SANDIA REPORT

SAND2008-4073
Unlimited Release
Printed July 2008

The Benefits of Adaptive Partitioning for Parallel AMR Applications

Johan Steensland
Sandia National Laboratories, CA

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

SAND2008-4073
Unlimited Release
Printed July 2008

The Benefits of Adaptive Partitioning for Parallel AMR Applications

Johan Steensland
Advanced Software Research and Development,
Sandia National Laboratories, P.O. Box 969,
Livermore, CA 94550-9915, USA,
jsteens@somnet.sandia.gov,
Phone, fax: (925)-294-2474, (925)-294-2234

Abstract

Parallel adaptive mesh refinement methods potentially lead to realistic modeling of complex three-dimensional physical phenomena. However, the dynamics inherent in these methods present significant challenges in data partitioning and load balancing. Significant human resources, including time, effort, experience, and knowledge, are required for determining the optimal partitioning technique for each new simulation. In reality, scientists resort to using the on-board partitioner of the computational framework, or to using the partitioning industry standard, ParMetis.

Adaptive partitioning refers to repeatedly selecting, configuring and invoking the optimal partitioning technique at run-time, based on the current state of the computer and application. In theory, adaptive partitioning automatically delivers superior performance and eliminates the need for repeatedly spending valuable human resources for determining the optimal static partitioning technique. In practice, however, enabling frameworks are non-existent due to the inherent significant inter-disciplinary research challenges.

This paper presents a study of a simple implementation of adaptive partitioning and discusses implied potential benefits from the perspective of common groups of users within computational science. The study is based on a large set of data derived from experiments including six real-life, multi-time-step adaptive applications from various scientific domains, five complementing and fundamentally different partitioning techniques, a large set of parameters corresponding to a wide spectrum of computing environments, and a flexible cost function that considers the relative impact of multiple partitioning metrics and diverse partitioning objectives.

The results show that even a simple implementation of adaptive partitioning can automatically generate results statistically equivalent to the best static partitioning. Thus, it is possible

to effectively eliminate the problem of determining the best partitioning technique for new simulations. Moreover, the results show that adaptive partitioning can provide a performance gain of about 10 percent on average as compared to routinely using the industry-standard, ParMetis.

Acknowledgment

This work was supported Sandia National Laboratories' ASC program, under the Focused Research Innovation and Collaboration program element, administered by Dr. David Womble. Thanks to Jaideep Ray, Sandia National Laboratories, CA, for suggestions leading to a significantly improved paper. Thanks to John Peterson at CFDLab, University of Texas, for scientific collaboration and providing the Convection application, Richard Drake at Sandia, New Mexico, Computational Shock and Multiphysics Department, ALEGRA/NEVADA code project, for providing the Mach-Stem application, to Charles Norton at Jet Propulsion Lab, NASA, for providing the Quake application, and to James Overfelt at Sandia, New Mexico, for providing the LaserRaster application. Thanks to Karen Devine, also at Sandia, New Mexico, for providing Zoltan with all drivers and source code, and for supporting the project throughout. Thanks to Ralf Deiterding at Oakridge National Laboratory, Tennessee, for providing the Spheres and ShockTurb applications. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

Contents

List of Figures

List of Tables

1 Introduction

Adaptive mesh refinement (AMR) methods are being widely used for numerical simulations of physical phenomena in many scientific domains, including earthquake (tectonic/thermal) modeling [20], computational shock multi-physics [16], computational heat transfer [3], materials science applications such as crack propagation [8], biological problems such as E-Coli proliferation [19], and generalized classes of nonlinear reaction-diffusion systems [7]. The survey articles by Carey et. al [5, 6] detail several additional diverse application areas including Laplace-Young fluid surface interface modeling, electro-rheological flows, drift-diffusion semiconductor modeling, high-speed compressible flow, variable-density transport in porous media (see also [21]), and glacier flow modeling [11]. The governing partial differential equations are solved by numerical methods on a discrete mesh. When dynamic and localized features in the solution require higher mesh resolution for obtaining sufficient numerical accuracy, the mesh adapts dynamically to accommodate this. Consequently, these simulations are inherently dynamic.

Parallel implementation of AMR potentially leads to realistic modeling of complex three-dimensional physical phenomena. However, it also presents significant challenges in dynamic resource allocation. The parallel efficiency is limited by the effectiveness of the partitioner to partition and distribute the underlying mesh at runtime to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load. Because application adaptation during execution results in repeated modifications of the mesh, it is necessary to repeatedly repartition the mesh to maintain efficient use of resources.

The specific requirements for an effective partitioning depend on properties of the mesh. As the mesh adapts to the solution, these requirements change. Consequently, no partitioning technique performs the best for all types of simulations [25]. The best partitioner for a particular time frame of a parallel simulation might be the worst for the next time-frame. By explicitly considering these dynamic conditions, the scalability for large, realistic simulations could possibly be significantly improved. Significant human resources, including time, effort, experience, and knowledge, are required for determining the optimal partitioning technique for each new simulation. In reality, scientists resort to using the on-board partitioner of the computational framework, or to using the partitioning industry standard, ParMetis.

To meet the challenges in dynamic resource allocation inherent in parallel AMR, we introduce another level of adaptation: *adaptive partitioning*, meaning dynamic and automatic switching of partitioning techniques, based on the current run-time state. In theory, adaptive partitioning automatically delivers superior performance and eliminates the need for repeatedly spending valuable human resources for determining the optimal static partitioning technique. In practice, however, enabling frameworks are non-existent due to the inherent significant inter-disciplinary research challenges.

This paper presents a study of a simple implementation of adaptive partitioning and discusses

implied potential benefits from the perspective of common groups of users within computational science. The study is based on a large set of data derived from experiments including six real-life, multi-time-step adaptive applications from various scientific domains, five complementing and fundamentally different partitioning techniques, a large set of parameters corresponding to a wide spectrum of computing environments, and a flexible cost function that considers the relative impact of multiple partitioning metrics and diverse partitioning objectives.

Related work includes the AdaptiveRepart algorithm [22] in the ParMetis library [18], the VPI scheme [31] in the Jostle package [30], and the new hypergraph partitioner [9] in the Zoltan Library [15]. A common characteristic for these three bodies of work, is that they focus on developing state-of-the-art algorithms with the ability to efficiently optimize a number of metrics given some user-defined parameters. In contrast to the algorithm focus, the present paper focuses on the understanding of the benefits of being able to automatically switch partitioning algorithm repeatedly at run-time, based on the current state of the application and computer system.

2 The Parallel Mesh Application Simulator

Achieving an objective comparison of a set of load balancers for an arbitrary set of adaptive applications can be prohibitively problematic. First, gathering all applications, compiling and running them on the same platform, might be impossible for practical, technical, and political reasons. Second, deriving run-time metrics like load imbalance, data migration, and communication can be challenging, as it often requires sophisticated profiling tools that force re-instrumentation of the application source code. Third, applications often have tailored interfaces to certain load balancers, making it a daunting task to implement an interface to another load balancer.

We have developed "the parallel mesh application simulator" [27] that enables easy comparison of load balancers for arbitrary AMR applications. The simulator is depicted in Figure 1 and is a uniform and accessible test-bed, which addresses each of the problems discussed above. Rather than working with application source code and makefiles, we represent each application with an *application trace*. An application trace is a sequence of serial (un-partitioned) mesh files, corresponding to the adaptive mesh at regular intervals in the simulation. Typically, these trace files are created by the code developer in the application's native execution environment. We convert the trace files from their native format to the Exodus format [23]. The simulator reads these Exodus mesh files and — according to input parameters like the number of processors and the load balancing configuration — partitions the mesh via an interface to the Zoltan library [15]. Much in the spirit of earlier work by Parashar and students [24], our simulator executes in serial and partitions the mesh into local (virtual) partitions. Useful metrics, such as data migration, edge cut, load imbalance, and partitioning time, are derived uniformly for all time-steps.

Our parallel mesh application simulator provides limitless load balancing plug-and-play during run-time. The number of desired partitions can be specified and all of Zoltan's algorithms can be freely configured on a time-step basis.

