PERFORMANCE MODELING OF DETERMINISTIC TRANSPORT COMPUTATIONS

Darren J. Kerbyson
Los Alamos National Laboratory
Performance and Architectures Laboratory, CCS-3
P.O. Box 1663, Los Alamos, NM 87545
djk@lanl.gov

Adolfy Hoisie
Los Alamos National Laboratory
Performance and Architectures Laboratory, CCS-3
P.O. Box 1663, Los Alamos, NM 87545
hoisie@lanl.gov

Shawn D. Pautz
Los Alamos National Laboratory
Transport Methods, CCS-4
P.O. Box 1663, Los Alamos, NM 87545
pautz@lanl.gov

Abstract In this work we present a performance model that encompasses the key characteristics of a Sn transport application using unstructured meshes. Sn transport is an important part of the ASCI workload. This builds on previous analysis which has been done for the case of structured meshes. The performance modeling of an unstructured grid application presents a number of complexities and subtleties that do not arise for structured grids. The resulting analytical model is parametric using basic system performance characteristics (latency, bandwidth, MFLOPS rate etc), and application characteristics (mesh size etc). It is validated on a large HP AlphaServer system showing high accuracy. The model compares favorably to a trace based modeling approach which is specific to a single mesh/processor mapping situation. The model is used to give insight into the achievable performance on possible future processing systems containing thousands of processors.

Keywords: Performance Evaluation, Performance Modeling, Unstructured-Meshes, Large-scale Systems.

1. Introduction

In this work we present the development and use of a performance model for an application code developed for solving the Boltzmann equation deterministically (Sn transport) on unstructured meshes. The unstructured mesh application that is modeled here is known as Tycho [12]. The problem is of great importance to the ASCI workload, hence other codes are also being under development for this purpose e.g. [14].

Unstructured meshes have several benefits over the use of structured meshes in terms of the calculations undertaken, but have significant extra overhead in terms of performance. Several important performance factors that can reduce the overall calculation efficiency of this type of computations on large-scale parallel systems are analyzed in this paper.

The algorithms employed in deterministic Sn (discrete ordinate) computations fall in a class generically named wavefront techniques. In a nutshell they utilize an iterative approach using a method of "sweeping" [3]. Each spatial cell in a mesh is processed in a specified order for each direction in the discrete ordinates set. The wavefronts (or sweeps) are software pipelined in each of the processing directions. Wavefront algorithms exhibit several interesting performance characteristics, related to the pipelined nature of the wavefront dynamics. These include a pipeline delay across processors for a sweep, and a repetition rate of both computation and communication in the direction of the sweep. In the case of a structured mesh a high efficiency of calculation can be achieved as all active processors perform the same amount of work, and communicate the same sized boundary data [3].

Efforts devoted to the performance analysis of Sn transport date back many years. Research has included the development of performance models as a function of problem mesh and machine size [8]. More detailed performance models have been developed that also include inter-processor communication, and SMP cluster characteristics [3–4]. However, these all considered an underlying structured mesh.

The key contribution of this paper is the development of an analytical performance model of Sn transport on unstructured meshes. This is the first performance model for Sn Transport on unstructured meshes. The model encapsulates the main performance characteristics parameterized in terms of mesh size and system configuration. Two analytical models are considered, one in a general form and one in a mesh specific form. A third "trace" model, similar to Dimemas [2], is included to compare the different models.

The approach that we take for modeling is application centric. It involves understanding the processing flow in the application, the key data structures, and how they use and are mapped to the available resources. From this a performance model is constructed that encapsulates the key performance char-

acteristics. This approach has been successfully used on an adaptive mesh code [6], a structured mesh transport code [3], and a Monte-Carlo particle simulation code [9].

The analytical model developed here is shown to have reasonable accuracy through a validation process on a HP AlphaServer parallel machine. The general model is able to add insight into the achievable performance that could be obtained on hypothetical future architectures and hence indicating the efficiency and sizes of mesh that could be processed. Specifically we use the model to explore expected achievable performance on future large-scale systems which may be capable of a 100 tera-flops prior to their availability.

The paper is organized as follows. In Section 2, the Sn transport calculation is detailed and comparisons between its operation on structured and un-structured meshes are made. In Section 3 the key characteristics of the processing are described which are used in the development of the performance models in Section 4. The models are validated in Section 5 on a number of unstructured meshes, and are used to explore the performance on future architectures that cannot currently be measured in Section 6.

2. Overview of Sn Transport Algorithms

2.1 The method of sweeping

Two examples are depicted below that illustrate the method of sweeping used within an Sn transport calculation for a structured and an unstructured mesh. In both cases, the calculation dependencies in the direction of the sweeps are clearly shown.

Structured meshes In three-dimensions, each sweep direction can be considered to originate in one of the 8 corners ("octants") of the spatial domain. Within each octant, the ordering of cell processing is identical. Figure 1 shows the first six steps of two separate sweeps at different angles for a two-dimensional spatial domain, originating from different octants. The edge of the sweep corresponds to a wavefront and is shown as black. It requires the grey cells to have been processed in previous steps. The same operation can take place in three dimensions resulting in a wavefront surface. The wavefront propagates across the spatial domain at a constant calculation velocity since the time needed to process a cell is constant. This processing algorithm as developed in [8], uses direct indexing of the spatial mesh as the cell processing order is deterministic for each sweep direction.

