A Performance Model of the Krak Hydrodynamics Application

Kevin J. Barker, Scott Pakin, and Darren J. Kerbyson
Performance and Architecture Laboratory (PAL)
Computer and Computational Sciences (CCS-3)
Los Alamos National Laboratory
{kjbarker, pakin, djk}@lanl.gov

Abstract

We present an analytic performance model of a large-scale hydrodynamics code developed at Los Alamos National Laboratory. This modeling work is part of an ongoing effort to develop models and modeling techniques for large-scale codes and systems of interest to Los Alamos and the national laboratory community [3]. Krak [1] comprises over 270,000 lines of source code and is capable of executing on a large number of parallel processors. Developing an accurate model is complicated by the irregular partitioning of input spatial grid cells to processors and the various material properties assigned to each cell. Model development proceeds by separating inter-processor communication from computation and modeling each individually. In addition, several approximations concerning subgrid size, shape, and material composition are made which reduce modeling complexity without adversely impacting prediction accuracy. We validate our model on several spatial grid sizes and processor configurations and demonstrate an accuracy at the largest scale on 512 processors to within a 3% error.

1. Introduction

Expectation of future workload performance is often a primary criterion in the procurement of a new large-scale parallel machine. While reliance on developers performance estimations sufficed in the past, the development of modeling techniques has made possible quantitative performance analysis for both current and near-to-market parallel systems. At Los Alamos National Laboratory (LANL) analytic performance modeling has proven useful in the study of future systems as well as in the optimization of systems after installation [5]. In addition, models can be useful for quantitatively evaluating the potential performance benefit of alterations to the application, such as the data-partitioning algorithms.

An analytic performance model can be considered a static representation of a workload's dynamic characteristics. Such characteristics include single-processor computation performance, inter-processor communication patterns, and the sizes of inter-processor messages. Performance models relate these characteristics to features of the input deck (e.g., spatial grid size and data partitioning method), and provide a performance estimation given a mapping of tasks onto the physical machine resources. Therefore, a performance model is dependent on both application and system characteristics.

Performance models do not aim to capture the performance of every line or subroutine in the source application; instead the model captures the broad aspects of the workload that have the primary impact on overall runtime, such as single-processor computation and inter-processor communication performance. While accurately modeling single-processor performance from first principles is an active and important area of research, our goal is to accurately quantify application performance on large-scale systems. Therefore, a scalability analysis is the focus of the model developed here.

The application we describe is a hydrodynamics code developed at LANL called Krak which consists of over 270,000 lines of source code in both Fortran and C, spread over more than 1,600 source files. This is a large-scale, production code capable of scaling to hundreds of processors by utilizing MPI for inter-processor communication. Our contribution lies not only in the development of the analytic performance model of a significant application utilizing an irregular mesh partitioning [4], but in the demonstration that careful approximations can be used to accurately model the performance of complex applications at large scale.

We begin our discussion in Section 2 with an overview of the application, including a description of the modeled input decks. We describe how data is partitioned among processors and the ramifications of this partitioning strategy on the modeling effort. We then discuss the development of the performance model, and proceed by separating the communication component from computation, and modeling each individually. Section 3 describes the performance model for computation, while Section 4 focuses on communication. We show that, although the application contains many variables such as the exact data partitioning among processors and the properties of the materials contained in the input deck, a generalized abstraction suffices to accurately model runtime, thus simplifying the model. Section 5 describes our work to validate the model, and we provide data to indicate our model closely predicts

actual application performance. Finally we draw conclusions in Section 6.

2. Krak Overview

Krak simulates forces propagating through objects composed of various materials and is often used to simulate high-energy explosives and object penetrations. Objects are mapped onto a spatial grid of "cells", with each cell being defined by (typically four) "faces", which are in turn composed of connections between "nodes". "Ghost nodes" are those nodes whose associated faces comprise boundaries between processors. Each cell in the spatial grid is assigned exactly one material. Krak is a Lagrangian code, meaning that the spatial grid deforms as forces propagate through the objects. "Slip lines" allow segments of the spatial grid to slide past one another, allowing the computation to remain numerically stable. Computation progresses until a specified number of time-steps or a given length of simulation time has passed.