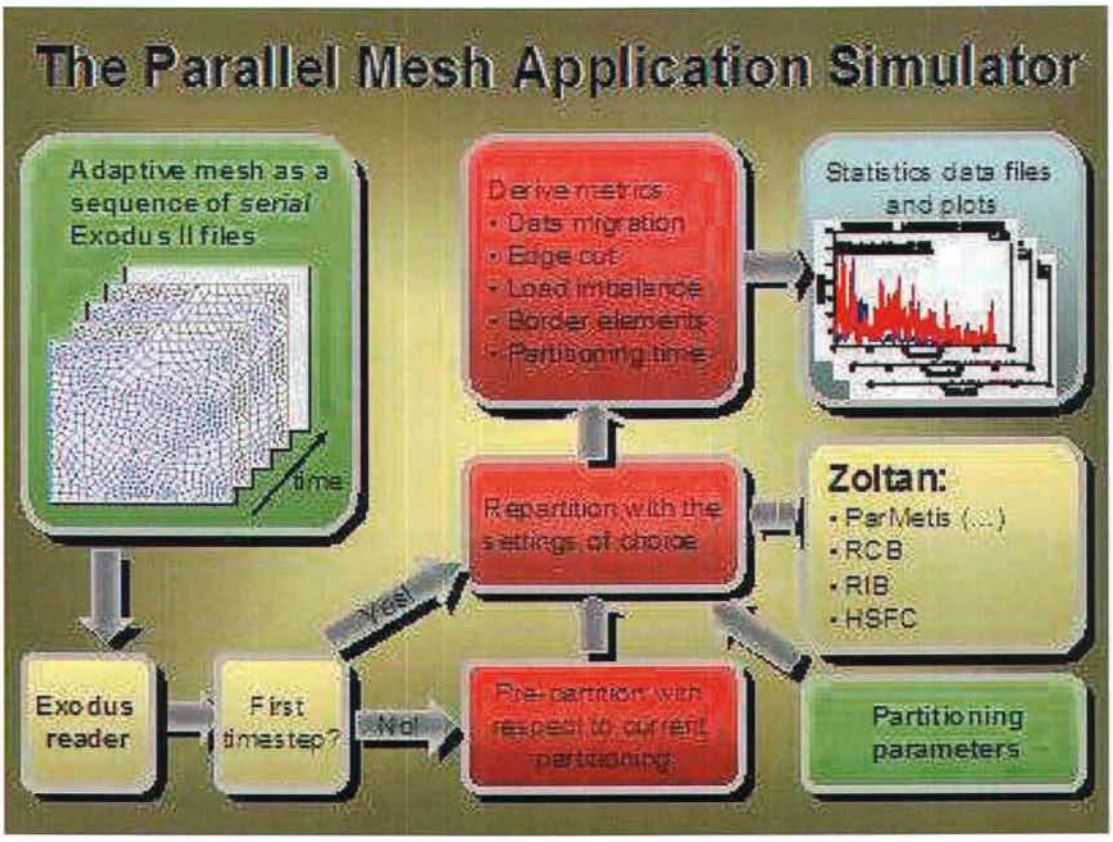

Figure 1. The parallel mesh application simulator with its constituent parts. It consists of an application trace reader (left) and an analytical subsystem (middle) that evaluates the quality of the partitions obtained from Zoltan's (right) partitioners. It also outputs quality metrics for plotting and further (manual) analysis (right).

3 Cost Function

To capture the success of a given partitioner for a given mesh, we use a cost function [27]. Shloegel et al [22] use Cost = ITR × edgecut + migration, where ITR is a user defined weight that determines the relative importance of optimizing edge cut. This cost function is designed specifically for the task of selecting the most cost effective algorithm in the adaptive partitioner AdaptiveRepart in ParMetis. However, apart from edge cut (or communication) and data migration, there is a third significant contributor to parallel inefficiency, and therefore cost: load imbalance. Thus, we expand on the above cost function to also include load imbalance, as follows:

$$\Pi = \mathrm{CCR} \times \mathrm{loadimbalance} + \mathrm{ITR} \times \mathrm{edgecut} + \mathrm{migration}, \tag{1}$$

where CCR is a weight that determines the relative importance of optimizing load imbalance. In a physical computer, CCR could be thought of as the "compute-to-communicate ratio" (hence the acronym). Because migration is given in number of elements, and edge cut could be regarded as

the number of elements that need to communicate, their unit (elements) are basically the same. To obtain the same unit for the load imbalance, we define it as (*max load* (in elements)) - (*average load* (in elements))[1]. Thus, all three metrics have the same unit (elements), and Cost is expressed as a sum of elements, weighted by the relative costs.

This simple cost function provides a flexible way to gauge the effectiveness of a partitioner to simultaneously optimize and trade-off three of the most significant contributors to parallel inefficiency. By adjusting the parameters CCR and ITR, we obtain a wide spectrum of hypothetical computing conditions.

In a real-world scenario, a small CCR would translate to a communication-dominated application, and expensive interprocessor communication compared to in-processor computations in the parallel computer. Thus, the partitioner has to focus less on load imbalance and more on network traffic, such as communication and data migration. A small ITR would translate to an application with frequent repartitionings and expensive data movement. Thus, the partitioner has to execute fast and focus more on minimizing data migration.

4 Real-World Applications

We use six real-world adaptive mesh applications from a variety of scientific domains. The applications have different sizes, geometries, structures, and refinement patterns. Because of this difference in properties, each application poses a unique challenge to the partitioning algorithm. Moreover, different stages in the same application might be fundamentally different. Consequently, different stages might require different partitioning configurations to generate high-quality partitionings. The applications are listed in Table 1.

Described below, ShockTurb and Spheres are structured adaptive mesh applications from the Virtual Test Facility [28, 12] implemented in AMROC [13, 14] at the California Institute of Technology. The structured two-dimensional grid hierarchies are transformed into three-dimensional finite element meshes as follows. Each grid patch is transformed from a rectangular mesh with quadratical elements to a cuboid of cubic elements, simply by giving each square a height equal to its sides. A refined grid patch is positioned on top of its parent and expanded into three dimensions analogously. The three-dimensional mesh thus expands and contracts as the two-dimensional grid is refined and un-refined. Figure 4 illustrates the idea. Note that even though these two applications are native *structured* applications, they pose significant and realistic challenges to the partitioners.

Quake Space technologies, e.g. InSAR (Interferometric Synthetic Aperture Radar) imaging, now allow the measurement of previously unobservable earth phenomena. InSAR allows scientists to discover and measure small changes (order of centimeters) in surface elevation that occur during and after earthquakes, and other seismic/tectonic events, and may some day aid in earthquake prediction. GeoFEST [1] is a NASA-developed finite element software package for modeling solid stress and strain in geophysical and other continuum domain applications. Coupled with the

[1]This definition of load imbalance is appropriate for our applications with uniform element workload.

parallel adaptive mesh refinement capabilities of the Pyramid [2, 20] library, GeoFEST uses InSAR data as boundary and initial conditions to compute stress, strain, and displacement fields beneath the earth's surface. The parallel scalability and AMR data structures of Pyramid enable such simulations to run faster and use fewer system resources than the traditional, fixed-grid approach allowed (see Figure 2, above).

MachStem In the MachStem application (see Figure 2, below), a planar, high-Mach shock wave impacts a wedge of material at some angle of incidence. The shock reflects off the wedge, and for some wedge angles, produces a "Mach stem" (or a double Mach stem) feature: a localized region of extremely high gradients and complex flow structure. The availability of experimental pressure data for the Mach stem problem makes it a desirable code-verification application [10]. The Mach stem problem is implemented in ALEGRA [4]: a finite element, multi-material, arbitrary Lagrangian/Eulerian hydrocode, which simulates shock hydrodynamics and solid dynamics. ALEGRA can operate on a fully unstructured mesh or on a multi-block structured mesh. The physics algorithms are built on top of the NEVADA framework, which provide mesh and field data storage, load balancing, input parsing, data output utilities, communication utilities, and access to third party libraries.

Laser-Raster The laser-raster problem (see Figure 3, above) is based on Sandia work on exposure of a silicon wafer to an advanced lithography tool using laser rastering. The mesh is three-dimensional but thin. At first it is only a single element layer thick, but the adaptation produces refinement normal to the face so it becomes a true three-dimensional mesh. Nowadays, the application is used as a regression test of the software Calore [3].

Convection in Porous Media Double-diffusive natural convection in a homogeneous porous medium [21] (see Figure 3, below) occurs whenever there are two diffusing components (e.g. heat and species concentration), which have competing effects on the density of a background fluid (e.g. water). This temperature- and species concentration-dependent density is based on the Boussinesq approximation, and couples the continuity, momentum (Darcy's flow law), energy, and species concentration equations. A variety of scales exists in the pressure, temperature, and solute concentration fields. In particular, sharp boundary layers are often observed in the solute field, and h-adaptive finite element techniques are used to investigate these effects.

ShockTurb — Planar Richtmyer-Meshkow instability ShockTurb (see Figure 4, above) treats the interaction of two contacting gases with different densities. When the gases are subject to a shock wave, the interface between them becomes unstable and the result is called a Richtmyer-Meshkov instability. The Richtmyer-Meshkov instability finds applications in stellar evolution and supernova collapse, pressure wave interaction with flame fronts, supersonic and hypersonic combustion and in inertial confinement fusion. In the simulation, an incident Mach 10 shock wave causes vortices along a sinusoidally perturbed gas interface (five symmetric perturbations). The geometry is rectangular and closed, except at the left-most end. The gases are air and SF_6 (sulfur and fluoride). The simulation is motivated by physical experiments [29].

Spheres — Cylinders in hypersonic flow In the Spheres application (see Figure 4, below), a

constant Mach 10 flow passes over two spheres placed inside the computational domain. The flow results in steady bow shocks over the cylinders. This is a realistic flow problem with complex boundaries. Both the workload and the number of patches increase sharply during the first 50 time steps. They then decrease until time step 125, where they stabilize and remain constant for the duration of the execution. The increase in patches occurs when the incident flow starts to hit the two spheres. Heavy turbulence is present until bow shocks begin to form behind the spheres. The turbulence is decreased when the bow shocks become stable, attributing to the decrease in both the workload and the number of patches. When the bow shocks are fully formed and stable, the workload and the number of patches are constant.

