. However, in this case it is unlikely that a handshake operation with the other on-chip core could account for the magnitude of this increase. Instead, note that the slope of the curve for message sizes below 1024 bytes is larger than the slope for larger messages. These two per-byte transmission costs -  $G_{copy}$  and  $G_{dma}$  - are given in Table 2. The subscripts denote that it is likely that the larger messages are transferred using a dma operation. Since a slower message copy is used for messages smaller than 1025 bytes, the fixed increase at 1025 bytes is due to the dma setup cost.

Using the notation in the previous section, and assuming  $L \approx 0$  for on-chip message transmission, the total time to send a message smaller than 1025 bytes is modeled in equation (5). Note that  $o_{copy}$  is the processing overhead before and after the message copies on the sender and receiver while  $G_{copy}$  is the total time per byte to copy the data from one application buffer to the other. For message sizes larger than 1024, we let  $o = o_{copy} + o_{dma}$  and model the total end

to end message communication time in equation (6).

We solve equations (5) and (6) simultaneously for given respective message sizes to obtain values for  $o$  and  $o_{copy}$ . These are given in Table 2. The value of  $o$  is nearly the same as in the off-node model. Additionally, the per-byte gap to move the data from sender to receiver is lower on-chip than off-node for all message sizes. These observations greatly increase our confidence in these communication models and derived parameter values.

Similar to the off-node model, the models for Send and Receive are given by equation (7) in Table 1 for message size less than or equal to 1024 bytes and by (8a) and (8b) for messages larger than 1024 bytes.

**Table 2: XT4 Communication Parameters**

Off-node	Value	On-chip	Value
$G$	0.0004	$G_{copy}$	0.000789 $\mu\text{s}/\text{byte}$
	$\mu\text{s}/\text{byte}$	$G_{dma}$	0.000072 $\mu\text{s}/\text{byte}$
$L$	0.305 $\mu\text{s}$	$o$	3.80 $\mu\text{s}$
$o$	3.92 $\mu\text{s}$	$o_{copy}$	1.98 $\mu\text{s}$

### 3.3 MPI All-reduce

To complete this section, we develop a simple abstract model of the MPI all-reduce execution time. The all-reduce operation is performed at the end of each iteration in Sweep3D and Chimaera. Letting  $C$  denote the number of cores per node, the all-reduce execution time is given by  $T_{allreduce} = [\log_2(P) - \log_2(C)] \times C \times Total\_Comm_{offchip} + \log_2(C) \times C \times Total\_Comm_{onchip}$  (9)

Note that in the special case of  $C = 1$ , the equation reduces to  $\log_2(P) TotalComm$ . This all-reduce model has less than 2% error for up to 1024 dual-core nodes on the XT4, and thus provides insight into all significant processing and communication costs in this group communication primitive. The model can be used for any applications that perform all-reduce.

**Table 3: Model Application Parameters**

Parameter	LU	Sweep3D	Chimaera
$N_x, N_y, N_z$	<i>Inputsize</i>	<i>Inputsize</i>	<i>Inputsize</i>
$W_g$	<i>measured</i>	<i>measured</i>	<i>measured</i>
$W_{g,pre}$	<i>measured</i>	0	0
$H_{tile}(cells)$	1	$mk \times mmi/mmo$	1
$n_{sweeps}$	2	8	8
$n_{full}$	2	2	4
$n_{diag}$	0	2	2
$T_{nonwavefront}$	$T_{stencil}$	$2T_{allreduce}$	$T_{allreduce}$
<i>Message Size<sub>EW</sub> (Bytes)</i>	$40N_y/m$	$8H_{tile} \times \#angles \times N_y/m$	$8H_{tile} \times \#angles \times N_y/m$
<i>Message Size<sub>NS</sub> (Bytes)</i>	$40N_y/m$	$8H_{tile} \times \#angles \times N_x/n$	$8H_{tile} \times \#angles \times N_x/n$

## 4 The Plug-and-Play Model

In this section we develop a plug-and-play re-usable LogGP model for wavefront applications. Section 4.1 outlines the significant structural and behavioral differences among wavefront codes, and develops a simple set of parameters to capture those differences. Section 4.2 develops the basic re-usable model equations, assuming for simplicity that each core executing the computation is on a different node. Section 4.3 then extends the model for the case that the computation runs on multi-core chips or nodes.

```

For each tile of height 1 do
  Pre-Compute specified domain of grid
  Receive from west; Receive from north
  Compute specified domain of grid
  Send to east; Send to south

```

(a) LU

```

For each angle-group of size mmi do
  For each tile of height mk do
    Receive from west; Receive from north
    Compute specified domain of grid
    Send to east; Send to south

```

(b) Sweep3D

**Figure 4: Wavefront Operations**

### 4.1 Application Parameters

Recall from Figure 2 in Section 2.2 that different wavefront applications can have different numbers of sweeps as well as a different structure to the sweeps. We specify the number of sweeps,  $n_{sweeps}$ , as shown in Table 3.