Unstructured meshes An example two-dimensional unstructured mesh is depicted in Figure 2. Two sweep directions are again used to illustrate the processing over a total of six steps. As before, the cells being processed in the

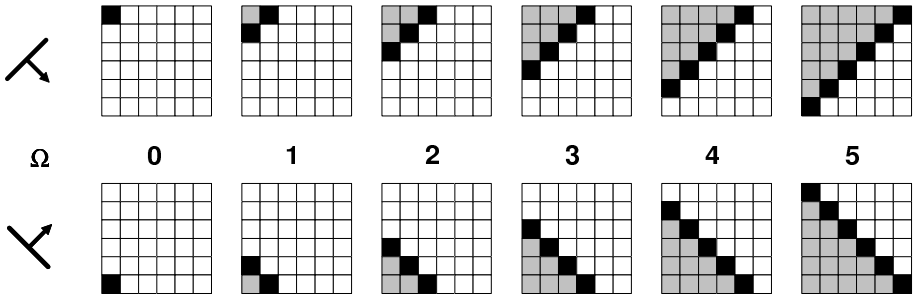

Figure 1. Example sweep processing on a 2-dimensional structured mesh.

current step are shown as black and require the previously calculated grey cells. The ordering of cell processing is direction dependent. The incoming data to a cell are determined by the mesh geometry, and it is apparent that the propagation speed of the wavefronts also varies with direction. The same situation occurs in three-dimensional geometry, only with the mesh being composed of tetrahedrons, pyramids, or prisms.

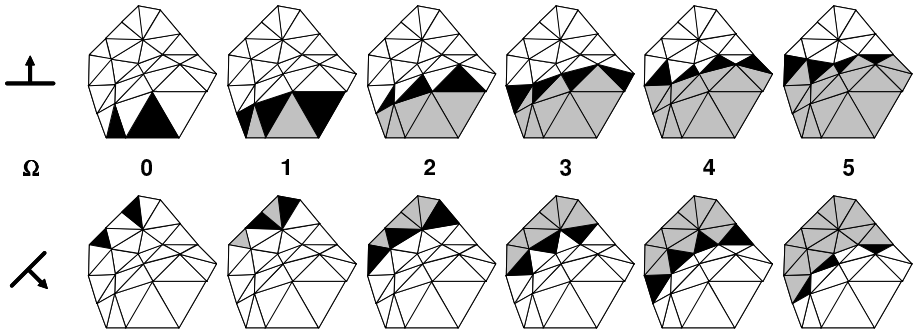

Figure 2. Example sweep processing on an 2-dimensional unstructured mesh.

2.2 Sweeping in Parallel

Parallel wavefront computations exhibit a balance between processor efficiency and communication cost [3]. Faster wavefronts, generated by a data decomposition leading to small subgrid sizes per processor, introduce higher communication costs but result in high processor utilization. The opposite holds true for slower moving sweeps due to larger subgrid sizes. In order to optimize wavefront dynamics, Sn transport applications typically utilize blocking of the spatial subgrid and/or blocking of the angle set.

Important performance considerations in parallel wavefront applications, which need to be captured in the model, are as follows:

pipeline effects a processor is inactive until a sweep surface enters cells in that particular processor. However, multiple sweeps are active at any given time in the processor array. Overlap exists between computation and communication within each sweep, and across the active sweeps.

communication costs for boundary data transfer.

load balancing of the number of cells processed on each PE in a step. This applies to wavefronts on unstructured meshes only as an equal number of cells are processed in each step on each PE for a structured mesh.

In order to analyze these effects, the processing that takes place on both structured and unstructured meshes is illustrated below.

Structured meshes In codes such as Sweep3D which performs an Sn transport computation on a structured meshes, the 3-D mesh is mapped onto a 2-D processor array such that each processor has a column of data which is further blocked in its third dimension. The processing is effectively synchronized after the first sweep has moved across the PE array resulting in all processors being active. The processing involved in each sweep is dependent on the block size (a known constant). Thus one diagonal of processors will be processing one sweep while the previous diagonal is processing the next sweep and so on (Figure 3). The direction of sweep travel is the indicated by Ω with inter-processor communications shown by arrows. It has been shown that the cost of performing this calculation on a structured mesh conforms to a pipeline model [3]:

$$T_{Total} = (P_x + P_y - 1)(T_{CPU} + ST_{msg}) + (N_{sweep} - 1)(T_{CPU} + 4T_{msg}) \quad (1)$$

where P_x and P_y are dimensions of the processor grid, Nsweep is the number of sweeps, T_{CPU} and T_{MSG} is the time to process a cell block, and the time to communicate a message respectively. The first part of this equation corresponds to the length of the pipeline and the second part it the number of repetitions once the pipeline is filled.

Unstructured meshes The processing on an unstructured grid follows the same dependency rules as above, but the mesh partitioning is typically done in all 3 dimensions. An example 2-D partitioning of an unstructured mesh is shown in Figure 4. The communications between processors are shown by arrows, and a simplified propagation of the sweep in the indicated direction is shown by the grey lines. Tycho actually enables sweeps in all directions to

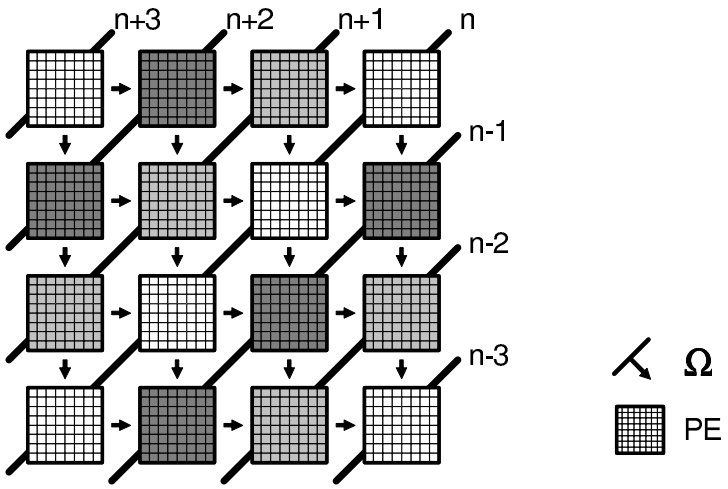

Figure 3. The pipeline processing of sweeps in parallel.

commence simultaneously. Thus each cell is processed for each sweep angle set whilst still taking into account the dependencies in each of the sweep directions. The unit of processing work can be considered as a single cell-angle pair.