Application	Element	Dim	Timesteps	Avg #elmts	Max #elmts
Quake	Tetra	3D	6	$308K$	$1.6M$
MachStem	Quad	2D	109	$8.2K$	$12.5K$
LaserRaster	Cube	3D	65	$4,4K$	$10.3K$
Convection	Tetra	3D	73	$87K$	$94K$
SpheresSAMR	Cube	3D	70	$233K$	$341K$
ShockTurbSAMR	Cube	3D	100	$135K$	$231K$

Table 1. Table of applications

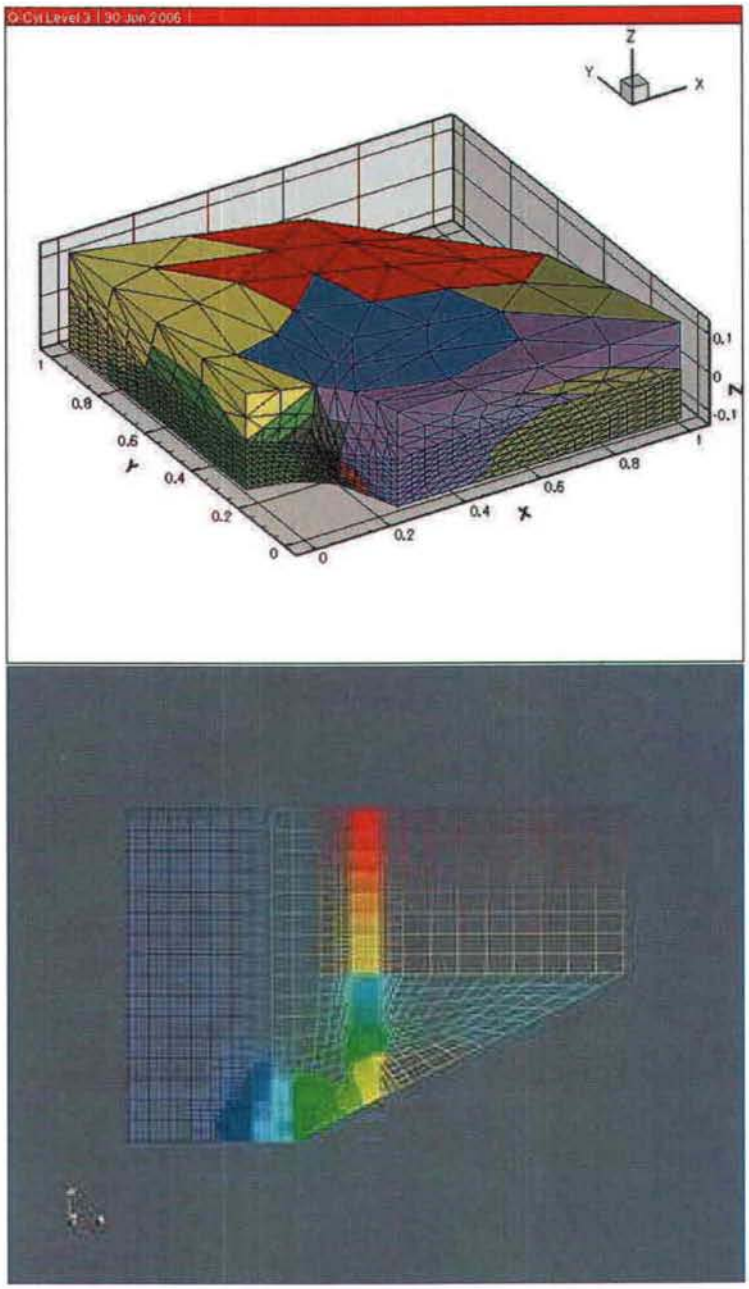

Figure 2. Mesh configurations taken from two applications, namely Quake (above) and Mach-Stem (below). Note that these figures correspond to *one* instance of the adaptive meshes that the simulations employ; these meshes adapt as the simulation progresses.

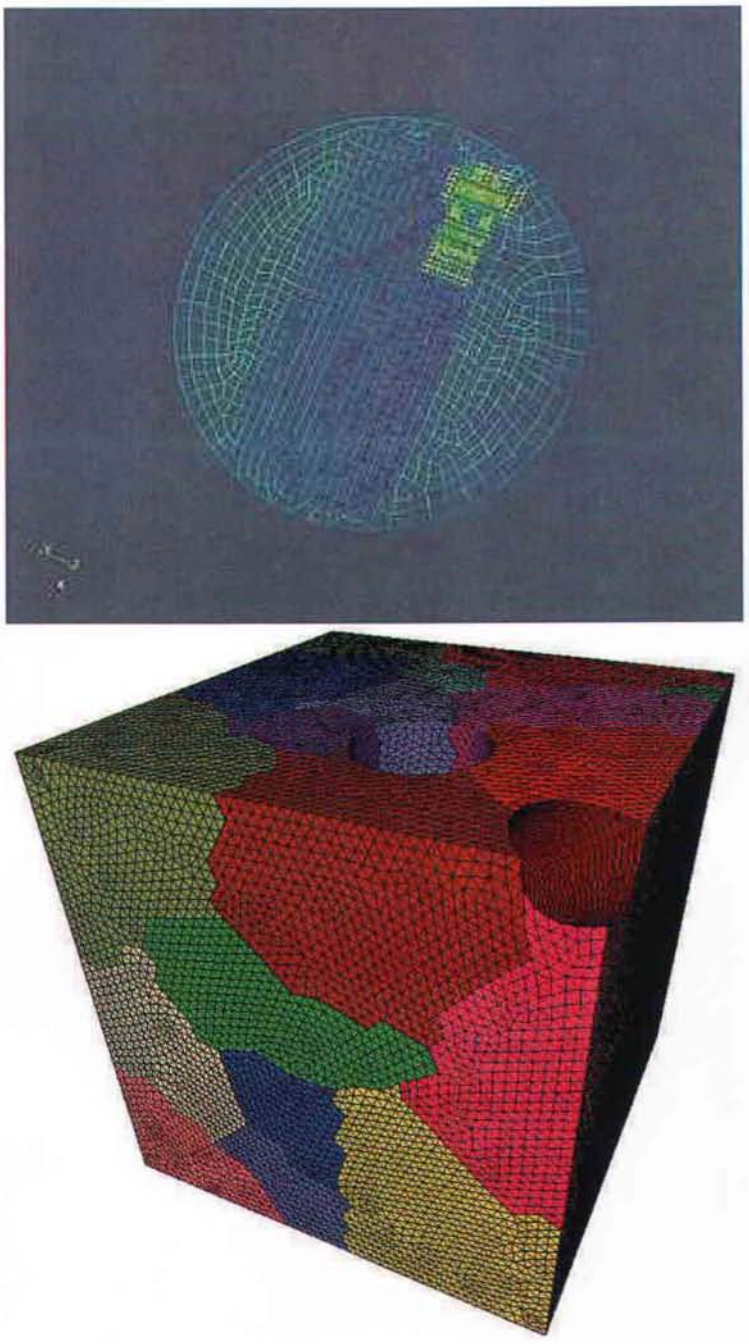

Figure 3. Mesh configurations taken from two applications, namely Laser-raster (above) and Convection (below). Note that these figures correspond to *one* instance of the adaptive meshes that the simulations employ; these meshes adapt as the simulation progresses.

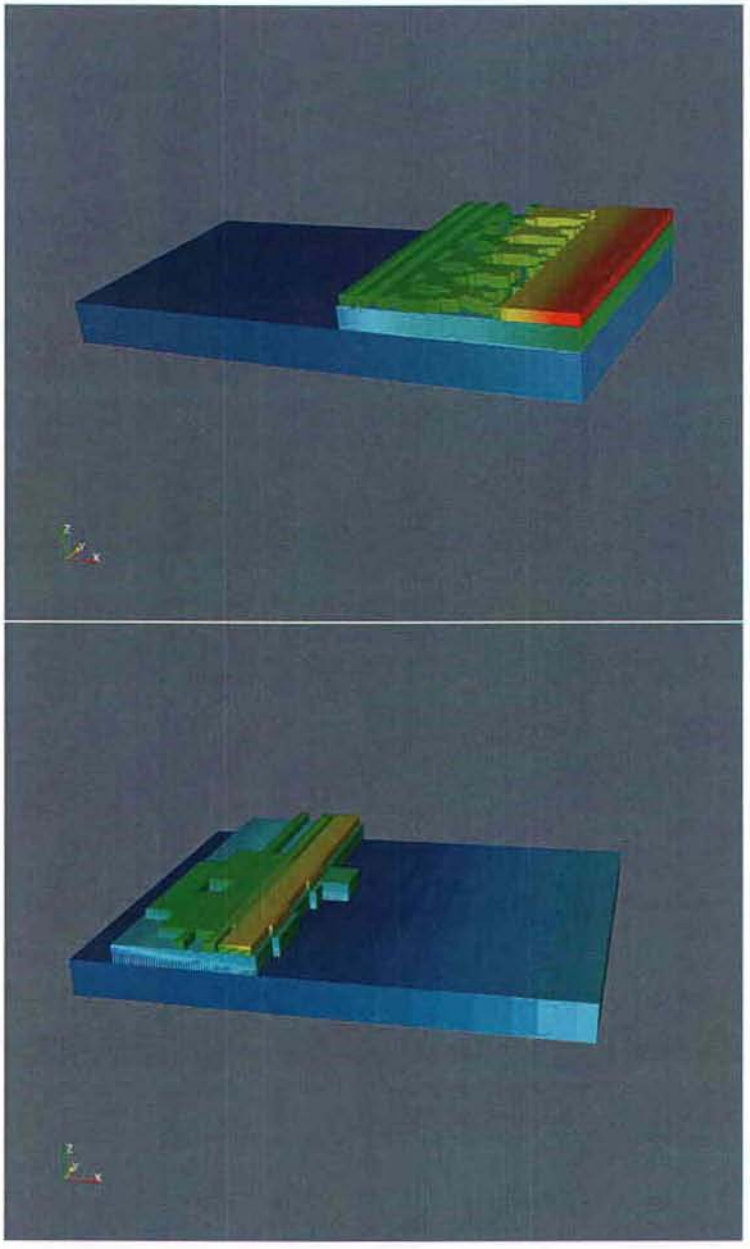

Figure 4. Mesh configurations taken from two applications, namely ShockTurb (above) and Spheres (below). Note that these figures correspond to *one* instance of the adaptive meshes that the simulations employ; these meshes adapt as the simulation progresses.

5 Partitioning Algorithms

In this study, we use five partitioning algorithms: recursive coordinate bisection (RCB) [15], RCB with a post re-mapping step (RCB+Remap), AdaptiveRepart [22], HyperGraph [9], and Hilbert space-filling curve (HSFC). While RCB, RCB+Remap, HyperGraph, and HSFC are implemented in Zoltan [15], AdaptiveRepart is a part of the ParMetis library [18] to which Zoltan has an interface.