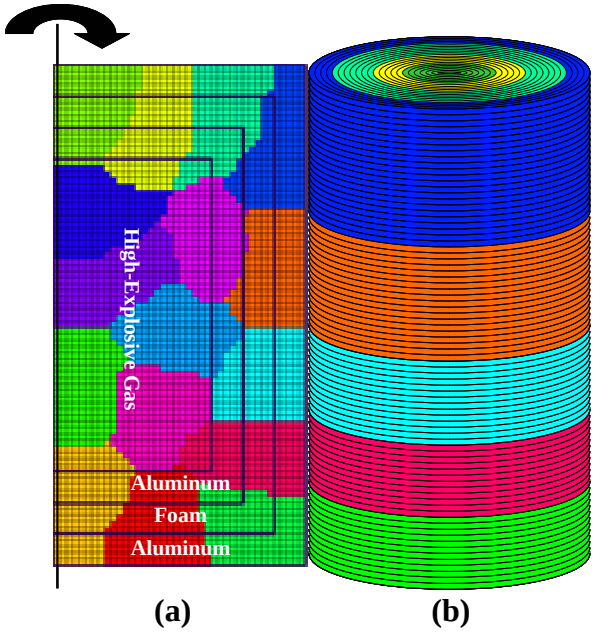

Figure 1: Example partitioning of 3200 cells on 16 processors; colors represent processor subgrids and thick lines represent material boundaries (a); the 3D cylindrical domain after rotation (b)

The input grid is partitioned into subgrids, with one subgrid assigned to each processor. Krak executes in *strong-scaling* mode, meaning the size of each subgrid in terms of cells decreases as the number of processors increases. Partitioning is performed using Metis [2] with an algorithm to balance cell counts on each processor while minimizing edge cuts. The partitioning is done in an irregular fashion (Figure 1), meaning a varying number of cells of each material is assigned to each processor. In addition, for a given input grid and processor count, the communication pattern between processors is not predictable without knowing precise partitioning information.

2.1. Input Description

We study three spatial grid sizes, which we term *small* (3200 cells), *medium*, (204,800 cells), and *large* (819,200 cells). Each spatial grid contains the same proportion of four different materials: a core of high explosive gas, a layer of aluminum, a layer of foam, and a second layer of aluminum (Figure 1). The rectangular, 2D spatial grid is rotated about a vertical axis so that the domain becomes a cylinder, with the high explosive gas forming an inner cylinder at the center. An explosive detonator is placed on the axis of rotation, slightly below center.

2.2. Iteration Composition

An iteration is composed of a series of "phases" that are separated by one or more global reduction operations which act as synchronization points. Such operations are used to combine data from every process, often for operations such as convergence testing or global error estimation. Table 1 contains a brief description of each phase. All phases contain some amount of computation, while several phases contain inter-processor communication as well. In order to develop a performance model it is necessary to separate communication from computation and model each separately.

Table 1: Summary of Krak activities by phase

Phase Number	Action	Sync Points
1	Broadcast (4 bytes, 8 bytes)	2
2	Bcast (4 bytes, 8 bytes) Boundary exchange Gather (32 bytes)	1
3	Computation only	3
4	Ghost node updates (8 bytes)	1
5	Ghost node updates (16 bytes)	1
6	Computation only	3
7	Ghost node updates (16 bytes)	1
8	Computation only	1
9	Computation only	1
10	Computation only	1
11	Computation only	2
12	Computation only	1
13	Computation only	1
14	Computation only	1
15	Broadcast (4 bytes, 8 bytes)	2

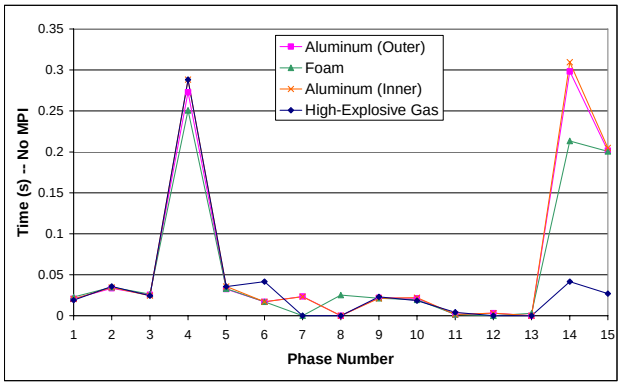

Figure 2: Computation time by phase on 256 processors with a spatial grid of 65,536 cells