In the case of LU, sweep 1 must completely finish executing on all the processors before sweep 2 can begin, and sweep 2 must also completely finish before the iteration ends. In Sweep3D, sweep 4 must complete before

sweep 5 begins and sweep 8 must complete before the iteration ends. However, as shown in Figure 2(b), sweep 2 in Sweep3D can begin as soon as the first corner processor  $(n, m)$  finishes its stack of tiles for sweep 1, and sweep 3 can begin as soon as the stack of tiles for sweep 2 have been processed by the main diagonal processor  $(n, 1)$ . Note that while sweep 3 is starting up in Sweep3D, sweep 2 is finishing its last few wavefronts. Chimaera has some similarities and some differences in how soon each sweep follows the previous sweep, as shown in Figure 2 and noted in Section 2.2. Other wavefront applications may still have other structures for their sweeps.

We have developed two new simple parameters namely,  $n_{full}$  and  $n_{diag}$ , also given in Table 3, to capture the relevant behavior of a wide range of possible sweep structures.  $n_{full}$  specifies the number of sweeps that must fully complete before the next sweep begins, while  $n_{diag}$  specifies the number of sweeps that must complete at the second corner processor on the main diagonal of the wavefronts. All other sweeps (e.g., sweep 2 in Sweep3d) only need to complete on the processor where the sweep originates before the next sweep begins. The reader can verify that the values of  $n_{full}$  and  $n_{diag}$  for each application in Table 4 are accurate for the corresponding sweep structure in Figure 2.

Further differences among the three codes, including those illustrated in the bold-faced portions in Figure 4, are captured using parameters as follows. First, LU performs a pre-calculation before performing the MPI receives, while Sweep3D and Chimaera do not. We use  $W_{g,pre}$  (Table 3) to specify the computation per grid point that occurs before the receives, and set this parameter to zero if no computation is performed. Second, Sweep3D has an input parameter ( $mmo$ ) that defines the number of angles to be computed for each data cell, whereas Chimaera and LU have a fixed amount of work per data cell, as shown in Table 3. Sweep3D also has a parameter ( $mmi$ ) that specifies the number of angles to be computed before sending the boundary values to the near neighbors. Common values of these parameters in the benchmark are 6 and 3, respectively. We use  $mmi$  and  $mmo$  to compute an effective value of the height of the tile, as described next. Third, Sweep3D has a parameter ( $mk$ ) that defines the height of a tile (in terms of the number of grid cells). We define a new parameter  $H_{tile}$  in our model inputs in Table 3. LU and Chimaera each have a fixed tile height equal to one cell. Sweep3D computes  $mmi$  of the angles in the tile before sending the boundary values, and then computes another  $mmi$  of the angles. In terms of total code execution time, this is the same as computing all of the angles for a tile of height  $H_{tile} = mk \times mmi/mmo$ , as shown in the table. Note that this implies that  $W_g$  is the measured total computation time for all angles in a cell.

Parameter  $T_{nonwavefront}$  is the execution time for the

**Table 4: LogGP Model for Sweep3D from [3]**

$W_{i,j} = W_g \times mmi \times mk \times jt \times it$	(s1)
$StartP_{i,j} = \max(StartP_{i-1,j} + W_{i-1,j} + Total\_comm + Receive,$ $StartP_{i,j-1} + W_{i,j-1} + Send + Total\_Comm)$	(s2)
$Time_{5,6} = StartP_{1,m} + 2[(W_{1,m} + Send_E + Receive_N + (m-1)L) \times \#kblocks \times mmo/mmi]$	(s3)
$Time_{7,8} = StartP_{n-1,m} + 2[(W_{n-1,m} + Send_E + Receive_W + Receive_N + (m-1)L + (n-2)L)$ $\times \#kblocks \times mmo/mmi] + Receive_W + W_{n,m}$	(s4)
$T = 2(Time_{5,6} + Time_{7,8})$	(s5)

operations performed between iterations. For instance LU performs a four-point stencil computation after the 2 sweeps in each iteration, while Sweep3D performs two all-reduce operations. The model of stencil execution time ( $T_{stencil}$ ) is omitted to conserve space but is a sum of terms with similar simplicity and abstraction as the all-reduce model.

Note that a wide range of different wavefront application behaviors are captured in the relatively simple and small set of application parameters in Table 3. In particular, the parameters can be used to specify various amounts of work before and after the boundary values are received, a range of tile height, an arbitrary number of sweeps per iteration, a wide range of sweep precedence structures including those in the three benchmarks, and a general processing time between iterations. Hence these application parameters support the evaluation of LU, Sweep3D, Chimaera, other possible wavefront applications, and many if not most possible application code design changes.

The parameters are more complete than previous parameters for Sweep3D or LU because they include both the sweep structure and the computations that are performed at the end of each iteration. As discussed in Section 5, the reusable model accurately computes execution time for each wavefront code from these application parameters. Hence, the parameter values provide a succinct summary of the key differences among wavefront codes with respect to measured application performance.

## 4.2 Re-usable Model: One Core per Node

Table 4 provides the accurate LogGP model of Sweep3D in [3], which serves as a starting point as well as a useful comparison and contrast for our plug-and-play re-usable model for a wide variety of wavefront applications. We briefly discuss these previous equations before developing the new reusable model. Note that in this Section we assume the computation is mapped to one core per node on the XT4. In this case all communication is off-node and the sub-models in equations (1)-(4) apply.