The RCB and HSFC algorithms are *geometric* while AdaptiveRepart and HyperGraph are *graph based.* Geometrical algorithms operate on coordinates defined in some space, while graph-based methods operate on vertices and edges (that may or may not have physical coordinates). Geometric methods produce decompositions that implicitly try to control communication costs by maintaining geometric locality of the elements. Graph-based methods attempt to explicitly control communication costs by data dependencies. As a consequence, these two types of algorithms behave rather differently and seem to address different partitioning objectives.

RCB lacks a dedicated partitions-to-processor mapping mechanism. To keep data migration under control, it relies on the assumption that the coordinate bisection will preserve some important geometric properties of the mesh. Zoltan offers a remapping function, that, given an existing distribution and a newly computed partitioning, can remap the new partitions on to the processors to minimize the data migration between the existing and the new distribution.

AdaptiveRepart is (as its name indicates) adaptive in the sense that it tries to adapt to its input. Two fundamentally different partitioning approaches are tried in parallel: a scratch/remap approach and an incremental, diffusion, approach. A scratch/remap algorithm operates in two steps. First, the data is partitioned from scratch, disregarding the current distribution. Second, the resulting partitions are remaped with the objective to reduce data migration. Incremental algorithms operate by modifying the current distribution with the objective to create a new high-quality partitioning while keeping data migration under control. For AdaptiveRepart, the approach that, given a cost function (see Section 3), generates the more cost effective solution, is selected. This test is applied to a coarse version of the graph. The selected version is then successively un-coarsened and refined to produce the final result.

Hypergraph is a new algorithm developed for the Zoltan library. It is based on a model for dynamic load balancing, where the sum of the communication and migration cost is minimized. This model can be solved with a hypergraph with fixed vertices. The resulting algorithm is a parallel multilevel partitioning algorithm within the Zoltan load balancing toolkit.

The HSFC algorithm is an implementation of the widely used inverse space-filling curve partitioning method [26]. The method consists of three conceptual steps: indexing, sorting and coloring. In the indexing step, each element is numbered according to its Hilbert curve index. This creates a one-dimensional list of indexed elements. To preserve locality, the element list is then sorted on the Hilbert indices. Last, consecutive portions of the sorted list are colored (or assigned) to their respective partitions.

The partitioners unsed in this study, and their characteristics, are summerized in Table 2.

Algorithm	Class	Model	Package
RCB	Geometric	Scratch	Zoltan
RCB/Remap	Geometric + Graph	Scratch/Remap	Zoltan
AdaptiveRepart	Graph based	Scratch/Remap Incremental	ParMetis
HyperGraph	Graph based	Incremental	Zoltan
HSF	Geometric	Scratch	Zoltan

Table 2. Table of partitioning algorithms

6 Adaptive Partitioning

In the following, assume an adaptive scientific application A with m time-steps, and a set of partitioners $P_1, P_2, ..., P_n$. Assume that A has been executed n times — once for each partitioner, resulting in n sequences of partitionings, each m time-steps long. Additionally, assume that there are quality metrics, including load imbalance, edge cut, and data migration, computed for each time-step and partitioner. Last, assume that these metrics are weighed together to form a cost metric for partitioner P applied to application A at time-step t by the cost function $\Pi(P,A,t)$ described in Section 3.

A first, theoretical approach to establish adaptive partitioning could be to select, for each time-step, the algorithm associated with the lowest cost as estimated by Π. However, there is a major problem inherent in this approach.

Because partitioning techniques might differ fundamentally in their algorithmic steps, their end product, i.e., the processor-mapped partitions, might be fundamentally different. This is especially true for techniques from different classes. For example, a geometric partitioning algorithm considers the coordinates of each element, while a purely graph-based technique does not. Instead, the graph-based technique considers vertices and edges. Adding to the complexity, *incremental* techniques use the partitioning at time-step $t-1$ as a basis for computing the partitioning at time-step t, and *scratch-remap* techniques first partition the mesh from scratch, and then re-map the resulting partitions onto the processors with the objective to maximize data "overlap" with the existing distribution. As a result, data that partitioner P_1 clusters on processor p_1, might be distributed over *all* processors *but* p_1 by partitioner P_2. Even though the partitioning metrics indicate that two partitionings are similar quality-wise, their end products may still be fundamentally different.

Thus, switching the current partitioning technique for a theoretically superior technique might actually decrease parallel efficiency [27]. The data necessary to migrate to conform to the native data mapping of the new technique might be substantial. It follows that estimating P_i's effect on a A at time-step t, exclusively based on P_i's quality metrics for t, is insufficient. Estimated that way, the quality metrics in general, and the data migration component in particular, are suspect. This is because the distribution for P_i at time-step t are based on the distribution at time-step $t-1$, *also computed by* P_i. It follows that the cost $\Pi_t(P_i,A,t)$ is also dependent on P_i applied to A at time-step

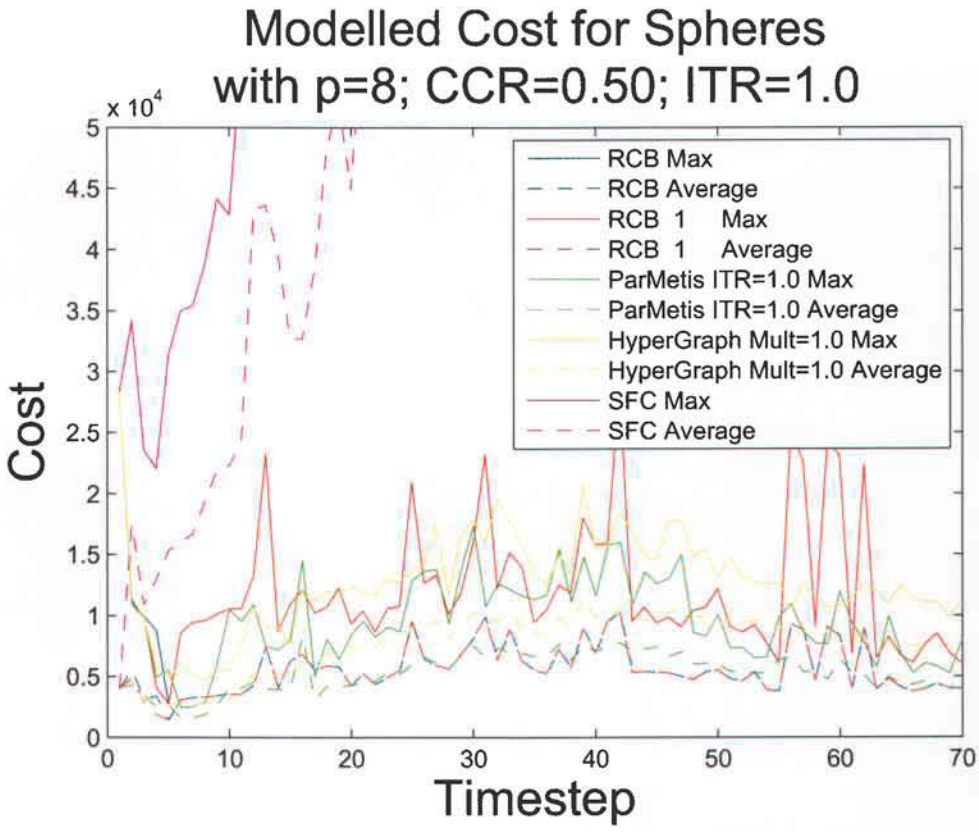

Figure 5. The modeled cost for a set of partitioners for two different data sets (max, avg) for Spheres with $p = 8$, CCR=0.5, and ITR=1.0 for Π applied to both max and average values. Max and average values (across processors) refer to those of load-imbalance, edge-cuts and data-migration, as they enter Eq. 1. Note that "Timestep" on the horizontal axis refers to the chronological index of the mesh in the application trace and may not refer to the timestep of the time-integrator used in the simulation.

$t-1$. However, for a random algorithm-switch, the two consecutive time-steps are associated with two different partitioning algorithms.

6.1 Uniform Starting Point

We suggest that the available partitioners be divided into the two groups *incremental* and *scratch/remap*. The first step is to force the incremental partitioners to a uniform starting point. For example, if we for A normally use either RCB (scratch/remap) or a diffusion technique (incremental) from ParMetis, then we would force ParMetis to let RCB partition the first time-step. When ParMetis is subsequently used, it will modify the original RCB partitioning, rather than modifying something completely different (perhaps computed by some other ParMetis algorithm). The resulting sequence of partitionings is more likely to be similar to that created by RCB.

The generalization to multiple incremental partitioners is trivial. However, it is an open question how to handle multiple scratch/remap partitioners.

6.2 Switching-Penalty

In the following, we assume multiple incremental partitioners, denoted $P_1^I.P_2^I,...,P_n^I$ and a single scratch/remap partitioner denoted P^S.

We propose a system of penalty factors to better estimate the data migration cost associated with switching from one partitioner to another. Switching from P^S to some P_i^I is considered "safe", as P_i^I will use P^S as a starting point for its incremental partitioning. Thus, the transition $P^S \rightarrow P_i^I$ does not result in a penalty. However, switching from some P_i^I to P^S, is considered "unsafe" as P^S will consider neither the current P_i^I nor P^S from the previous time-step. Hence, any similarity to P^S at the previous time-step is coincidental. Because the result is unpredictable, it results in a penalty f. In short, we define the penalty function, $F(f)$ as follows:

$$F(f) = \begin{cases} f \text{ for } P_i^I \rightarrow P^S \\ 1 \text{ otherwise.} \end{cases}$$

The data migration component in the cost function Π is then multiplied by $F(f)$.

6.3 The Simple Implementation

Apart from the uniform starting point and switching penalties described above, our simple, theoretical approach to adaptive partitioning is identical to the first approach outlined in Section 6. For each time-step, the algorithm associated with the lowest cost, considering the switching penalty, is selected. The resulting sequence of minimal cost algorithms is called the adaptive partitioning sequence.