The sweep processing on the unstructured mesh can also be blocked - a number of cell-angle pairs can be processed per step. However, this can result in processor inefficiency down the pipeline - processor idleness can occur due to boundary data between processors not being a constant size over steps.

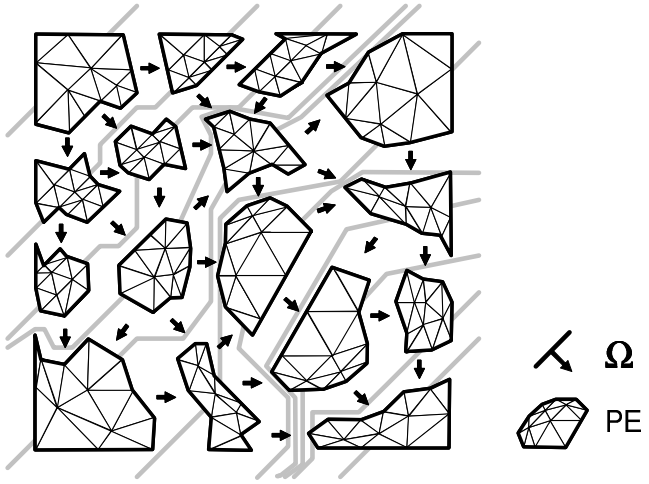

Figure 4. Example partitioning and sweep flow on a 2-D unstructured mesh.

An approach to maximize processor utilization is to consider each cell-angle pair as a task, and to assign each a priority. Tasks are placed in a priority queue with the highest priority ones being processed first. The key to this approach is in the assignment of priorities. In general, processor boundary cells will be of high priority (boundary cells need to be processed in order for the sweeps to travel down the pipeline), and cells upstream of the boundary cells (i.e. those needing to be processed prior to the boundary cells) are given an even higher priority. This scheduling approach attempts to maximize cell boundary production in order to keep the processors downstream in the pipeline busy. Several different heuristic scheduling schemes have already been analyzed using Tycho [12].

3. Key Processing Characteristics

The key processing characteristics in Tycho are: mesh partitioning, pipeline processing, processor utilization, and strong scaling. An understanding of these factors is required in order to formulate the performance model. Two situations are considered in this analysis - a general case when no knowledge of the mesh exists (except for the total number of cells), and a mesh specific case in which the mesh is inspected for detailed information which can be used in the model.

3.1 Mesh partitioning

The partitioning is not done within Tycho, rather a suitable tool such as Metis [5]is utilized. This mesh partitioner aims to produce equally sized partitions while minimizing boundaries. Such an optimal partitioning of the mesh in general would keep the work across PEs constant and minimize the communication cost. However, due to the pipeline processing and load-balancing characteristics in Tycho, this partitioning may not be optimal.

Tycho also utilizes a 3-dimensional processor decomposition. This contrasts with Sn transport on structures meshes which typically utilizes a 2-dimensional processor decomposition.

For such a 3-D partitioning, the number of cells per partition can be taken to be $E_p = N/P$ where N is the number of cells in the mesh, and P is the number of PEs (which is equal to the number of partitions). In the general case each 3-D partition would ideally have six nearest neighbors each with a boundary size of $Ep^{2/3}$ cells.

When considering a specific mesh the number of cells per processor, E_p, and a vector of neighbor communications, $N_c(s,p)$, and average communication sizes, $N_s(s,p)$, per step for each PE can be obtained by inspection once the mesh has been partitioned and the work scheduled. Note that the vectors are defined for each step, s, and for each PE, p.

3.2 Pipeline processing

Sweeps in all directions start simultaneously in Tycho. The first cell- angle pairs processed are those that lie on the boundary of the spatial mesh which have no inflows in the sweep direction. This corresponds to nearly all boundary elements. The sweeps thus generally start from the surface of the mesh and work their way to the centre before propagating out the opposite side. The dynamics of the pipeline is determined by the pipeline length and by the amount of computation done on each mesh partition. The pipeline length is determined by the number of stages in the propagation of the sweep from one side of the mesh to another. In 2-D the number of gray lines in Figure 4 would represent the number of stages. In general, given an ideal 3D partitioning, the pipeline length is given by:

$$P_L = (P_x - 1) + (P_y - 1) + (P_z - 1) \tag{2}$$

where P_x, P_y and P_z are the number of PEs in each of the three dimensions respectively. The total work done, or the total number of cell-angle pairs processed, on each mesh partition in an iteration is equal to:

$$W_p = E_p * N_\Omega \tag{3}$$

where N_Ω is number of sweep directions. For a specific mesh, the pipeline length, P_L can be obtained by inspection of the mesh after the partitioning has been performed and is equal to the maximum number of PEs traversed in any sweep direction. The total amount of work done per partition remains as above.

3.3 Processor utilization

Each step in Tycho consists of three stages: do the work at the top of the priority queue, send boundary data to PEs downstream, and receive boundary data from upstream PEs. The amount of work done in a step is determined by an input parameter $MCPS$ (MaxCellsPerStep) and specifies the maximum number of cell-angle pairs that can be processed in a step in each processor. Thus $MCPS$ effectively represents a blocking factor.