Equation (s1) models the time to compute each set of  $mmi$  (out of  $mmo$ ) angles for a tile. The parameters  $it$  and  $jt$  define the  $x$  and  $y$  dimensions of the tile. Furthermore,  $W_g$  in this previous model is the computation

time for one angle of one data cell whereas in our new model this parameter is the total execution time for all the angles of one data cell. The total execution time for a sweep is the same regardless of which corner it originates from. The model computes execution times for a sweep that starts from the upper left corner in the processor grid using the processor indexing in Figure 1(b), and then applies portions of the sweep time to the appropriate actual sweeps in the code. Equation (s2) defines the time at which the sweep starts on each other processor in the grid. Equation (s3) computes the time until the corner processor on the main diagonal completes its stack of tiles in the sweep. Equation (s4) computes the time until the sweep completely finishes on processor  $(n, m)$ , and equation (s5) sums the total time to execute the 8 sweeps.

The specific terms in each equation are described in detail in [3]. Here we simply focus on the overall structure of the model and note that the terms in each equation reflect the sequencing of the operations in the code. We also note that the synchronization term in equation (s3), namely  $(m-1)L$ , and the term in equation (s4), namely  $(m-1)L + (n-2)L$ , are due to back-propagation of handshake replies. Synchronization costs were significant on the IBM SP/2, but are a tiny fraction of the total execution time on up to 8192 processors on the XT4 due to the extremely small value of  $L$  on the XT4. For this reason, we were unable to verify the form of the terms on the XT4. We omit the synchronization terms in the development of the re-usable model, noting that these previous or other synchronization terms can be incorporated in the re-usable model for other architectures, as needed.

The typical approach to modeling a new wavefront code is to modify an existing model such as the one in Table 4, to reflect the different behaviors in the new code. The high level of abstraction in the model facilitates modifications. However, the model modification process is error-prone and thus each new model must be extensively validated. Furthermore, model extensions such as new synchronization terms must be propagated to each of the customized models. To reduce such development and validation costs, we choose to modify the performance model so as to build in the impact of the various possible behaviors, relying on the input parameters developed in the previous section to specify the appropriate features for each application. The

**Table 5: Plug-and-play LogGP Model: One Core Per Node**

$W_{pre} = W_{g,pre} \times H_{tile} \times N_x/n \times N_y/m$	(r1a)
$W = W_g \times H_{tile} \times N_x/n \times N_y/m$	(r1b)
$StartP_{1,1} = W_{pre}$	(r2a)
$StartP_{i,j} = \max(StartP_{i-1,j} + W_{i-1,j} + Total\_comm_E + Receive_N, StartP_{i,j-1} + W_{i,j-1} + Send_E + Total\_Comm_S)$	(r2b)
$T_{diagfill} = StartP_{1,m}$	(r3a)
$T_{fullfill} = StartP_{n,m}$	(r3b)
$T_{stack} = (Receive_W + Receive_N + W + Send_E + Send_S + W_{pre})N_z/H_{tile} - W_{pre}$	(r4)
$Time\ per\ iteration = n_{diag}T_{diagfill} + n_{full}T_{fullfill} + n_{sweeps}T_{stack} + T_{nonwavefront}$	(r5)

idea is analogous to re-usable software which is popular due to reduced software development and testing costs.

To our knowledge, plug-and-play performance models – in which the user only needs to specify a few input parameter values in order to obtain performance predictions for application codes with different behavior – have not previously been developed. An open question addressed in this research is whether building in the various possible behaviors leads to a more complex set of equations, possibly negating the advantages of the model generality. The results below show that for the varied behaviors in wavefront applications, it has been possible to construct a set of equations that are as simple as the equations that are tailored to a given application. This was an unanticipated result that may not hold for other classes of applications. However, the results are encouraging for this important class of application and may provide an incentive to extend the study more widely.

The new plug-and-play performance model (for one core per node) is given in Table 5. We outline the key differences as compared to the previous model for Sweep3D in Table 4. Equations (r1a) and (r1b) provide the work per tile before and after the boundary values are received, respectively. These are computed in a similar way to equation (s1) in the previous model, but account for the pre-processing time before the boundary values arrive, if any. Equation (r1b) also illustrates the simplicity of incorporating the  $mmi$  parameter into  $H_{tile}$ . Equation (r2a) accounts for the pre-processing time for the first time in the sweep, and equation (r2b) is similar to the previous model equation (s2). Note in equation (r2b) that the first term on the right corresponds to the case where the message from the West is the last to arrive at processor  $(i, j)$ . In this case the message from the North has already arrived, but cannot be received until the West message is completely received. The second term corresponds to the case where a message from the North arrives last. In this case, processor  $(i, j - 1)$  does a send to its East before it sends to its South processor (i.e. to processor  $(i, j)$ ). The  $Total\_Comm$  in this case is for the message send and receive between processors  $(i, j - 1)$  and  $(i, j)$ . The next three equations provide a different perfor-

mance cost breakdown than the previous model. In particular, equations (r3a), (r3b) and (r4) specify the time for the sweep to reach the processors on the diagonal, the time to reach the processor at the opposite corner, and the time to process a stack of tiles, in contrast to the previous equations (s3) and (s4) which compute the total critical path time for the given sweeps in Sweep3D. The new breakdown is simple and intuitive and is also needed for computing the total execution time for various possible sweep precedence structures.

$W_{pre}$  does not appear in equations (r3a) and (r3b) because the parallel pre-computation for the first tile is accounted for in equation (r2a). The per-tile processing time in (r4) includes  $W_{pre}$  for each tile. The total number of tiles that need to be computed is given by  $N_z/H_{tile}$ . The subtracted  $W_{pre}$  is an adjustment for the final tile in the stack. The per-tile processing time also includes two sends and two receives. The processor at the corner of the main diagonal or opposite the processor that started the sweep will not perform both send operations. However, all processors compute their tiles at the same rate due to the blocking nature of the MPI send and receives.