Figures 5 and 6 depict an example of adaptive partitioning for the application Spheres with $p = 8$, CCR=0.5, and ITR=1.0 for Π applied to both max and average values. Figure 5 shows the costs (Π) associated with a set of partitioning techniques. Connected lines correspond to Π applied to max values, and dotted lines to Π applied to average values. Figure 6 shows the resulting dynamics of the adaptive partitioning, i.e., the adaptive partitioning sequence. The blue line shows the adaptive partitioning sequence resulting from applying Π to max values, and the red line shows the result using average values.

As described above, the quality metrics in general, and the data migration component in particular, of the computed adaptive partitioning sequence are suspect. Switching the current partitioning technique for a theoretically superior technique might actually decrease parallel efficiency. The data necessary to migrate to conform to the native data mapping of the new technique might be

substantial. Consequently, to obtain the accurate metrics for the adaptive partitioning, the application/computer configuration is executed again, this time with the adaptive partitioning sequence applied.

The difference in cost between the adaptive partitioning sequence and the adaptive partitioning with accurate metrics for Spheres with $p = 8$, CCR=0.5, and ITR=1.0 for Π (Eq. 1) applied to both max and average values is illustrated in Figures 7 and 8, the third and fourth clusters of histograms. Maximum and average values here refer to the maximum or average values, across processors, of load-imbalance, edge-cuts and data-migration, as they enter Eq. 1. Each cluster of bars depicts the results fora given combination of CCR and Π applied to max or average values. Each bar shows the total cost over all timesteps for a given partitioning algorithm. In Figure 7, the cost of the adaptive partitioning sequence (the rightmost bar in each cluster) is compared to the cost of the other (static) algorithms. In Figure 8, the cost of the adaptive partitioning with accurate metrics (the rightmost bar in each cluster) is added. It is clear that the HSFC algorithm is significantly worse than all the other partitioners, while RCB (both native and with remapping) lags AdaptiveRepart from ParMetis. The optimal sequence of partitioners is "theoretically" superior to ParMetis; however, when data-migration due to switching of partitioners is factored it, ParMetis is generally superior, except when CCR = 1.0. In this example, the adaptive partitioning with accurate metrics was more expensive than the adaptive partitioning sequence. As discussed above, the result might as well have been the opposite.

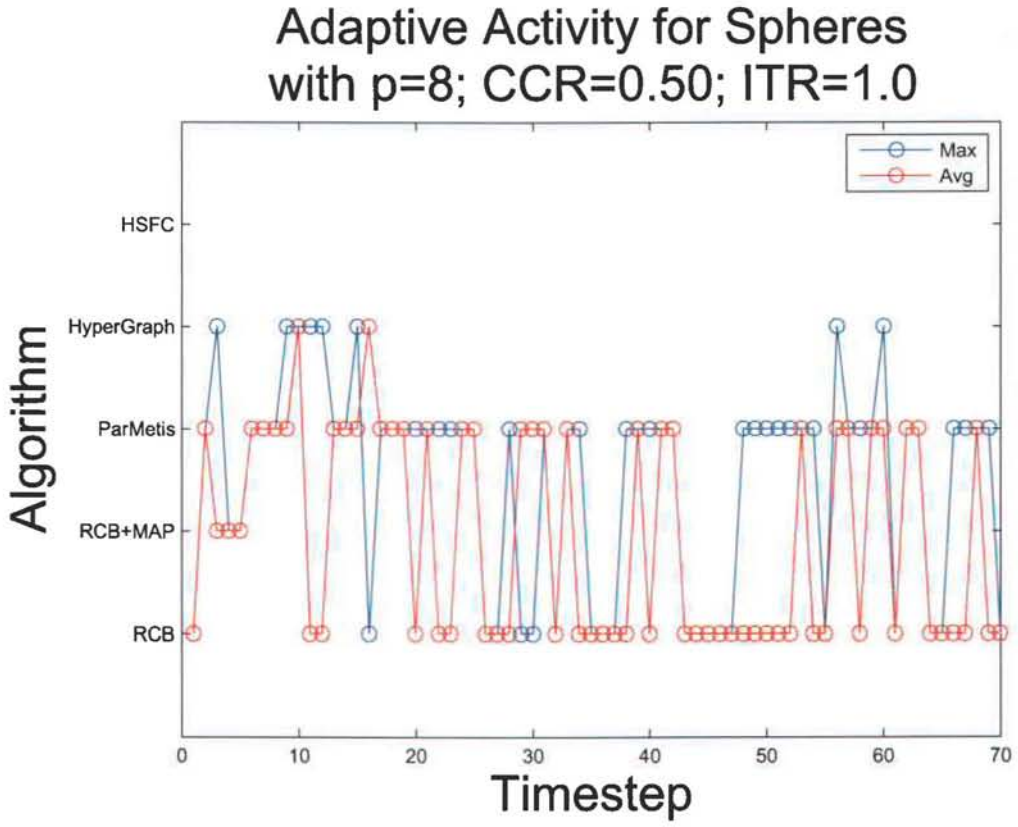

Figure 6. The resulting sequence of partitioners for Spheres with $p = 8$, CCR=0.5, and ITR=1.0 for Π applied to both max and average values. The blue line shows the adaptive partitioning sequence resulting from applying Π to max values, and the red line shows the result using average values. Max and average values (across processors) refer to those of load-imbalance, edge-cuts and data-migration, as they enter Eq. 1. Note that "Timestep" on the horizontal axis refers to the chronological index of the mesh in the application trace and may not refer to the timestep of the time-integrator used in the simulation.

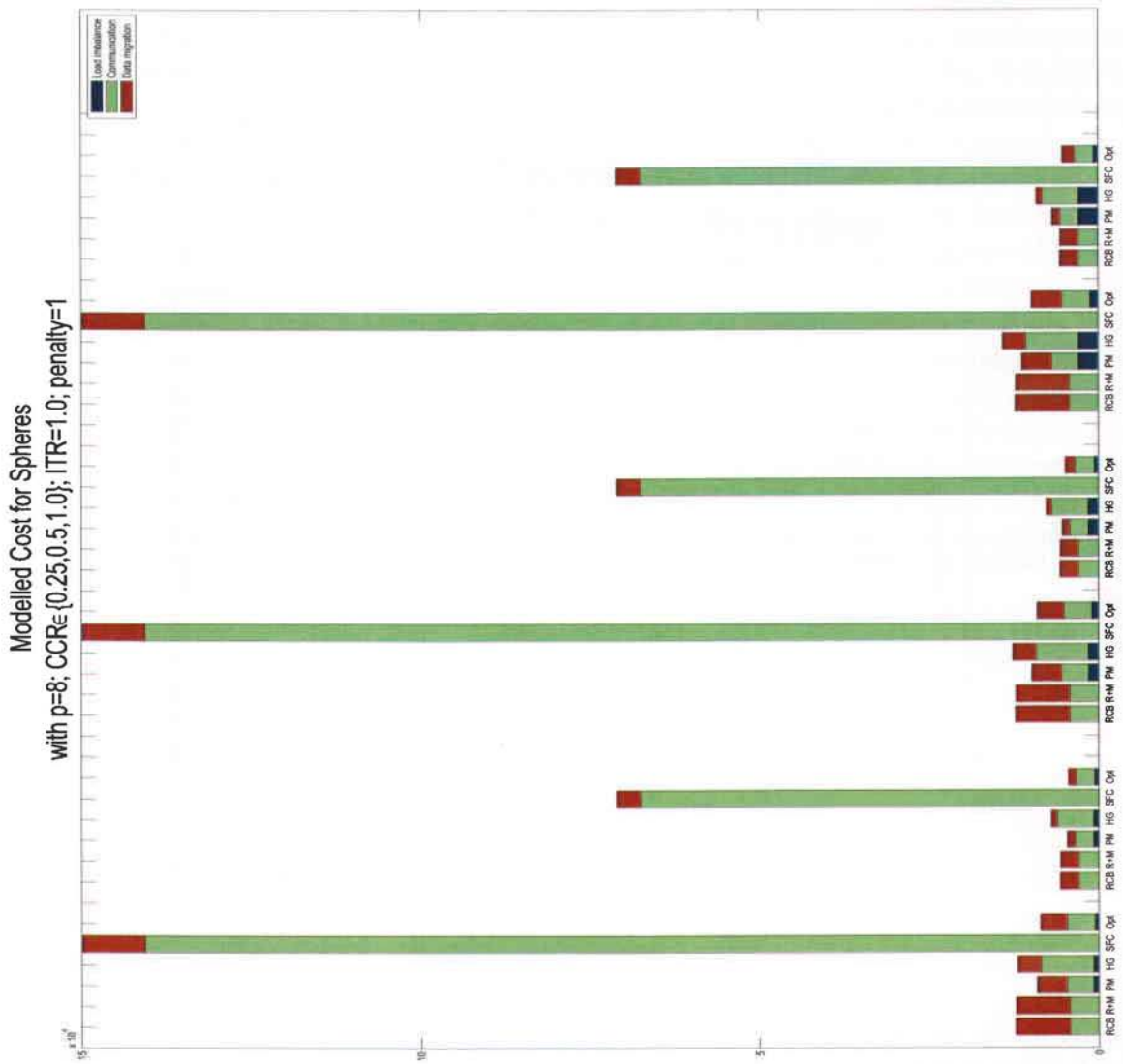

Figure 7. The (theoretical) cost of the adaptive partitioning sequence (rightmost bar in each cluster) compared to a set of static techniques for CCR = {0.25, 0.5, 1.0} (the left, middle and right clusters of histograms).Results from Π, as applied to average (across processors) values of load-imbalance, edge-cuts and data-migration, is plotted first, followed by Π as applied to the max values of the same variables. Results are plotted for the application "Spheres" with ITR=1.0. The costs associated with load-imbalance, communication and data-migration are plotted in blue, green and red, respectively. RCB = the recursive coordinate bisection algorithm, R+M = RCB with remapping, PM = ParMetis, HG = Hypergraph partitioner in Zoltan, SFC = Hilbert space-filling curve partitioner (Zoltan) and Opt=Optimal sequence of partitioners.