Figure 2 depicts the time spent in each phase with an input spatial grid of 65,536 cells. MPI communication time is not included. Because of the fairly large processor count, subdomains are homogeneous in terms of materials. It can be seen that the time required for certain phases (for instance, phase 14) is material dependent; computation varies across processors due to cells material composition. In other phases the computation time is a function of only the number of cells allocated to a processor and not the material type of those cells. Each phase is therefore modeled independently.

3. Modeling Krak Computation

The computation time for an iteration can be expressed as:

$$T_{comp}(Phases, PEs, Cells) = \sum_{i=1}^{|Phases|} T_{phase}(Phases_i, PEs, Cells) \quad (1)$$

In Equation (1), *Phases* is an array of phase numbers, *PEs* is the array of processor IDs, and *Cells* is a matrix containing the material ID for each cell assigned to each processor ($Cell_{ij}$ contains the material ID for local cell *j* found on processor *i*).

The computation time for a single phase is stated as:

$$T_{phase}(Phase, PEs, Cells) = \max_{0 \le j < |PEs|} \sum_{k=1}^{|Cells_j|} (T(Phase, Cells_{jk}, |Cells_j|)) \quad (2)$$

Here, *T()* returns the per-cell cost from a piecewise linear equation given the phase and material type.

Combining Equation (1) with Equation (2) gives the total computation time for a single iteration:

$$T_{comp}(Phases, PEs, Cells) = \sum_{i=1}^{|Phases|} \max_{0 \le j < |PEs|} \sum_{k=1}^{|Cells_j|} T(Phases_i, Cells_{jk}, |Cells_j|) \quad (3)$$

Note that because phases are separated by global synchronization events, the time for a given phase is the maximum time over all processors in that phase.

As described in Section 2, the distribution of materials within a spatial grid as well as the allocation of cells to processors has an impact on overall application performance. Figure 3 illustrates this, indicating the computation time required per cell varies with both cell count (per processor) and, for some phases, material type. We therefore begin modeling computation by first developing a "mesh-specific" model, which utilizes precise knowledge concerning material and cell distribution. While the mesh-specific model should yield validation information, it is impractical for use in large-scale scalability analysis and often relies on information that cannot be known in advance. We then proceed to describe a "general" model in which several abstractions and simplifying assumptions are made. The generalized model utilizes a simplified input which allows for more rapid model evaluation, making it more useful for studying application performance at large scale.

3.1. "Mesh-Specific" Model Development

A contrived spatial grid is used to determine how computation time scales with grid size, and thereby, through a linear regression, determine the computation cost per cell of each material. Two processes are required; in order for a detonation to occur, high-explosive gas must be present. However, the gas can be isolated to a single process while the material on the second process varies. In theory, this allows for the construction of a piecewise linear equation to describe the time required for a cell of each material in each phase, given the number of cells of each material in the local subgrid. By summing these times, the computation time required for each subgrid can be calculated.

In practice, this is not always the case. From Figure 3 (center), it can be seen that the cost per cell often appears constant for larger subgrid sizes. However, as subgrid size decreases, the cost per cell increases until the computation time per subgrid approaches a constant regardless of the number of cells. Capturing the behavior at the knee of this curve has proven difficult, as the validation results presented in Section 5 will attest.