Equation (r5) provides the time for one iteration of a wavefront computation by (a) combining the appropriate number of terms for sweeps that must complete at the main diagonal or at the opposite corner before the next sweep can begin, and (b) by adding the term for computations that occur at the end of the iteration or possibly between the sweeps. Note that equation (r5) provides for an infinite variety of sweep sequences while the model inputs  $n_{full}$  and  $n_{diag}$  are the key measures of the sweep precedence structure for a given application.

The new re-usable model is significantly more powerful than previous models of specific wavefront codes, yet it has a similarly small number of intuitive equations. As detailed in Section 5, the re-usable model is also highly accurate and comparable to the previous models of the Sweep3D code [1, 4, 8], unless the wavefront code is configured inefficiently. Model accuracy is discussed further in Section 5.



### 4.3 Multi-Core Reusable LogGP Model

To apply the reusable LogGP model for execution on multiple cores per node we first note that the  $W_{pre}$  and  $W_g$  inputs must be measured when the application executes on at least four cores, or the number of cores that will share a cache or other memory resources at a given node, whichever is larger. Four cores are required regardless of the number of cores per node [3], so that the code path that is executed is about the same as will be executed for larger configurations.

Two extensions to the model are also required. First, equation (r2b) needs to be modified to specify which of the MPI Send and Receive operations are on-chip and which are off-node. Second, message contention at shared node resources needs to be accounted for in equation (r4). Note that all of the communications in equation (r4) should be off-node because the processing of the stack of tiles occurs at the rate of the slowest communication in each direction.

Let the wavefront application be mapped to the multi-core nodes such that the cores at each node form a  $C_x \times C_y$  rectangle in the  $m \times n$  processor grid (see Figure 1 and Figure 2). In this case, the off-node communications occur at the edge of the rectangle. Let  $(i, j)$  again denote the location of each core in the processor grid, and note that processor indices start from 1 in both directions. Using this notation, Table 6 provides the required modification to equation (r2b) for on-chip communication between two cores on the same node. For example, the  $Send_E$  operation in equation (r2b) occurs between cores  $(i, j - 1)$  and  $(i + 1, j - 1)$ . This will be off-node if the core  $(i, j - 1)$  is at the right edge of the  $C_x \times C_y$  rectangle (i.e., if  $i \bmod C_x = 0$ ) and will otherwise be on-chip. The remaining rules are derived in a similar manner.

For message contention, we note that the primary message contention on the Cray XT4 will occur during the dma transfer of message data from kernel memory to the NIC via the shared bus. Once the message data is in the NIC memory, there should be very little contention since messages are traveling in one direction only between any two nodes and because the NIC has a separate port for each destination node. The time to transmit a message on the bus can be derived from the measured communication primitives in Section 3.2. For each message interference a value of  $I$  is added to the appropriate Send or Receive operation, as specified in Table 6.

Representative validation of the multi-core model for the dual-core Cray XT4 is given in Section 5. Similar to previous models [3, 8], a maximum of 10% error was observed for LU, Chimaera and Sweep3D, unless the problem size per node is small. The case where the problem size per node is small, the communication dominates the total execution time. Hence, such configurations are not of interest for production runs. In those cases of less practical interest,

**Table 6: Re-usable Model Extensions for CMP Nodes**

Modifications to Equation (r2b)
For $C_x \times C_y$ cores per node, all communication are off-node except the following: $i \bmod C_x \neq 0 \ \& \ C_x \neq 1 : Send_E = Send_{onchip,E}$ $i \bmod C_x \neq 1 \ \& \ C_x \neq 1 :$ $Total\_comm_E = Total\_comm_{onchip,E}$ $j \bmod C_y \neq 1 \ \& \ C_y \neq 1 : Receive_N = Receive_{onchip,N}$ $j \bmod C_y \neq 0 \ \& \ C_y \neq 1 :$ $Total\_comm_S = Total\_comm_{onchip,S}$
Modifications to Equation (r4)
For CMPs with a shared bus to memory let $I = (odma + Message\_size \times G_{dma})$ $1 \times 2$ cores/node : add $I$ to $Receive_N$ and $Send_S$ $2 \times 2$ cores/node : add $I$ to each $Send$ and $Receive$ $2 \times 4$ cores/node : add $2I$ to each $Send$ and $Receive$

the abstract communication and contention model leads to somewhat larger errors (i.e., in the order of 25%).

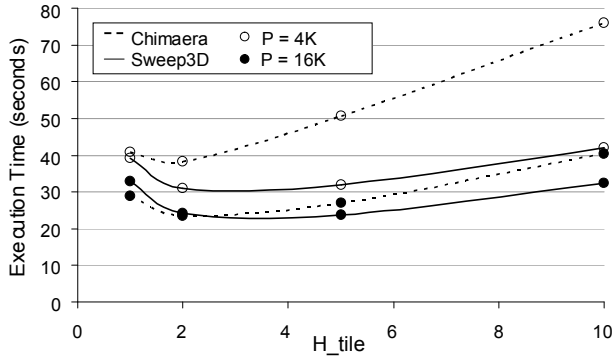
## 5 Results: Application and Platform Design

In this section we apply the plug-and-play wavefront application model to illustrate its utility in evaluating application design and configuration (Section 5.1), hardware platform procurement questions such as platform sizing and configuration (Section 5.2), hardware platform design alternatives, specifically the number of cores per node (Section 5.3), application bottlenecks (Section 5.4), and application re-design to alleviate one of the bottlenecks (Section 5.5). We illustrate these applications for the two particle transport benchmarks, Chimaera and Sweep3D, noting that the model can be applied in a similar manner to LU or any other wavefront benchmark or production code of interest. The applications illustrate the versatility of the analytic model in supporting the rapid evaluation of a number of system configuration and design alternatives.