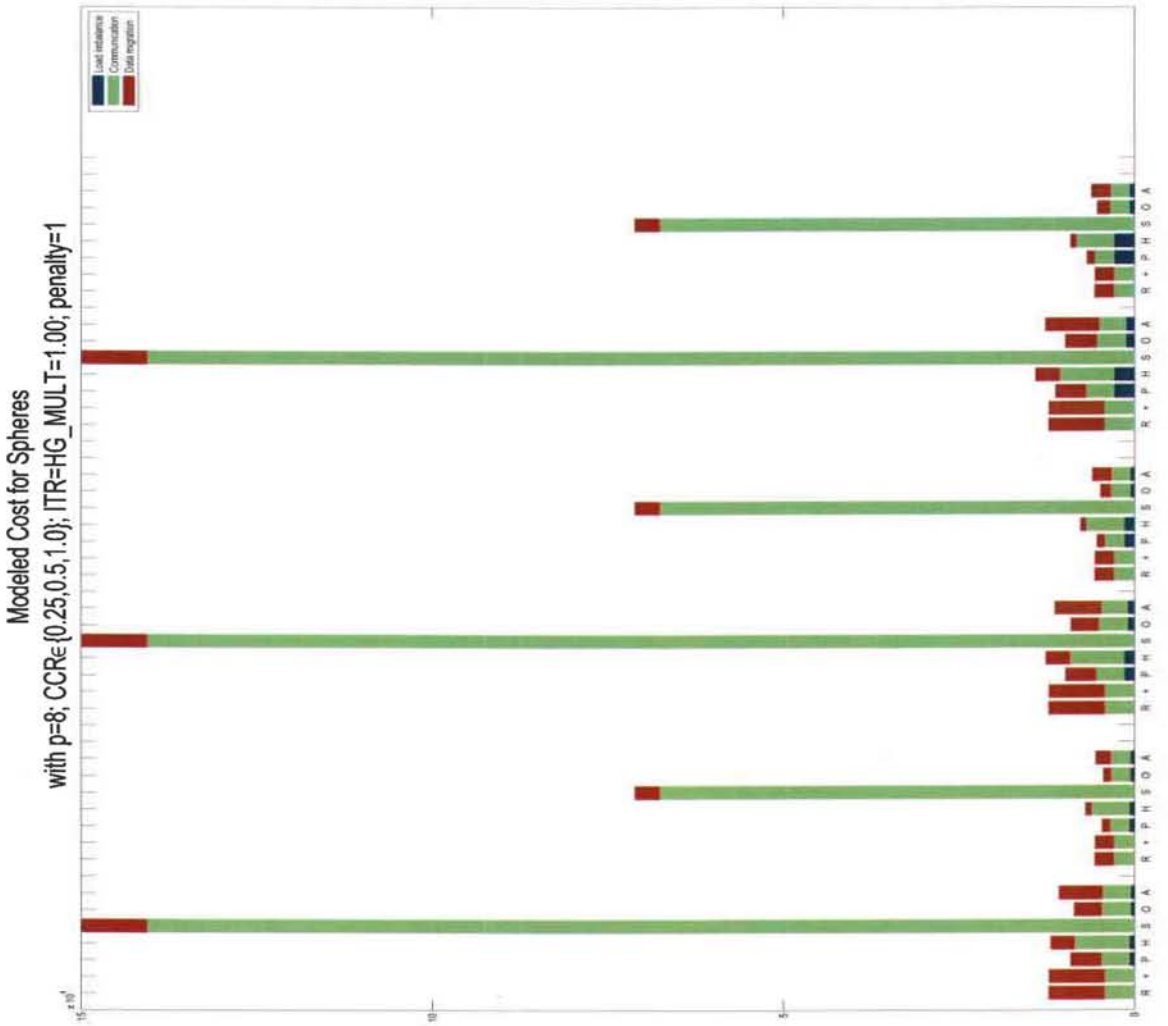

Figure 8. The accurate cost of the adaptive partitioning sequence (rightmost bar in each cluster) compared to a set of static techniques for CCR = {0.25, 0.5, 1.0} (the left, middle and right clusters of histograms). Results from Π, as applied to average (across processors) values of load-imbalance, edge-cuts and data-migration, is plotted first, followed by Π as applied to the max values of the same variables. Results are plotted for the application "Spheres" with ITR=1.0. The costs associated with load-imbalance, communication and data-migration are plotted in blue, green and red, respectively. Note the difference in data migration (red part of bars) as compared to the theoretical adaptive partitioning in Figure 7. R = the recursive coordinate bisection algorithm, + = RCB with remapping, P = ParMetis, H = Hypergraph partitioner in Zoltan, S = Hilbert space-filling curve partitioner (Zoltan), O=Optimal sequence of partitioners and A = optimal partitioner sequence with data migration costs (i.e. with accurate metrics).

7 Experimentation

7.1 Hypothesis and Setup

First, we state our *Hypothesis*: In theory, adaptive partitioning gives superior results. In practice, however, enabling frameworks are non-existent due to the inherent significant inter-disciplinary research challenges. In our experiments, we assume a dynamic and fictitious application-computer-network execution environment where the three main contributors to parallel inefficiency — load imbalance, communication and data migration — are all significant. Then, compared to a set of static partitioning configurations that generate comparable cost in this environment, a simple implementation of adaptive partitioning, based on a uniform starting point and switching penalties, can perform in parity with the best static algorithm and thus eliminate the significant problem of wasting vast human resources to determine the best (or suitable) partitioner for each new simulation.

To investigate whether our hypothesis held true, we used a penalty factor $f \in \{1,2,4,8\}$ applied to the cost function Π for the five fundamentally different partitioning techniques (see Section 5) for six multiple-timestep AMR applications from different scientific domains (see Section 4, Eq. 1). By using the simulator (see Section 2), we derived the following data points for each timestep: load imbalance, max and average edge cut, and max and average data migration.

Rather than examining metric by metric, we wished to investigate the *impact* a given partitioning technique had on the overall parallel efficiency. To accomplish this, we used Π (see Section 3) applied to two complementing sets of data. First, Π was applied to average data, i.e., the edge cut and data migration were the average values over all processors. Second, Π was applied to max data, i.e., the edge cut and data migration were the maximum values for all processors. The load imbalance was identical for both function applications. By applying the cost function to these two different sets of data, we captured a wider gamut of conditions. For example, assume a sophisticated, highly parallel network where *all* inter-processor communication can be performed in parallel. At this extreme, the determining factor for cost will be the max values. In contrast, consider a primitive (bus-based) network that forces all communication to be performed in serial. At this extreme, the determining factor will be the average values.

Apart from the penalty function $F(f)$, the cost function Π has two parameters: CCR and ITR. Because geometrical partitioning techniques generate superior load balance but inferior communication patterns compared to graph-based techniques, setting the parameters so that a given metric determines the overall cost would give predictable and trivial results. Instead, our goal was to vary these parameters within a range that makes all three contributors (load imbalance, edge cut, and migration) significant to overall cost. Thus, we strived to create a fictitious application and computing environment that was unbiased and did not favor a certain metric or partitioner. To achieve such an environment, we used the settings CCR $\in \{0.25, 0.5, 1.0\}$ and ITR $\in \{0.1, 0.25, 0.5, 1.0\}$ [2]

[2] Our settings for ITR are far from the settings recommended by the ParMetis authors (between 100 and 1000). However, we are not aiming for an optimal setting per se. Rather, we aim to set ITR and CCR to fulfill the requirements for our fictitious environment so that our hypothesis can be scrutinized.

For all applications except Quake, the number of partitions p tested was 8 and 16. For Quake, due to its much greater size, we used 32 and 64. For each time-step of each application, we applied both the average and max data sets to Π with all permutations of ITR, CCR, f, and p. All partitioners were forced to use RCB as their starting point. HyperGraph and *ParMetis* AdaptiveRepart were configured explicitly to minimize the relevant cost function, i.e., the ITR for AdaptiveRepart and MULT for Hypergraph were equal to the ITR in the cost function.

7.2 Results

7.2.1 Best Penalties and Algorithms

Figure 9 (left) shows the number of experiments for which each penalty performed the best. Although $f = 1$ was the best penalty on average and the quality of the result decreased when f increased, more than 60 percent of all configurations benefit from an $f > 1$.

Figure 9 (right) shows the number of experiments for which each partitioning algorithms performed the best. The best algorithm on average was AdaptiveRepart, followed by Hypergraph. However, Hypergraph is a relatively complex and therefore costly algorithm. Its run-times are roughly ten times as large as those for AdaptiveRepart. The other included algorithms exhibited success only in few configurations. Consequently, the obvious choice for a static algorithm that will most often provide good results was AdaptiveRepart.

7.2.2 Adaptive Partitioning vs. The Best Static Algorithm

We compared the results of adaptive partitioning with the best static algorithm chosen explicitly for the particular application configuration.

The cost of the adaptive partitioning for the configuration was divided with the cost of the best static partitioner for the same configuration. The cost was here taken as the sum of the cost over all time-steps.

The results are displayed in Table 3 through 6. Each table shows the results for all combinations of CCR, p and Π applied to max and average values, for a given ITR. For $(X,Y) = Z$, X refers to the best penalty for the adaptive partitioning, Y refers to the index of the best static algorithm, where 1=RCB, 2=RCB+MAP, 3=AdaptiveRepart, 4= HyperGraph, and 5=HSFC, and Z refers to the cost ratio described.