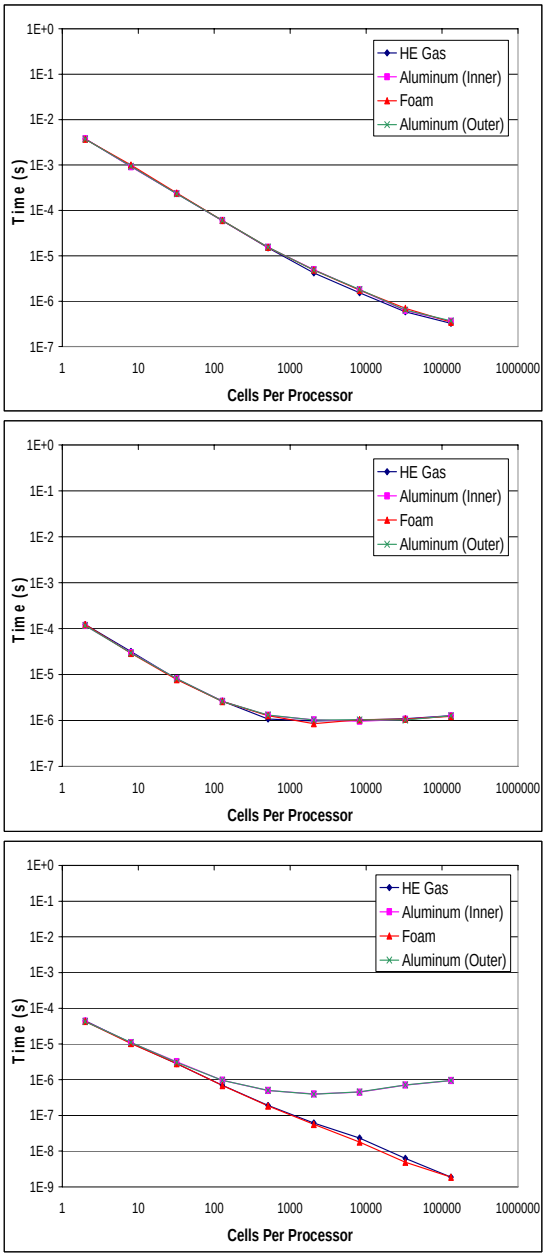

Figure 3: Per-cell computation times for Phase 1 (top), Phase 2 (center), and Phase 7 (bottom)

An alternative method for modeling computation utilizes the actual input domain (not a contrived grid as previously described) and involves the construction and solution of a series of linear equations with four variables (the computation time per cell of each material). A linear equation is constructed for each phase on each processor; the solution of this system of equations will provide a per-cell computation cost (in terms of time) that can then be fed into the model. Again, it is possible to construct a piecewise linear equation to allow for interpolation between measured values. This second method is used for the validation results presented below. Total computation time is calculated using Equation (3).

.

3.2. "General" Model Development

The "general" model differs from the mesh-specific model in that, instead of examining each subgrid generated by the Metis [2] mesh partitioning algorithm, the input domain is classified as either *heterogeneous* or *homogeneous*, and the ratio of materials in each subgrid remains fixed as the number of processors increases. As before, Equation (3) is used to calculate the total computation time. A feature of strong-scaling execution is that as the number of processors scales, the size of each subgrid decreases, making it more likely that a subgrid will contain only cells of a single material. Table 2 defines the ratio of materials in these two cases for the global spatial grid. The ratio of materials used in heterogeneous mode is derived from the input deck shown in Figure 1 and would therefore vary from one input to another. For homogeneous mode, it is assumed that for each material a subgrid exists that is composed exclusively of cells of that material. By calculating which material results in the longest computation time, the time required for each phase of computation can be determined.

Table 2: Ratio of materials in Krak general model

Type	H.E. Gas	Aluminum (Inner)	Foam	Aluminum (Outer)
Hetero.	39.1%	17.2%	20.3%	23.4%
Homo.	100%	100%	100%	100%

In reality, because Krak operates in strong-scaling mode, as the number of processors increases the size of each processor's subdomain becomes smaller. The material composition of each processor's subdomain transitions from being more heterogeneous (with the ratio of materials matching the ratio of materials in the global spatial grid when only a single processor is used) to more homogeneous. The Krak model does not yet have a way to model this transition; however, at large processor counts, the homogeneous case seems to adequately model true application behavior.

Each processor's subdomain is assumed to contain an equal number of cells. In addition, each subdomain is assumed to be square, so that each boundary between processors contains $\sqrt{|Cells|/|PEs|}$ faces. Boundary faces are divided equally among the materials in use. The number of ghost nodes on each boundary is one more than the number of boundary faces, and half of the ghost nodes on each boundary are "local" (i.e., owned by the local

processor) with the remaining half "remote" (owned by the neighboring processor).