Throughout the results, we evaluate Chimaera with a problem size of  $240^3$  cells, which is the largest current cubic problem size available as part of the benchmark. We evaluate Sweep3D with two problem sizes of interest to LANL [1]:  $10^9$  cells and 20 million cells. For both problem sizes of Sweep3D, we set the number of angles,  $m_{mo}$ , to six. The Chimaera code requires 419 iterations to complete a time step for the problem provided with the benchmark. Unless otherwise noted, we set the number of iterations per time step in Sweep3D to 120 which we anticipate will be more representative of many actual particle transport simulations than the default value of 12.

### 5.1 Application Design: $H_{tile}$

As shown in the LogGP model (Table 5) and in previous studies of Sweep3D, the height of a tile or  $H_{tile}$  is a key



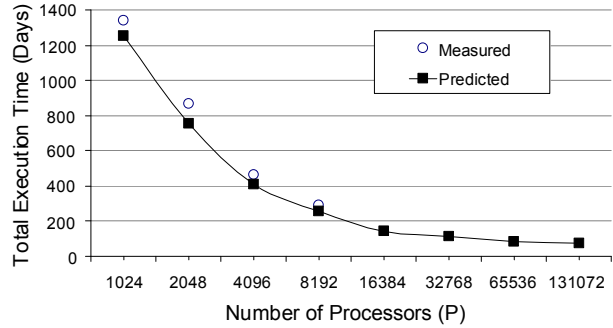
**Figure 5: Execution time vs.  $H_{tile}$**   
(Chimaera  $240^3$ , Sweep3D 20M cells, 480 Iterations)

configuration parameter. A larger value of  $H_{tile}$  leads to a larger ratio of computation to communication, as shown in equations (r1a) and (r1b). This leads to longer pipeline fill times as shown in equations (r2a) and (r2b), but also to lower communication costs because the communication overhead ( $o$ ) and latency ( $L$ ) occur less frequently as shown in equation (r4).

A key software configuration question that is easily addressed using the model is what value of  $H_{tile}$  to use for a given application code, problem size, and number of processors, in order to achieve minimum execution times. Figure 5 shows the execution time per time step vs  $H_{tile}$  for Chimaera and Sweep3D on the  $240^3$  and 20 million problem sizes, respectively. For each benchmark and problem size, we provide a curve for a small system configuration (4096 processors) and the maximum number the problem can practically run on (16K processors). In each case,  $H_{tile}$  in the range of 2, 4 or 5 minimizes the execution time. Results for Sweep3D with the  $10^9$  problem size on 16K - 128K processors (omitted to conserve space), also show that  $H_{tile}$  in the range of 2 to 5 minimizes execution time. In contrast, previous work evaluating Sweep3D on the SP/2 that has higher communication overhead and latency, found  $H_{tile}$  in the range of 5 to 10 (i.e.,  $mk = 10$  and  $mmi/mmo = 0.5$  or 1) minimized execution time [1].

For clarity in Figure 5, we have omitted the curve for Chimaera with problem size  $240 \times 240 \times 960$  – another problem size of interest to AWE – for which  $H_{tile} = 2, 4$  or 5 provides a 20% improvement in execution time (compared with  $H_{tile} = 1$ ) on 16K processors, which is similar to the improvement in Sweep3D execution time for the  $10^9$  cell problem size. We note that the architects of Chimaera are implementing a parameter that corresponds to  $H_{tile}$  so that these projected lower execution times will be achievable. Our results illustrate the ability of the model to rapidly evaluate software design modifications in order to determine whether the implementation effort is justified.

We use  $H_{tile} = 2$  for the results in the remainder of this section, noting that in some cases the execution time will be



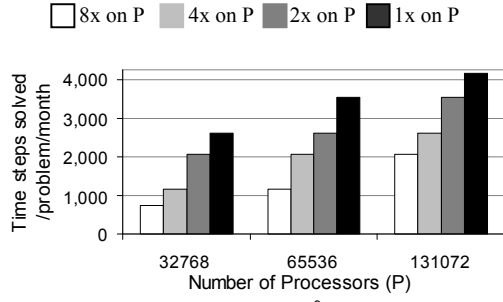
**Figure 6: Execution time vs. System Size**  
(Sweep3D,  $10^9$  cells  $10^4$  time steps,  $H_{tile} = 2$ )

slightly lower if  $H_{tile}$  is set to 4 or 5. One further point of interest in Figure 5 is that on 16K processors the execution time for one iteration of Sweep3D with a problem size of 20M cells (with 480 iterations to complete the time step) is very similar to the execution time of Chimaera with a problem size of  $240^3$  cells (requiring 419 iterations). These two benchmarks perform different processing. For example, Sweep3D computes six angles while Chimaera computes ten angles. Of interest is that the codes have qualitatively similar processing costs. Hence, in the remainder of this section we consider particle transport and platform design issues using the  $240^3$  problem in Chimaera and the  $10^9$  problem in Sweep3D.