The average cost ratio was 100%, which means that our simple implementation of adaptive partitioning performed statistically as well as the best static algorithm for each particular configuration. The standard deviation was 8.5, which means that the results were reasonable stable.

ITR=HG_MULT=0.1

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,4)=49	(1,4)=100	(1,4)=48	(2,4)=100	(1,4)=46	(1,1)=104
Quake64	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=96
Mach8	(4,4)=87	(8,3)=98	(4,5)=98	(2,5)=109	(2,5)=120	(2,5)=119
Mach16	(2,4)=96	(4,4)=100	(4,4)=99	(8,4)=99	(8,5)=105	(4,5)=106
L-R8	(1,1)=103	(1,1)=102	(1,1)=102	(1,1)=102	(1,1)=103	(1,1)=101
L-R16	(8,2)=95	(8,2)=108	(8,2)=99	(1,2)=108	(2,2)=104	(1,2)=102
Conv8	(2,4)=101	(1,4)=98	(2,4)=98	(2,5)=97	(2,5)=101	(8,5)=103
Conv16	(2,4)=100	(2,4)=100	(2,4)=100	(2,4)=93	(4,5)=102	(4,5)=110
Sphe8	(4,3)=109	(4,3)=103	(4,3)=108	(4,3)=104	(4,3)=106	(1,2)=122
Sphe16	(2,4)=105	(4,4)=105	(2,4)=104	(4,4)=101	(2,4)=108	(1,2)=116
Shock8	(2,3)=84	(2,3)=99	(2,3)=86	(4,3)=100	(2,3)=89	(2,2)=97
Shock16	(2,4)=112	(4,3)=100	(4,4)=111	(8,3)=100	(2,4)=103	(2,2)=107

Table 3. Adaptive partitioning vs. the best static algorithm using ITR=HG_MULT=0.1 and all CCRs and both max and avg data sets. Notation: For $(X,Y)=Z$, X is the best penalty, Y is the index of the best static algorithm, where 1=RCB, 2=RCB+MAP, 3=ParMetis, 4=HyperGraph, 5=HSFC, and Z is the cost ratio of the best adaptive and the best static algorithm.

7.2.3 Adaptive Partitioning vs. AdaptiveRepart

We compared the results of adaptive partitioning with the industry standard, AdaptiveRepart, explicitly configured for the particular application configuration.

The cost of the adaptive partitioning for the configuration was divided with the cost of AdaptiveRepart for the same configuration. The cost was here taken as the sum of the cost over all time-steps.

The results are displayed in Table 7 through 10. Each table shows the results for all combinations of CCR, p and Π applied to max and average values, for a given ITR. For $(X,Y)=Z$, X refers to the best penalty for the adaptive partitioning, Y refers to the index of AdaptiveRepart (3), and Z refers to the cost ratio described.

The average cost ratio was 89.1%, which means that our simple implementation of adaptive partitioning performed statistically about 10 percent better than using the industry standard, AdaptiveRepart, routinely for all configurations. The standard deviation was 17, which means that the results varied more than for the comparison with the best static algorithm.

7.3 Conclusions and Future Work

Our hypothesis was that a simple implementation of adaptive partitioning can eliminate the problem of determining the best partitioning algorithm for new application configurations. Studying

ITR=HG_MULT=0.25

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,4)=96	(1,4)=100	(1,4)=93	(1,4)=100	(1,4)=64	(2,1)=105
Quake64	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=97
Mach8	(8,3)=95	(8,3)=102	(1,5)=98	(2,5)=107	(4,5)=107	(2,5)=122
Mach16	(2,3)=97	(2,3)=99	(2,3)=97	(4,3)=103	(4,5)=105	(8,5)=101
L-R8	(4,4)=100	(1,4)=106	(1,1)=102	(1,4)=104	(1,1)=105	(1,1)=105
L-R16	(2,4)=99	(2,3)=93	(2,4)=99	(4,3)=95	(4,4)=99	(8,3)=98
Conv8	(2,4)=96	(2,4)=96	(4,4)=93	(2,4)=87	(2,4)=84	(4,5)=83
Conv16	(2,4)=95	(2,4)=94	(2,4)=91	(2,4)=91	(4,5)=98	(4,5)=92
Sphe8	(2,3)=99	(4,3)=102	(2,3)=99	(8,2)=103	(4,3)=97	(1,2)=117
Sphe16	(2,4)=107	(4,3)=96	(2,4)=106	(4,2)=100	(2,4)=105	(1,2)=119
Shock8	(4,3)=100	(4,3)=99	(4,3)=100	(8,3)=99	(4,3)=102	(2,2)=98
Shock16	(2,3)=94	(8,3)=112	(2,3)=95	(8,3)=111	(4,3)=95	(1,2)=105

Table 4. Adaptive partitioning vs. the best static algorithm using ITR=HG_MULT=0.25 and all CCRs and both max and avg data sets. Notation: For $(X,Y)=Z$, X is the best penalty, Y is the index of the best static algorithm, where 1=RCB, 2=RCB+MAP, 3=ParMetis, 4=HyperGraph, 5=HSFC, and Z is the cost ratio of the best adaptive and the best static algorithm.

Tables 3 through 10, the conclusion is that the hypothesis is true. The cost ratios for the comparisons were 100% or less.

The results show that even a simple implementation of adaptive partitioning can effectively eliminate the significant problem of determining the best static algorithm for achieving well scalability of adaptive scientific applications. In fact, the simple implementation of adaptive partitioning performed statistically as good as the best static algorithm chosen explicitly for each particular application configuration. Moreover, choosing the simple implementation of adaptive partitioning over the industry standard, AdaptiveRepart, can on average increase performance for many applications with unknown characteristics.

These results lead to the overall conclusion that adaptive partitioning has great potential. However, it should be obvious that the main benefit of adaptive partitioning is *not* to improve on the parallel efficiency or to generally make larger and more complex simulations possible. Rather, the greatest benefit of adaptive partitioning for unstructured AMR lies in its potential to eliminate expensive and time consuming searches for suitable partitioning algorithms that provide the expected parallel efficiency for existing and new simulations with unknown characteristics. Consequently, future efforts should focus on tools for autmation.

Future work includes implementing a fully automatic adaptive partitioner as a part of the Zoltan library. This, in turn, implies research within many areas, such as control systems, software components, algorithm characterizations, and databases. Moreover, methods for determining optimal settings for adaptive partitioning, e.g. the value of f, are needed. For a thorough description of the fundamental parts and some initial implementations, see work by Johansson [17].

ITR=HG_MULT=0.5

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,4)=103	(1,4)=95	(1,4)=103	(1,4)=95	(1,4)=91	(1,4)=96
Quake64	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100
Mach8	(8,3)=108	(2,3)=108	(2,3)=96	(2,3)=105	(2,3)=96	(2,5)=111
Mach16	(4,3)=97	(2,3)=106	(4,3)=97	(2,3)=103	(2,3)=97	(8,3)=105
L-R8	(2,4)=97	(1,3)=104	(2,4)=97	(1,3)=99	(8,4)=103	(1,3)=108
L-R16	(2,3)=96	(2,3)=100	(2,3)=96	(1,3)=100	(2,3)=101	(1,3)=101
Conv8	(1,4)=96	(1,3)=89	(1,4)=96	(4,4)=90	(1,4)=88	(2,2)=86
Conv16	(1,3)=98	(1,3)=98	(1,3)=94	(2,3)=98	(2,3)=95	(2,3)=93
Sphe8	(2,3)=108	(4,3)=104	(2,3)=107	(4,2)=105	(1,3)=101	(1,2)=116
Sphe16	(1,3)=100	(4,3)=107	(1,3)=101	(2,2)=111	(1,3)=102	(1,2)=118
Shock8	(2,3)=107	(2,3)=104	(2,3)=106	(4,3)=104	(2,3)=106	(2,2)=100
Shock16	(4,3)=100	(8,3)=107	(4,3)=100	(8,3)=107	(4,3)=100	(2,2)=104

Table 5. Adaptive partitioning vs. the best static algorithm using ITR=HG_MULT=0.5 and all CCRs and both max and avg data sets. Notation: For $(X,Y)=Z$, X is the best penalty, Y is the index of the best static algorithm, where 1=RCB, 2=RCB+MAP, 3=ParMetis, 4=HyperGraph, 5=HSFC, and Z is the cost ratio of the best adaptive and the best static algorithm.

ITR=HG_MULT=1.0

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,4)=87	(1,4)=98	(1,4)=88	(1,4)=98	(1,4)=88	(1,4)=98
Quake64	(1,4)=100	(1,4)=103	(1,4)=100	(1,4)=100	(1,4)=100	(1,4)=100
Mach8	(1,3)=98	(4,3)=108	(1,3)=98	(8,3)=104	(1,3)=97	(8,3)=103
Mach16	(2,3)=101	(8,3)=102	(2,3)=96	(8,3)=102	(8,3)=101	(8,3)=102
L-R8	(1,3)=97	(8,3)=96	(1,3)=97	(8,3)=98	(4,3)=103	(8,3)=102
L-R16	(4,3)=94	(1,3)=100	(4,3)=94	(1,3)=100	(4,3)=94	(1,3)=99
Conv8	(1,4)=99	(1,4)=99	(1,4)=98	(1,4)=103	(1,4)=98	(1,4)=99
Conv16	(1,3)=96	(1,3)=104	(1,3)=100	(1,3)=99	(1,3)=100	(1,3)=102
Sphe8	(4,3)=101	(4,3)=105	(4,3)=101	(8,3)=104	(4,3)=101	(1,2)=109
Sphe16	(2,4)=98	(2,2)=92	(2,4)=98	(4,2)=105	(4,4)=101	(2,2)=114
Shock8	(2,3)=108	(1,3)=98	(2,3)=107	(4,3)=100	(4,3)=106	(2,3)=97
Shock16	(1,3)=106	(4,3)=108	(1,3)=110	(4,3)=108	(2,3)=103	(2,3)=105

Table 6. Adaptive partitioning vs. the best static algorithm using ITR=HG_MULT=1.0 and all CCRs and both max and avg data sets. Notation: For $(X,Y)=Z$, X is the best penalty, Y is the index of the best static algorithm, where 1=RCB, 2=RCB+MAP, 3=ParMetis, 4=HyperGraph, 5=HSFC, and Z is the cost ratio of the best adaptive and the best static algorithm.