The processing situation is complicated by the processing dependence between upstream and downstream cells in the sweep directions. This dependence may lead to downstream PEs waiting for the upstream PEs to send the necessary boundary information. There will almost always be a degree of inefficiency in this operation and processors will be starved of work waiting for the results from other PEs. It is interesting to note that for the case of structured meshes, the work on each PE is equal throughout and thus there processors are fully utilized once the pipeline is filled. To quantify this inefficiency, the metric of Parallel Computational Efficiency, PCE [12], is used:

$$PCE = \frac{W_p}{\sum_{i=1}^{\#steps} \max_p(\|work(P,S)\|)} \quad (4)$$

where $work(P,S)$ is the number of cell-angle pairs processed in step S on processor P, and W_p is the total number of cell-angle pairs processed on each processor in an iteration. PCE represents the fraction of the maximum number of cells that are processed in all steps in an iteration. When $PCE = 1$ the efficiency is 100% - this can only occur on a small processor run (typically < 9 PEs). The lower the value of PCE, the greater the inefficiency.

A value for PCE can be obtained for a specific mesh after its partitioning and before the sweep execution. The number of steps required to perform the total number of cell-angle pairs per PE (excluding the pipeline effect) is given by:

$$\frac{W_p}{(MCPS * PCE)} \quad (5)$$

In the general case, i.e. without the inspection of the mesh, a value of the PCE has to be assumed - possibly based on experience from prior meshes. This assumption can be inaccurate reflecting the tradeoff between generality and accuracy always present in performance modeling work.

3.4 Strong Scaling

Typical Tycho runs are executed in a strong scaling mode - the input mesh size is constant and thus partitions become smaller on larger processor counts. This is easily incorporated into a modified expression for Wp. However, for the case of strong scaling, the memory hierarchy effects have to be carefully considered. For instance when a mesh partition becomes small enough to fit in cache the performance will be better than if main memory has to be accessed.

Figure 5 shows the computation time per cell for different meshes and partition sizes on an 833MHz Alpha EV68 processor with 8MB L2 cache. There are clearly three regions evident: when the partition does not fit into L2 cache (right hand plateau), when the mesh fits into L2 cache (left hand plateau), and when partial cache re-use occurs (middle region). There is some variation between meshes in this analysis due to the different memory access patterns and hence the actual cache reuse. It can be seen that a good approximation to this memory hierarchy performance can be encapsulated in a piece-wise linear curve. In general however, we are interested in large meshes - those that unfortunately will not exhibit cache re-use.

4. Performance Models for Tycho

Three performance models for Tycho are described below using the key characteristics described in Section 3. The first two, the General Model (GM) and

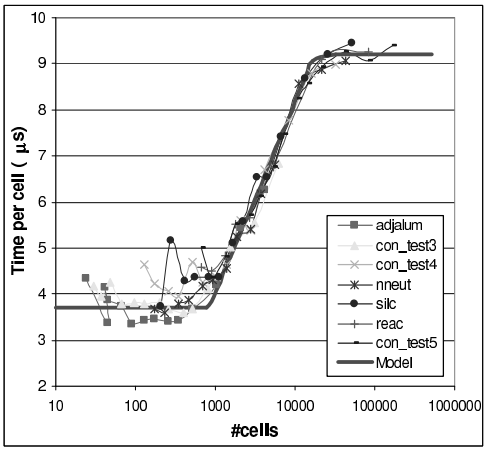

Figure 5. Processing time per cell on different mesh and partition sizes.

Mesh Specific Model (MSM) are analytic models. The GM uses the characteristics discussed in Section 3 for a general mesh - i.e. without using detailed knowledge on the mesh partitioning. The MSM on the other hand uses the knowledge of the partitioned mesh. These first two models are described together in Section 4.1 below. The third is a Trace Model (TM) that is based on an analysis of a communication trace obtained at run-time for a specific mesh on a specific processor count. The TM is described in Section 4.2.

4.1 Analytical performance models

In the analytical performance models we assume that the three stages of a Tycho step are distinct and do not overlap - those of computation, blocking sends, and blocking receives. This is a simplification as there will be a degree of overlap between computation and communication. However, the amount of overlap is assumed to be small, and as will be seen from the error analysis in Section 5, is a reasonable assumption. The runtime for an iteration of Tycho can be modeled as:

$$T_{iter} = \left(\sum_{S=1}^{\#steps} \max_{P}(Work(P,S)) \right) .T_{Elem}(\frac{N}{P}) + \sum_{S=1}^{\#steps} \max_{P} \left(\sum_{C=1}^{||N_c(S,P)||} T_{comm}(N_c(S,P,C), N_s(S,P,C)) \right) \quad (6)$$

where the first term represents computation and the second term represents communication. The model parameters are as follows:

$\#steps$	is the number of steps in an iteration
$Work(P,S)$	number of cell-angle pairs processed on processor P in step S
$N_c(S,P,C)$	destination PE for communication C in step S on processor P
$N_s(S,P,C)$	is the size of communication C in step S on processor P
$\|N_c(S,P)\|$	number of communications from processor P in step S
$T_{Elem}(x)$	time to process a cell-angle pair given x cells mapped to a PE.
$T_{comm}(x,y)$	time to communicate a message of size y bytes to processor x

Assuming no overlap between communication and computation, as stated earlier, in equation 6 the first term represents computation time and the second communication time. Due to the wavefront nature of these algorithms, within one wavefront, overlap exists between the computation of the angle-cells. The wavefront will be ready to propagate downstream as soon as the work on the largest subgrid contained in the wavefront will be completed. Hence the "max" function contained in the first term of the equation. Similarly, the second term is a sum of the non- overlapped communication steps for all wavefronts.