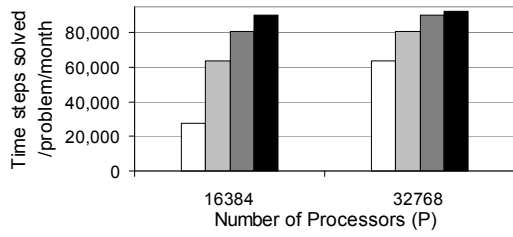
## 5.2 Platform Sizing and Configuration

For a given particle transport problem size of interest, increasing the number of processors decreases execution time, but with diminishing returns. Model results in Figure 6 illustrate this for Sweep3D for the  $10^9$  problem. In this figure, we show the execution time for  $10^4$  time steps, when the code uses both processors in each dual-core XT4 node. Single-core vs multi-core performance is evaluated in Section 5.3. The results in Figure 6 assume Sweep3D simulates 30 energy groups[1], which implies a 30-fold increase in execution time compared with the execution time for a single energy group. We use these values of interest to LANL to illustrate the system sizing question in the context of production problems of interest to the organizations that own the benchmarks.

Figure 6 also provides measured code execution times for the numbers of nodes that are available in the ORNL XT4, scaled to the 30 energy groups and 10,000 time steps. Note that we obtained an error in the order of 10% in the predicted execution times, due to the communication abstractions in the model. These results illustrate that the projected execution times (with these abstractions) are qualitatively correct and sufficiently accurate to support accurate decisions concerning how many processors should be allocated to a given particle transport simulation. Similar



(a) Sweep3D 10<sup>9</sup> Cells



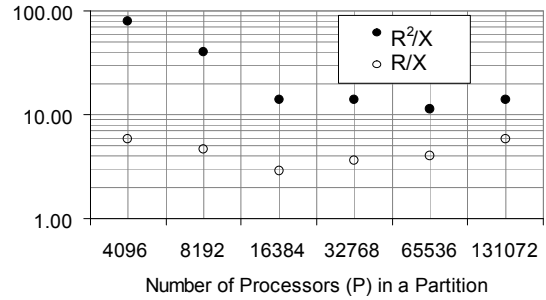
(b) Chimaera 240<sup>3</sup> Cells

**Figure 7: Throughput vs. Partition Size**

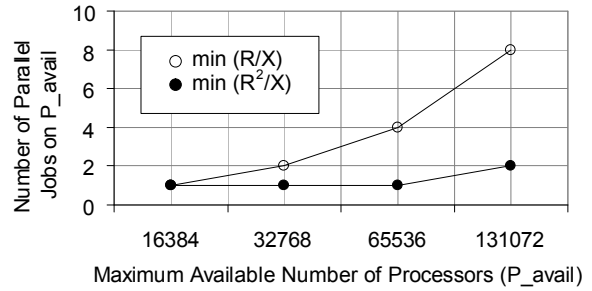
results for Chimaera on the 240<sup>3</sup> problem are omitted to conserve space.

As shown in the figure, the trade-off in execution time versus the number of nodes is complex. There are diminishing but perhaps still significant returns as the number of processors increases beyond 16K. A given user requiring nearly the minimum possible execution time may determine that the desired system size is 64K or 128K cores. On the other hand, due to the diminishing returns from 32K processors to 64K processors, another user may want to trade-off the execution time of one problem on 64K processors against solving two 1B problems simultaneously, each on half of the 64K processors.

We provide results for evaluating this trade-off in Figure 7. The black bars in Figure 7 provide the number of time steps solved per month when a single problem executes on the given number of processors. The other bars in Figure 7 show the number of time steps completed per month by each of 2, 4, or 8 particle transport simulations that are executed in parallel on equal-size partitions of the given number of processors. For example, the dark gray bars show the number of time steps solved per month in each of two problems solved when the given number of processors is partitioned in half. Note that when two 1B Sweep3d problems each run on half of 32K processors, approximately 2000 time steps are solved per month in each of the problems. This means that approximately five months or 150 days are required to execute 10,000 time steps. Figure 6 also shows that approximately 150 days are required to simulate the 10,000 time steps on 16K processors. Hence, Figure 7 is another



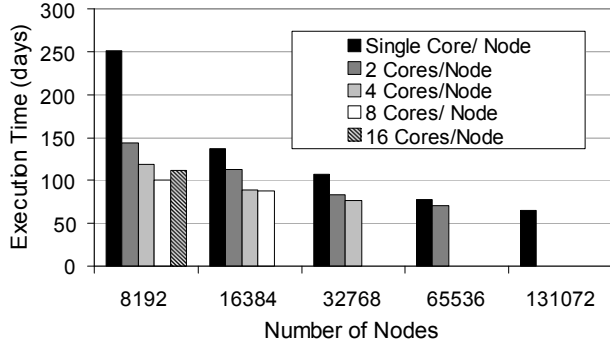
**Figure 8: Optimizing Partition Size**  
(Sweep3d 10<sup>9</sup>, P = 128K)



**Figure 9: The Optimized Number of Parallel Simulations** (Sweep3D 10<sup>9</sup> cells)

way to view the performance vs system size. In the case of 128K processors in Figure 7(a), two parallel simulations execute at 7/8 the rate of a single simulation, providing perhaps an attractive alternative for some users. The results in Figure 7 illustrate that a given site may want to consider the total number of simulations that need to be run when making procurement decisions and/or when allocating system resources to particle transport simulations.