4. Modeling Krak Parallel Communication

Communication during a single program iteration falls into three categories:

1. Boundary exchanges, in which the size of each message is dependent upon the number of shared message faces
2. Ghost node updates, in which the size of each message is dependent upon the number of ghost nodes that are "owned" by each member of a processor pair
3. Collective communications, in which the number and size of messages are independent of the spatial grid size and number of processors.

All point-to-point communication operations utilize asynchronous send operations coupled with blocking receives. Messages to multiple neighbors are therefore overlapped: asynchronous sends to each neighbor are posted, followed by operations to ensure the send operations have completed, and finally, blocking receives are posted to receive messages from neighboring nodes.

Basic point-to-point message passing time is modeled using the following piecewise linear equation:

$$T_{msg}(S) = L(S) + S \cdot T_B(S) \tag{4}$$

The term $L(S)$ describes the start-up cost for a message of S bytes, while $T_B(S)$ is the bandwidth cost to send a single byte between nodes.

4.1. Boundary Exchange

Because the spatial domain is partitioned among processors in an irregular fashion, each processor has N neighbors, where N varies across processors. Boundary exchanges consist of a step for each material, plus one additional phase. Each phase consists of six messages per neighboring process. For each pair of processors, message sizes are a function of both the number of shared faces of a given material and of the number of ghost nodes on those faces which are associated with more than one material.

The first two messages in each set of six are of size 12 bytes times the number of faces of the corresponding material plus 12 bytes for each ghost node touching more than one material. The remaining four messages contain only 12 bytes times the number of faces of the corresponding material. The six messages in the final, additional phase are of size 12 bytes times the total number of faces (independent of material) on the process boundary. Identical materials (such as the two aluminum materials in our input deck) are treated as one during boundary exchanges.

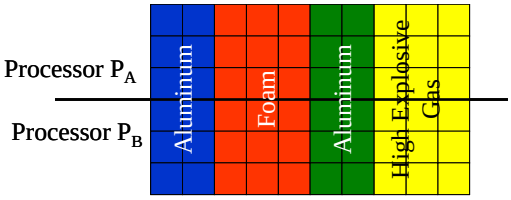

Figure 4: Processor boundary with four materials

As an example, consider Figure 4, in which the boundary between two processes comprises 3 faces of high explosive gas, 2 faces of aluminum, 3 faces of foam, and 2 more faces of aluminum. The message sizes and tallies are shown in Table 3.

The time required to complete a boundary exchange operation for one processor with a single neighbor can be described as:

$$T_{BE}(Faces) = 6T_{msg}\left(12 \cdot \sum_{i=0}^{|Materials|} Faces[i]\right) + \sum_{i=0}^{|Materials|} \begin{cases} 6T_{msg}(12 \cdot Faces[i]) & Faces[i] > 0 \\ 0 & otherwise \end{cases} \tag{5}$$

where *Faces[]* is an array containing the number of boundary faces for each material type, T_{msg} is the time required to send a message of a given size (defined in Section 4.4), and $|Materials|$ is the number of materials in the input spatial grid. T_{msg} is defined by Equation (4) and describes the time required to send a message of S bytes between processors. Note that Equation (5) does not account for overlapping of messages between different neighbors, for combining like materials, or for increasing the size of messages due to ghost nodes lying on material boundaries.

Table 3: Boundary exchange example

Material	Msg. Count	Size of Each Msg (bytes)
H.E. Gas	2	48 = 3·12 + 1·12
	4	36 = 3·12
Aluminum (both)	2	84 = 2·12 + 2·12 + 3·12
	4	48 = 2·12 + 2·12
Foam	2	60 = 3·12 + 2·12
	4	36 = 3·12
All	6	120 = (2+3+2+3)·12

4.2. Ghost Node Updates

Every ghost node is considered to be "local" to (i.e., owned by) exactly one processor. All other processors which share this node consider it to be "remote". Ghost node updates take place in phases 4, 5, and 7 from Table 1. Each phase requires two messages between each neighboring processor; one for local ghost node updates and one for remote. In phase 4, 8 bytes are transferred for each ghost node (Equation 6), and 16 bytes are transferred for each ghost node in each of phases 5 and 7 (Equation 7).

$$T_{GNPhase4}(N_L, N_R) = T_{msg}(8N_L) + T_{msg}(8N_R) \quad (6)$$

$$T_{GNPhase5,7}(N_L, N_R) = T_{msg}(16N_L) + T_{msg}(16N_R) \quad (7)$$

Again, note that Equations (6) and (7) do not consider the overlapping of messages between different neighbors.