The model as formulated in equation 6 represents both the MSM and GM. However many of the parameters in the model are substantially different between the two. The parameters of $T_{Elem}()$, and $T_{comm}()$ are hardware specific and remain the same. A two-parameter, piece-wise linear model for the communication is assumed which uses the Latency (L_c) and Bandwidth (B_c) of the network communication.

$$T_{comm}(D,S) = L_c(S,D) + S.\frac{1}{B_c(S,D)} \tag{7}$$

where L_c is the communication latency, B_c is the communication bandwidth, D is the message destination PE, and S is the message size.

In the MSM the parameters steps, $Work()$, $N_c()$, and $N_s()$ represent actual time histories of the work and communications done through all the steps in an iteration. These time histories are obtained by inspection after mesh partitioning and scheduling of the cell-angle pair tasks, prior to actual processing in Tycho. They are specific to both the mesh and the processor count. This type of model tends to reflect the static behavior of the code while parameterizing the main dynamic attributes. A similar approach was successfully taken in the modeling of an adaptive mesh code [6].

In the GM the assumptions presented in Section 3 for the processing characteristics can be used to simplify the model into a general form. In the GM, the number of steps is given by

$$\#steps = (\frac{E_P * N_\Omega}{MCPS * PCE}) + (P_x - 1) + (P_y - 1) + (P_z - 1) \tag{8}$$

where E_p is the number of cells per PE (assumed constant at N/P), $MCPS$ is the MaxCellPerStep (input parameter to Tycho), $N\Omega$ is the number of sweep

directions, and PCE is the Parallel Computational Efficiency as described earlier. P_x, P_y and P_z are the number of processors in the logical x, y, and z dimensions respectively, as described in section 3.1. The first part of this equation represents the number of work steps and the second part represents the pipeline length (which will in general be an underestimate).

The work on each PE in each step is assumed a constant:

$$Work(P, S) = \min(MCPS, E_p * N_\Omega) \tag{9}$$

The number of communications per step on each PE, $\|N_c(S, P)\| = 6$, and average communication sizes per step on each PE are also assumed constant, $N_s(S, P, C) = \min(E_p^{2/3}, MCPS2/3) * 40$. Note that each boundary cell communicated consists of 40 bytes of data.

The communication time is subject to a contention in the communication network. Our experience on using Tycho, and other codes on clusters of SMPs, is that the main contention occurs on the number of out-of-node communications that occur simultaneously. For example with the fat-tree network of the Quadrics network [13], the number of communications that collide in higher levels of the fat-tree is low due to dynamic routing. The contention is taken into account by a multiplicative constant on the communication time, T_{comm}, which represents the number of out-of-node simultaneous communications.

4.2 A trace model

Traces that are obtained at run-time can capture the full computation / communication interaction of an application but is specific to a mesh and to a particular processor count. The trace can be effectively re-played in order to give a prediction. Such a trace modeling approach is not new and has been used in tools such as Dimemas [2], and PACE [7]. The approach taken here is similar to that of Dimemas and provides a comparison with the analytical models.

The traces used here contain three event types: 1) the number of cell- angle pairs processed in a step, 2) the communication sends, and 3) the communication receives. The communication events include details on the source and destination PEs as well as the message sizes. However, it should be noted that no timing information is stored in the trace file.

Timing information is produced by a Trace Model Evaluator (TME) developed at Los Alamos. The TME allows different prediction sub- models to be used to predict: the time to process a cell-angle pair, the communication costs, and the communication contention. Different sub- models may be used to predict component times for different systems. The TME effectively replays the trace file whilst accounting for the expected time taken by each event, and also resolving communication dependencies and possible contention in the communication network [10]. It also determines when bi-directional traffic occurs

between any two nodes, which can also reduce the effective communication bandwidth. The TME is a detailed evaluation method with a potential high accuracy but unfortunately loses any generality in the model. The evaluation for a given input trace file is specific to a mesh and processor count. As it will be seen below, the analytical models compare favorably with this more specific trace model.

5. Performance Model Validation

The three performance models are validated in this section on a HP AlphaServer ES40 64 node system. Each node in this system consists of 4 Alpha EV68 processors running at 833MHz each with an 8MB L2 unified cache. The nodes are interconnected using the Quadrics QSnet high speed network with Elan3 switching technology. The details of this architecture are not described in detail here but a good overview of its performance characteristics can be found in [13]. The hardware parameters for this system are listed in Table 1.

Table 1. Hardware parameters for the HP AlphaServer ES40 validation system.

$T_{elem}(E_p)(\mu s)$		$\begin{cases} 9.2 & E_p \geq 16Kcells \\ 1.8Ln(E_p) - 8.4 & 800 < E_P < 16K \\ 3.7 & E_P \leq 800 \end{cases}$
$L_c(S,D)(\mu s)$	intra-node ($D \leq 4$)	$\begin{cases} 12.7 & S < 64bytes \\ 12.8 & 64 \leq S \leq 256 \\ 30.3 & 256 < S \leq 8192 \\ 25.7 & S > 8192 \end{cases}$
	inter-node ($D > 4$)	$\begin{cases} 9.28 & S < 64bytes \\ 9.00 & 64 \leq S \leq 256 \\ 21.4 & S > 512 \end{cases}$
$1/B_c(S,D)(ns)$	intra-node ($D \leq 4$)	$\begin{cases} 0.0 & S < 64bytes \\ 24.0 & 64 \leq S \leq 256 \\ 9.0 & 256 < S \leq 8192 \\ 3.2 & S > 8192 \end{cases}$
	inter-node ($D > 4$)	$\begin{cases} 0.0 & S < 64bytes \\ 25.5 & 64 \leq S \leq 512 \\ 13.7 & S > 512 \end{cases}$

Four meshes are used in the validation as listed in Table 2. These represent small and medium sized meshes, resulting in small mesh partitions on the largest processor configuration considered.