It is desirable to achieve a good trade-off between minimizing the execution time for a single simulation ( $R$ ) by running it on as many processors as possible, and maximizing the total number of simulations that complete per unit time ( $X$ ) by partitioning the available processors so that simulations run in parallel. It is possible to quantify this trade-off, as illustrated in Figure 8 for the 1B problem of Sweep3D. Two curves are plotted as a function of partition size for parallel simulations on 128K cores. When the partition size is 32K cores, four 1B simulations are run in parallel. The lower curve is the value of  $R/X$ , the ratio of time to complete each 1B particle simulation divided by the number of simulations that complete per time  $R$ . This ratio is minimized when the partition size is 16K processors and thus 8 simulations are run in parallel. The upper curve is  $R^2/X$ , which places greater emphasis on minimizing the execution time for each simulation, and is optimized at 64K processors per simulation. A given site or user can compare these optimized partitions with the results in Figure 6 and Figure 7 to arrive at a decision about how to configure the



**Figure 10: Execution Time on Multi-core Nodes**  
(Sweep3D  $10^9$  cells,  $10^4$  time steps)

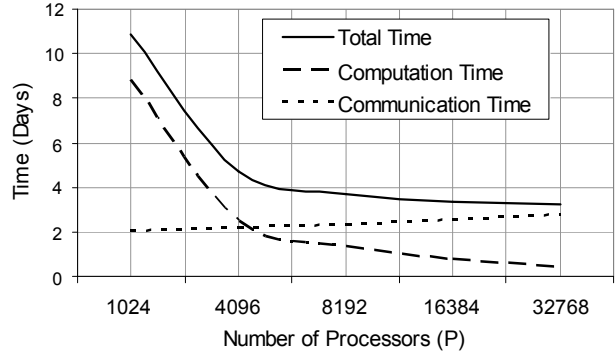
system. Figure 9 provides the optimal number of simulations to be run in parallel on 128K processors as well as for smaller platforms, for each of the two criteria considered in Figure 8.

Figure 7(b) provides results for the  $240^3$  problem size in Chimaera on a maximum of 32K processors. For the  $240^3$  problem size, running a single problem on 32K processors provide negligible improvement in execution time over running two problems in parallel on half the processors. On the other hand, partitioning 16K nodes into four partitions of size 4096 nodes yields better than a 50% reduction in execution time per problem compared with sixteen partitions of size 1024 processors. A given user can determine the ideal trade-off, but 4K - 16K processors will be the desired partition size for many if not most users. These curves again illustrate the value of the model for aiding in system procurement and configuration decisions.

### 5.3 Platform Design: Multi-core Nodes

We next examine the platform design issue of how many cores per node would be desirable for the important class of large particle transport simulations that make up a large fraction of the workload at places such as LANL and AWE. These results are obtained using the model extensions provided in Table 6, which assume a shared bus architecture within each node but can easily be modified for other node architectures. Results are provided here to illustrate the utility of the model in providing insights into the question of interest.

Figure 10 provides the execution time for a 1B particle transport simulation versus the number of nodes on the platform, and for various possible numbers of cores per node ranging from one core per node to sixteen cores per node. Because there are diminishing returns when the simulation runs on increasing numbers of nodes (with one core per node) there are also diminishing returns for increasing the number of cores per node. Note also that in these results, two cores on a given number of nodes (e.g., 64K nodes) provide slightly better execution time than four cores on half the nodes (e.g., 32K nodes) due to the assumed shared bus



**Figure 11: Cost Breakdown (Chimaera)**

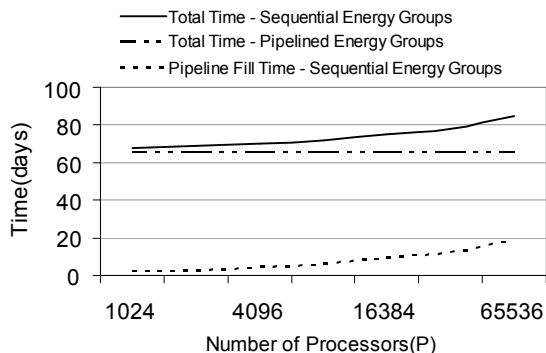
architecture.

If the target execution time is approximately the execution time on 64K single-core nodes, then the figures shows that this performance can be nearly achieved with 32K dual-core nodes or 16K quad-core nodes.

An 8K-node system with 16 cores per node has the same total number of cores as 32K quad-core nodes (and thus twice as many cores as a system with 32K dual-core nodes), but execution time is degraded due to contention for the shared bus. However, if the 16-core node is provisioned with a separate shared bus, shared memory, and NIC for each group of 4 cores, then the execution time on the system with 8192 nodes would be the same as the execution time for the 32K quad-core nodes. This is perhaps an even more viable multi-core design for particle transport simulations. These results again illustrate the value of the model in examining various system design and configuration questions.