4.3. Collective Communication

There are three types of collective communication operations which take place during each iteration. The number of operations and size of each message are fixed and do not vary with problem size or number of processors. These operations are described in Table 4.

Table 4: Collective communication operations per iteration

Type	Count	Size (bytes)
MPI_Bcast()	3	4
	3	8
MPI_Allreduce()	9	4
	13	8
MPI_Gather()	1	32

Collective communication is modeled as either "fan-out", "fan-in", or "fan-in and fan-out" pattern with messages reaching every node over a binary-tree structure. Therefore, a one-to-all communication requires log(*P*) messages, and a synchronization point requires 2log(*P)* messages. The time required for the collective operations described in Section 4.3 is modeled as:

$$T_{Broadcast}(N_{PE}) = 3\log(N_{PE})T_{msg}(4) + 3\log(N_{PE})T_{msg}(8) \quad (8)$$

$$T_{All-reduce}(N_{PE}) = 18\log(N_{PE})T_{msg}(4) + 26\log(N_{PE})T_{msg}(8) \quad (9)$$

$$T_{Gather}(N_{PE}) = \log(N_{PE})T_{msg}(32) \quad (10)$$

5. Model Validation

From Sections 3 and 4, an "input-specific" and "general" performance model encompassing computation and communication can be constructed. In both cases, computation time is given by Equation (3). Communication is given by the summation of Equations (5 – 10). It is assumed that computation does not overlap with communication; the overall runtime is the summation of the computation and communication components.

5.1. "Input-specific" Model Validation

Validation results are given in Table 5 for two spatial grid sizes on three different processor configurations. Validation runs were performed on a 256-node HP/Compaq machine consisting of ES-45 nodes (4 Alpha EV-68 processors running at 1.25 GHz), each with 16 GB of memory. Nodes are connected using a Quadrics QsNet-I fat-tree network [6].

Table 5: Validation results for mesh-specific model

Problem Size	PE Count	Meas. (ms)	Pred. (ms)	Error
Small	16	27	43	-59.0%
	64	88	41	52.7%
	128	28	30	-10.0%
Medium	16	310	290	5.9%
	64	88	89	-0.8%
	128	61	59	4.5%

In two cases, the predicted runtime was in error by more than 50%. This is the case near the knee of the per-cell cost curve described in Section 3 and shown in Figure 3, phase 2. This large error indicates that the linear regression itself, or the linear interpolation between measured values in the cost curves are not accurate. However, for the local cell counts that would be seen when running a large problem on a large number of processors, the model is accurate to within 10%. In addition, we will see in Section 5.2 that large errors are not observed when using the general model.

5.2. "General" Model Validation

Table 6 contains validation results using the "general" Krak model with a homogeneous material distribution. We include only this case in the table as it more accurately describes material distribution at large scale; however validation results for both homogeneous and heterogeneous material distributions are shown in Figure 5. As expected, the general model with a homogeneous material distribution validates well,

particularly at larger processor configurations. At large scale a heterogeneous material distribution is less accurate; due to the fixed global problem size, as the processor count increases subgrids become smaller and therefore more homogeneous. This fact leads to an over-prediction of runtime in the heterogeneous case caused by a misprediction in the communication cost. Separate messages are required for each material that touches a processor's subgrid boundary. Because of the small size of these messages at large scale, the latency suffered by each message becomes significant. In the homogeneous case, only a single material touches any subgrid boundary, meaning fewer boundary-exchange messages are required.

Table 6: Krak validation results for general model

Problem Size	PE Count	Meas. (ms)	Pred. (ms)	Error
Medium	128	61	66	-8.0%
	256	49	51	-4.0%
	512	44	43	2.9%
Large	128	170	177	-4.3%
	256	95	100	-4.6%
	512	67	67	-1.0%

6. Conclusions

We have described the development of an analytic model of a complex hydrodynamics code. This is part of an ongoing effort to develop models and techniques for modeling large-scale codes of interest to Los Alamos and the national laboratory community. The performance of this particular code is challenging to model because of the irregular method of domain decomposition and variations in workload on a per-cell basis caused by differences in material properties.