Measurements and model predictions for each of the four meshes are shown in Figure 6. The application input parameter $MCPS$ was set at 512 in all cases.

Table 2. Meshes used in the validation.

Mesh	*#Cells*	*Description*
Nneut	43,012	Neutron well-logging tool and surrounding media
Silc	51,963	Computer Chip and packaging for radiation shielding
Reac	165,530	Reactor pressure vessel and surrounding cavity structures
Con_test5	168,356	Cube divided into approximately equal-sized elements

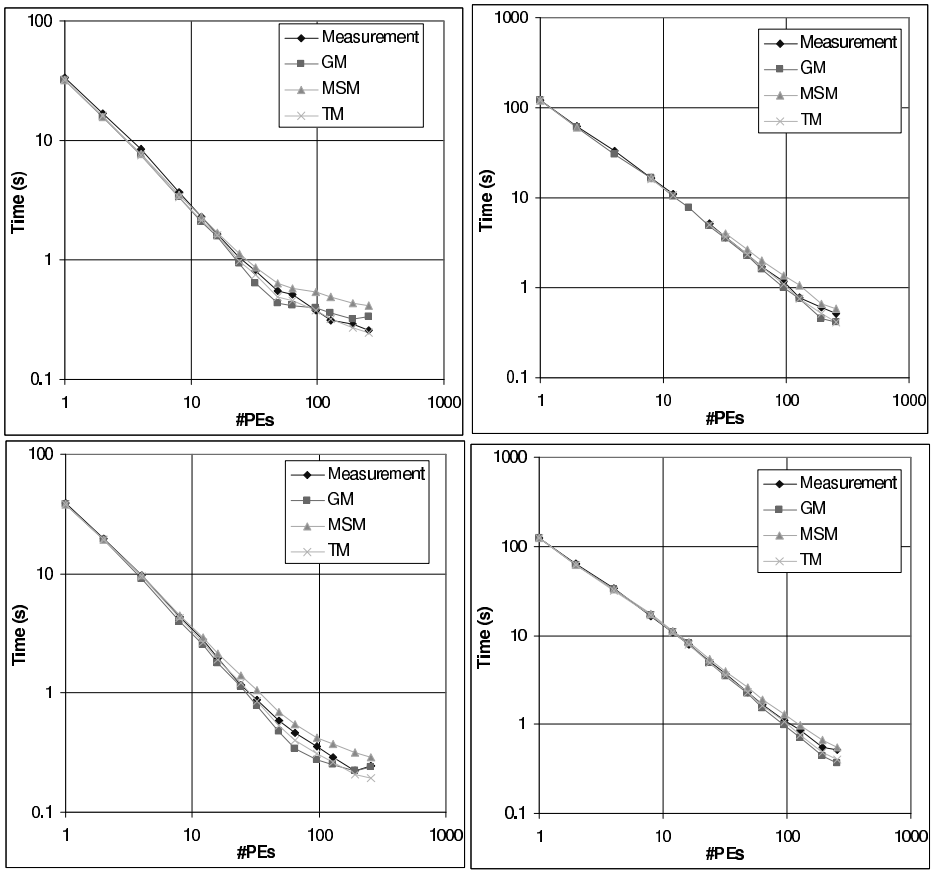

Figure 6. Model validation on a) Nneut, b) Silc, c) Reac, d) Con_test5.

A summary of the errors observed is listed in Table 3. It can be seen that all the models have a good prediction accuracy on small PE counts (case i., $P \leq 32$). This is expected since they all use the same model for computation, the dominant term in equation 6 for small processor counts. They do differ

however in their treatment of communication which becomes apparent on large processor counts (case ii., $P > 32$). The MSM will tend to over-predict since any possible overlap between computation and communication is not modeled. In contrast the GM will tend to under-predict since it assumes an idealized 3-D mesh having a minimum number of neighbors and hence a smaller degree of communication than actually occurs. The TM is the most accurate taking into account much of these effects, but requires a separate trace to be analyzed for each mesh/processor-count pairing.

Table 3. Summary of Model Prediction Errors (%).

	GM			*MSM*			*TM*		
	i.	ii.	Av.	i.	ii.	Av.	i.	ii.	Av.
Nneut	9.1	15.7	12.4	5.7	28.4	17.0	6.1	5.0	5.5
Silc	2.9	11.1	7.0	1.8	17.2	9.5	0.8	7.4	4.1
Reac	5.3	13.5	9.4	8.4	22.3	15.3	2.3	11.5	6.9
Con_test5	2.2	8.9	5.4	4.2	13.1	8.7	1.3	4.6	3.0
Overall			8.6			12.6			4.9

Given its general nature and its reasonable accuracy, GM is used in the following section to provide insight into the performance of Tycho on systems and configurations that can not be currently analyzed through measurements.

6. Performance Exploration

The performance models developed in Section 5 can be used in many different ways to explore the performance space of Tycho, on current and future system architectures. Here we use the validated GM model to: i) detail where time is spent on the current ES40 system (Figure 7). ii) analyze the impact on performance when using systems containing other processors while still using the Quadrics QsNet network (Figure 8). iii) predict the performance on larger meshes and system sizes (Figure 9). iv) optimize the runtime by calculating the value of MCPS that results in a communication cost constituting less than 20% of the total (Figure 10).