### 5.4 Application Bottlenecks

In our final set of experiments, we illustrate the use of the model to understand application bottlenecks which are not readily measured when running or simulating the actual code. Figure 11 provides the total execution time for the  $240^3$  problem in Chimaera, as well as the breakdown of total critical path time into computation and communication components, as a function of the number of processors that the particle transport code runs on. The communication component of the total execution time is derived from the *Send*, *Receive*, *TotalComm* and  $T_{allreduce}$  execution time terms in the model. The computation component is the rest of the total execution time. Note that the point at which communication dominates the total execution time is the point at which increasing the number of processors provides greatly diminished reduction in the total execution time. Since communication of the boundary values is required for the simulations, the only opportunity for improving the observed communication bottleneck is to further improve the inter-core communication efficiency. The model can also be used by system architects to project execution



**Figure 12: Pipeline Fill Redesign**

(Sweep3D  $4 \times 4 \times 1000$  cells/processor, 30 Energy groups,  $10^4$  time steps)

times for such communication improvements.

Figure 12 shows the total execution time for a fixed processor problem size as a function of the number of processors, and also the amount of total execution time that is due to pipeline fill at the beginning of each sweep. The pipeline fill time during each sweep is computed from equations (r3a) and (r3b). The results are for the case of  $10^4$  time steps and 30 energy groups.

### 5.5 Sweep Structure Re-design

The pipeline fill overhead might be reduced by the following Sweep3D re-design. Instead of performing all eight sweeps for the first energy group and iterating to convergence before solving the next energy group, we could pipeline the solution of the energy groups by performing the first two sweeps for all 30 energy groups followed by sweeps 3 and 4 for all 30 energy groups, and so forth.

Pipelining the energy groups might require more iterations to reach convergence. We can project the execution time if no additional iterations are needed by modifying the model input parameters. In other words, a total of 240 sweeps are required per iteration, with  $n_{diag} = 2$  and  $n_{full} = 2$ . The projected execution time with these parameters is also given in the figure, showing that nearly all of the pipeline fill overhead is eliminated. The projections can be made for an increased number of iterations to reach convergence, if the user can provide knowledgeable estimates of this increase. Again these results illustrate how the model can be used to rapidly gain insight into software bottlenecks and the impact of possible software modifications, in order to determine where implementation effort might profitably be placed.

## 6 Conclusions

This paper has developed and applied a plug-and-play analytic model for pipelined wavefront applications on regular orthogonal grids. The re-usable analytic model requires only a few input parameter changes to obtain a model of any

given wavefront application on a parallel architecture with multi-core nodes. It also supports the assessment of various possible design changes in a given wavefront application or parallel platform. The model was validated on a large Cray XT4 system for three key wavefront applications – namely LU, Sweep3D and Chimaera. The model has predictive error under 10% for configurations in which computation time dominates communication time.

As part of this work we have developed and validate highly accurate MPI send, receive and all-to-all communication models for a Cray XT4 system. Platform parameter values derived from those models show that the XT4 communication software and hardware are highly optimized.

We applied the plug-and-play models of Chimaera and Sweep3D to determine optimized application configurations ( $H_{tile} = 2$  to 5), platform procurement decisions using quantitative metrics, as well as platform and application design changes for large particle transport simulations. Model projections show, for example, that increasing the number of cores per node to more than 4 on a single bus, results in diminishing returns for particle transport codes.

## Acknowledgments

This research used resources at the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. DOE under Contract DE-ASC05-00OR22725. We acknowledge Andy Herdman and Ben Ralston regarding access to the Chimaera code, Patrick H. Worley and the ORNL PEAC project for access to the Cray XT4, and Howard Pritchard at Cray and David Sundaram-Stukel for their valuable comments on this work.

## References

- [1] A. Hoisie, H. Lubeck, and H.J. Wasserman. Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures using Multidimensional Wavefront Applications. *Int. J of High Performance Computing Applications*, 14(4):330–346, Winter,2000.
- [2] The message passing interface (mpi). <http://www-unix.mcs.anl.gov/mpi/>.
- [3] D. Sundaram-Stukel and M. K. Vernon. Predictive Analysis of a Wavefront Application Using LogGP. In *PPoPP '99: Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 141–150. ACM Press, 1999.
- [4] D.J. Kerbyson, A. Hoisie, and H.J. Wasserman. A Comparison Between the Earth simulator and Alphaserver Systems using Predictive Application Performance Models. *Computer Architecture News (ACM)*, December 2002.
- [5] A. Hoisie, O. Lubeck, H. J. Wasserman, F. Petrini, and H. Alme. A General Predictive Performance Model for Wavefront Algorithms on Clusters of SMPs. In *ICPP '00: Proceedings of the Proceedings of the 2000 International*

- Conference on Parallel Processing*, page 219. IEEE Computer Society, 2000.
- [6] M. M. Mathis, N. M. Amato, and M. L. Adams. A General Performance Model for Parallel Sweeps on Orthogonal Grids for Particle Transport Calculations. Technical report, Texas A&M University, 2000.
  - [7] M. Yarrow and R. Van der Wijngaart. Communication improvement for the LU NAS parallel benchmark: A model for efficient parallel relaxation schemes. Technical Report NAS-97-032, NASA Ames Research Center, November 1997.
  - [8] A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin. A performance comparison through benchmarking and modeling of three leading supercomputers: blue gene/l, red storm, and purple. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 74, New York, NY, USA, 2006. ACM.
  - [9] A. Alexandrov, M. F. Ionescu, K. E. Schauer, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.
  - [10] Cray XT4 data sheet. <http://www.cray.com/products/xt4>.