We began by developing an "input-specific" model which uses detailed information describing the spatial grid's partitioning and material composition. While this model can be used for validation purposes, it is unsuitable for scalability analysis. A piecewise linear equation derived from a regression analysis is used to predict per-cell computation costs for various materials and sub-sizes. However, it can be seen that this model suffers from inaccuracies for particular subdomain sizes.

The observation that subdomains tend to become more homogeneous in terms of material composition as the number of processors increases led to the development of a second "general" model that can be used for rapid and accurate modeling at large scale. Because each processor contains cells of only a single material, only the most computationally taxing material needs to be modeled, significantly simplifying the model's structure and implementation. We have observed that this is an accurate representation of reality at large processor counts. We have validated the general model using two spatial grids, and have demonstrated that on 512 processors, model accuracy is within 3% (Table 6).

Acknowledgements

We are grateful to Scott Runnels and Hank Alme of Los Alamos for their assistance in providing the version of Krak and input decks used here. This work was funded in part by the Accelerated Strategic Computing (ASC) program for the Department of Energy. Los Alamos National Laboratory is operated by the University of California for the U.S. Department of Energy under contract number W-7405-ENG-36.

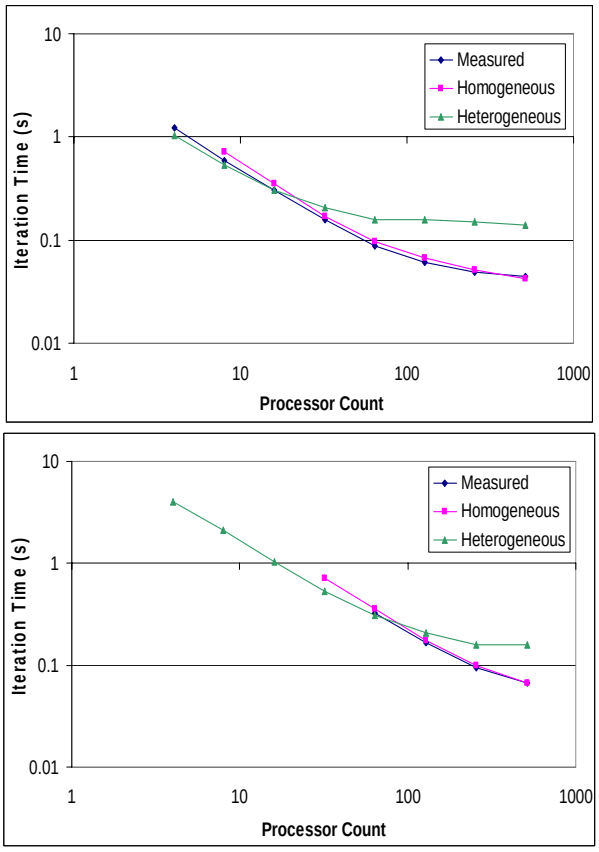

Figure 5: General model validation for medium (top) and large (bottom) problems

References

[1] D. Burton, Multidimensional Discretization of Conservation Laws of Unstructured Polyhedral Grids. 2nd International Workshop on Analytical Methods and Process Optimization in Fluid and Gas Mechanics, 1994.

[2] G. Karypis, V. Kumar. METIS 4.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System. Tech. Report, Dept. Computer Science, University of Minnesota, 1998.

[3] D.J. Kerbyson, H.J. Alme, A. Hoisie, F. Petrini, H.J. Wasserman, M.L. Gittings. Predictive Performance and Scalability Modeling of a Large-scale Application. In IEEE/ACM Supercomputing (SC'01), Nov. 2001.

[4] M.M. Mathis and D.J. Kerbyson. A General Performance Modeling of Structured and Unstructured Mesh Particle Transport Computations. Journal of Supercomputing, 34:181-199, 2005.

[5] F. Petrini, D.J. Kerbyson, S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. IEEE/ACM Supercomputing (SC'03), 2003.

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