The performance characteristics of the processors used in these scenarios are listed in Table 4. These computational characteristics were based on measurement made on single processors, in a similar way to that depicted in Figure 5. Note that the Itanium with a 3MB L3 cache, and the EV68 1.25GHz with a 16MB L2 cache, have a different memory hierarchy than that of the validation system, so that the curve with the number of cells per PE E_p, is also different. All the parallel systems considered use the Quadrics network, whose

performance characteristics are unchanged from those listed in Table 1, with the processors listed in Table 4.

Table 4. Element processing time (Telem(Ep) in µs) on different processors.

Memory	Alpha EV68 1GHz	Alpha EV68 1.25GHz	Itanium 800MHz
Main	7.0μs ($E_p \geq 16K$)	6.6μs ($E_p \geq 32K$)	22.2μs ($E_p \geq 8K$)
Main/Cache	$0.98 * Ln(E_p) - 3.4\mu$s ($800 < E_p < 16K$)	$0.76 * Ln(E_p) - 2.4\mu$s ($1600 < E_p < 32K$)	$3.7 * Ln(E_p) - 9.3\mu$s ($400 < E_p < 8K$)
Cache	3.0μs ($E_p \leq 800$)	2.4μs ($E_p \leq 1600$)	11.4μs ($E_p \leq 400$)

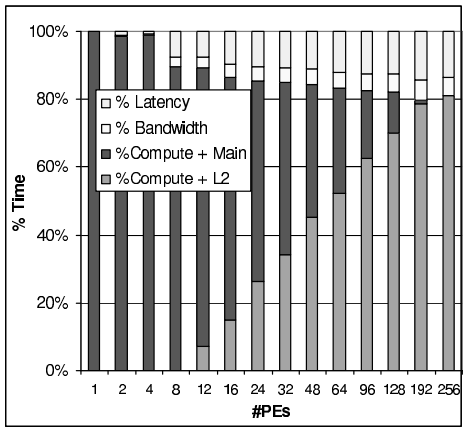

Figure 7. Time Component predictions improvement (Reac mesh on HP ES40).

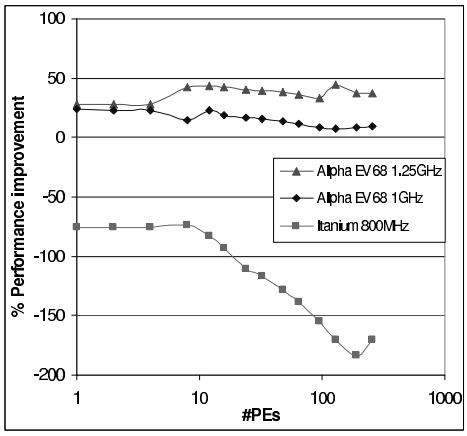

Figure 8. Predicted performance using Alpha EV68 1Ghz and Itanium 800MHz.

The time component predictions (Figure 7) are illustrated for the Reac mesh with $MCPS$ set at 64. A lower value of $MCPS$ results in more communications. However, it can be seen in Figure 7 that the application remains compute bound, with latency dominating the bandwidth component of the communication. In addition, due to the strong scaling behavior, the subgrid sizes get smaller at higher processor counts leading to better L2 cache behavior. For higher values of $MCPS$, less communication occurs and hence the overall time comprises a smaller communication component.

The predicted performance improvement over an Alpha EV68 833MHz system when using a system with either Alpha EV68 1GHz or an Itanium 800MHz reveals some interesting features (Figure 8). The 1GHz Alpha outperforms the

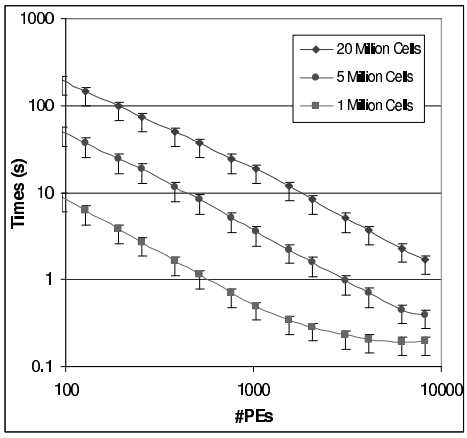

Figure 9. Predicted performance on larger meshes and processor counts.

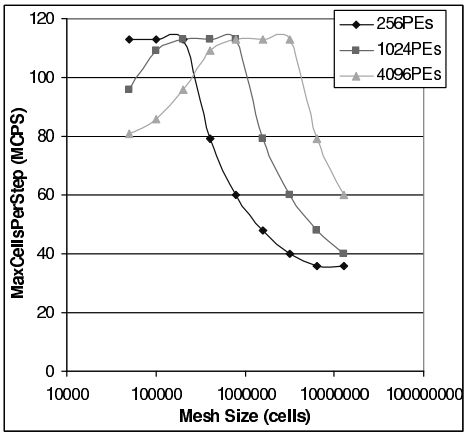

Figure 10. Predicted value of $MCPS$ (communication cost < 20% of total).

833Mhz processor as expected by about 20% for larger subgrid sizes (not fitting in the cache), but decreases to about 10% on larger PE counts due to similar L2 cache performance for the two processors. Similarly the 1.25GHz Alpha also outperforms the 833MHz EV68 due in part to its increased L2 cache (16MB vs. 8MB). The Itanium, at similar clock speeds, is performing significantly poorer than the Alpha. Its cache performance is also much poorer than that of the Alpha's, as evident by the lower performance on larger processor counts.

The predicted performance for larger meshes on larger processor counts indicates an expected good scaling behavior (Figure 9). This study considers meshes of size: 1,000,000 cells, 5,000,000 cells, and 20,000,000 cells being processed on up to 8192 PEs. Each curve assumes a PCE of 0.8 with the min and max bars indicating a PCE of 0.6 and 0.9 respectively. It can be seen that the smaller mesh has a significant communication component at large PE counts, as shown by the knee in the curve. If a value of $MCPS$ greater than 512 was used, the number of steps would decrease and hence the amount of communication would decrease. This indicates that for larger meshes, a corresponding larger value of $MCPS$ should be utilized.

The usage of the GM model to determine a value of $MCPS$ so that the communication component of the runtime is at most 20% is shown in Figure 10. A lower value of $MCPS$ is beneficial as it will result in a higher PCE value but increased communication cost. It can be seen that the value of $MCPS$ is dependent upon the mesh size as well as the processor count. It can be seen that the value of $MCPS$ actually exhibits an increase before decreasing as the mesh increases with size for a particular processor count. As the number of processors increases, the curve can be seen to shift to the right. The form of

these curves results from two competing factor. The first factor is the increased computation time as the mesh size increases due to less cache reuse - this leads to an increase in communication time resulting in a smaller $MCPS$ value. The second factor is the pipeline length as the mesh gets smaller - on a small mesh the pipeline length can dominate the PCE. Thus a greater degree of blocking (a lower value of $MCPS$) is required to keep processor utilization high.

From these analyses it can be seen that given the complexity of the performance issues associated with Tycho, this performance space cannot be analyzed without a general model. Issues such as determining the value of $MCPS$ are often too complicated and result from an interplay of many factors. It should be noted that it is planned to incorporate a version of the GM into the application code in order to dynamically determine the value of $MCPS$ at runtime and thus help optimize the time- to-solution for Tycho.

7. Summary

In this work we have presented predictive performance and scalability models for a deterministic transport application using unstructured meshes. The models take into account the main computation and communication characteristics of the entire code. Two of the models developed are analytic whereas a third is based on the analysis of runtime traces. The models are shown to have good accuracy through validation on a 64 node HP AlphaServer system.

The models varied in their degree of generality. The validation showed that as the models incorporated more specific details on the actual mesh being processed, the accuracy increased. However this also results in loss of generality, limiting the amount of insight into achievable performance.

It was shown that a generally applicable analytical model of this application can encapsulate the performance characteristics with reasonably accuracy and then used to explore the expected performance in many different performance scenarios. This is a key element of developing application based performance models - that is to explore the performance space - to aid in the development of the application code and to predict performance on future system architecture prior to their availability for measurement.

We believe performance modeling is key to building performance engineered applications and architectures. This work is one of few performance models that exist for entire applications. It follows on from our work on structured particle transport modeling [3], adaptive mesh refinement modeling [6], and Monte-Carlo simulation [9].

Acknowledgments

Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the US Department of Energy. The work was funded by LDRD at the Los Alamos National Laboratory.

References

[1] W.D. Gropp, D.K. Kaushik, D.E. Keyes and B.F.Smith. Performance Modeling and Tuning of an Unstructures Mesh CFD Application. In *Proc. SC2000*, Dallas, 2000.

[2] S. Girona, J. Labarta and R.M. Badia. Validation of Dimemas communication model for MPI collective operations. In *Proc. EuroPVM/MPI'2000*, Balatonfured, Hungary, September 2000.

[3] A. Hoisie, O. Lubeck and H. Wasserman. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *Int. J. of High Performance Computing Applications*, **14** pages 330-346, 2000.

[4] A. Hoisie, O. Lubeck, H.J. Wasserman, F. Petrini, H.J. Alme. A General Predictive Performance Model for Wavefront Algorithms on Clusters of SMPs. In *proc. ICPP 2000*, August 20-25, 2000. Toronto, Canada.

[5] G. Karypis and V. Kumar. *METIS 4.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System.* Technical Report, Department of Computer Science, University of Minnesota, 1998.

[6] D.J. Kerbyson, H.J. Alme, A. Hoisie, F. Petrini, H.J. Wasserman and M.L. Gittings. Predictive Performance and Scalability Modeling of a Large-scale Application. In *Proc SC2001*, Denver, November 2001.

[7] D.J. Kerbyson, E. Papaefstathiou, J.S. Harper, S.C. Perry and G.R. Nudd. Is Predictive Tracing Too late for HPC Users? In *High Performance Computing* (R.J. Allan, A. Simpson, and D.A. Nicole, Eds), Plenum Press, pages 57-67, March 1999.

[8] K.R. Koch, R.S. Baker and R.E. Alcouffe. *A Parallel Algorithm for 3D Sn Transport Sweeps.* LA-CP-92-406, Los Alamos National Laboratory, 1992.

[9] M. Mathis and D.J. Kerbyson. Performance Modeling of MCNP on Large-Scale Systems". In *Proc Los Alamos Computer Science Institute Symposium*, Santa Fe, October 2002.

[10] G.R. Nudd, D.J. Kerbyson et.al. PACE: A Toolset for the Performance Prediction of Parallel and Distributed Systems. *Int. J. of High Performance Computing Applications*, bf 14 pages 228-251, 2000.

[11] E. Papaefstathiou and D.J. Kerbyson. Predicting Communication Delays of Detailed Application Workloads. In *Proc of 13th ISCA Int. Conf. on Parallel and Distributed Computing Systems (PDCS)*, Las Vegas, August 2000.

[12] S.D. Pautz. An Algorithm for Parallel Sn Sweeps on Unstructures Meshes. *J. Nuclear Science and Engineering*, Vol. 140, pages 111-136, 2002.

[13] F. Petrini, W.C. Feng, A. Hoisie, S. Coll and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, **22** (1) pages 46-57, 2002

[14] S. Plimpton, B. Hendrickson, S. Burns and W. McLendon. Parallel Algorithms for Radiation Transport on Unstructured Grids. In *Proc. SC2000*, Dallas, November 2000.