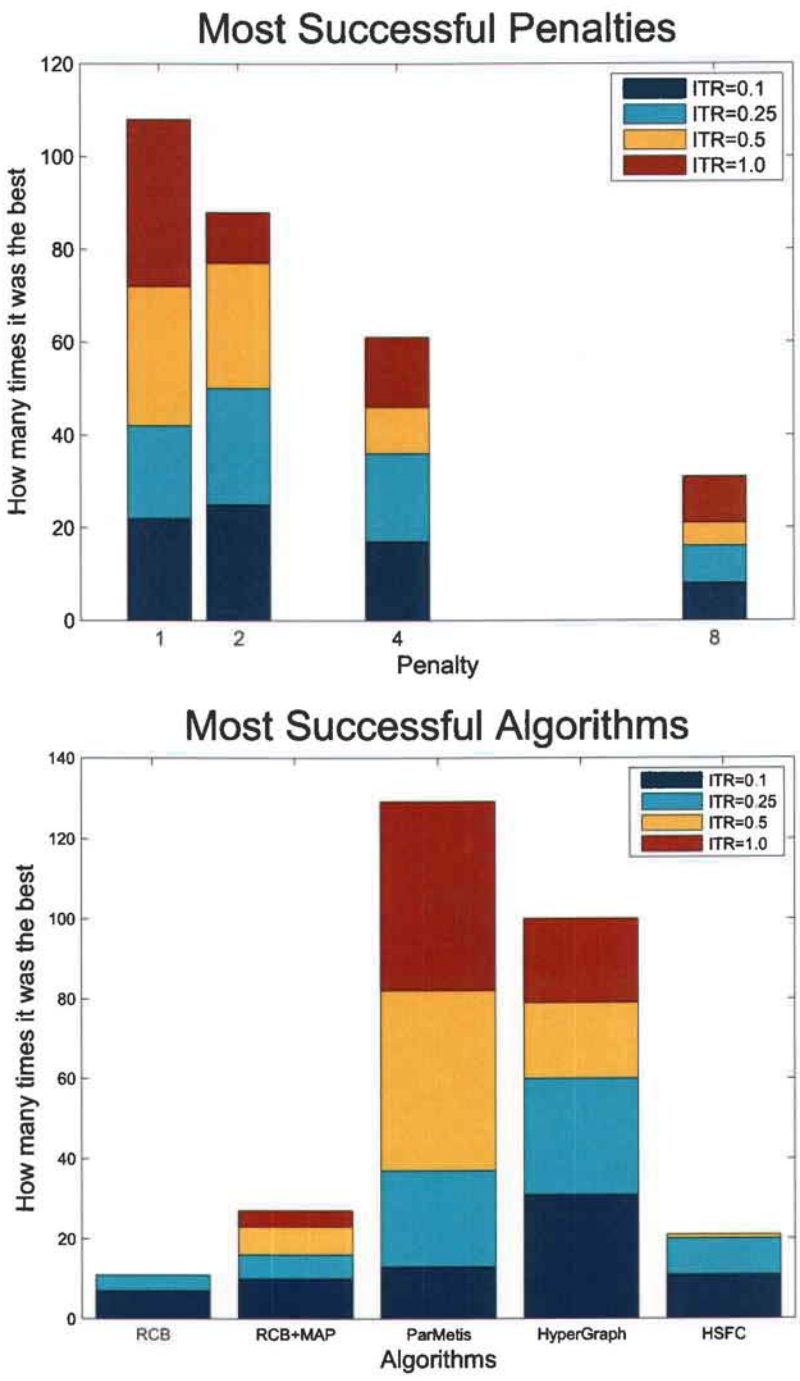

Figure 9. Best penalties (above) and algorithms (below) for the applications tested in this study.

ITR=HG_MULT=0.1

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,3)=10	(1,3)=44	(1,3)=11	(2,3)=49	(1,3)=12	(1,3)=53
Quake64	(1,3)=60	(1,3)=63	(1,3)=62	(1,3)=67	(1,3)=64	(1,3)=69
Mach8	(4,3)=86	(8,3)=98	(4,3)=87	(2,3)=97	(2,3)=88	(2,3)=74
Mach16	(2,3)=85	(4,3)=93	(4,3)=89	(8,3)=93	(8,3)=90	(4,3)=96
L-R8	(1,3)=75	(1,3)=79	(1,3)=70	(1,3)=71	(1,3)=64	(1,3)=58
L-R16	(8,3)=71	(8,3)=98	(8,3)=72	(1,3)=91	(2,3)=71	(1,3)=75
Conv8	(2,3)=86	(1,3)=78	(2,3)=84	(2,3)=75	(2,3)=76	(8,3)=53
Conv16	(2,3)=65	(2,3)=63	(2,3)=66	(2,3)=63	(4,3)=69	(4,3)=63
Sphe8	(4,3)=109	(4,3)=103	(4,3)=108	(4,3)=104	(4,3)=106	(1,3)=85
Sphe16	(2,3)=86	(4,3)=102	(2,3)=86	(4,3)=98	(2,3)=91	(1,3)=85
Shock8	(2,3)=84	(2,3)=99	(2,3)=86	(4,3)=100	(2,3)=89	(2,3)=82
Shock16	(2,3)=100	(4,3)=100	(4,3)=100	(8,3)=100	(2,3)=94	(2,3)=93

Table 7. Adaptive partitioning vs. AdaptiveRepart, using ITR=HG_MULT=0.1 and all CCRs and both max and avg data sets. Notation: For $(X,Y) = Z$, X is the best penalty, Y is the index of AdaptiveRepart (3), and Z is the cost ratio of the best adaptive and AdaptiveRepart.

ITR=HG_MULT=0.25

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,3)=80	(1,3)=89	(1,3)=78	(1,3)=90	(1,3)=54	(2,3)=91
Quake64	(1,3)=44	(1,3)=85	(1,3)=45	(1,3)=86	(1,3)=47	(1,3)=85
Mach8	(8,3)=95	(8,3)=102	(1,3)=94	(2,3)=101	(4,3)=88	(2,3)=87
Mach16	(2,3)=97	(2,3)=99	(2,3)=97	(4,3)=103	(4,3)=98	(8,3)=99
L-R8	(4,3)=93	(1,3)=104	(1,3)=92	(1,3)=102	(1,3)=88	(1,3)=92
L-R16	(2,3)=85	(2,3)=93	(2,3)=86	(4,3)=95	(4,3)=86	(8,3)=98
Conv8	(2,3)=90	(2,3)=90	(4,3)=86	(2,3)=80	(2,3)=77	(4,3)=63
Conv16	(2,3)=76	(2,3)=80	(2,3)=73	(2,3)=78	(4,3)=78	(4,3)=75
Sphe8	(2,3)=99	(4,3)=102	(2,3)=99	(8,3)=102	(4,3)=97	(1,3)=80
Sphe16	(2,3)=96	(4,3)=96	(2,3)=96	(4,3)=97	(2,3)=96	(1,3)=89
Shock8	(4,3)=100	(4,3)=99	(4,3)=100	(8,3)=99	(4,3)=102	(2,3)=84
Shock16	(2,3)=94	(8,3)=112	(2,3)=95	(8,3)=111	(4,3)=95	(1,3)=95

Table 8. Adaptive partitioning vs. AdaptiveRepart, using ITR=HG_MULT=0.25 and all CCRs and both max and avg data sets. Notation: For $(X,Y) = Z$, X is the best penalty, Y is the index of AdaptiveRepart (3), and Z is the cost ratio of the best adaptive and AdaptiveRepart.

ITR=HG_MULT=0.5

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,3)=54	(1,3)=78	(1,3)=55	(1,3)=79	(1,3)=51	(1,3)=82
Quake64	(1,3)=64	(1,3)=91	(1,3)=65	(1,3)=91	(1,3)=67	(1,3)=92
Mach8	(8,3)=108	(2,3)=108	(2,3)=96	(2,3)=105	(2,3)=96	(2,3)=103
Mach16	(4,3)=97	(2,3)=106	(4,3)=97	(2,3)=103	(2,3)=97	(8,3)=105
L-R8	(2,3)=93	(1,3)=104	(2,3)=93	(1,3)=99	(8,3)=99	(1,3)=108
L-R16	(2,3)=96	(2,3)=100	(2,3)=96	(1,3)=100	(2,3)=101	(1,3)=101
Conv8	(1,3)=93	(1,3)=89	(1,3)=92	(4,3)=89	(1,3)=83	(2,3)=81
Conv16	(1,3)=98	(1,3)=98	(1,3)=94	(2,3)=98	(2,3)=95	(2,3)=93
Sphe8	(2,3)=108	(4,3)=104	(2,3)=107	(4,3)=103	(1,3)=101	(1,3)=84
Sphe16	(1,3)=100	(4,3)=107	(1,3)=101	(2,3)=105	(1,3)=102	(1,3)=91
Shock8	(2,3)=107	(2,3)=104	(2,3)=106	(4,3)=104	(2,3)=106	(2,3)=95
Shock16	(4,3)=100	(8,3)=107	(4,3)=100	(8,3)=107	(4,3)=100	(2,3)=99

Table 9. Adaptive partitioning vs. AdaptiveRepart, using ITR=HG_MULT=0.5 and all CCRs and both max and avg data sets. Notation: For $(X,Y) = Z$, X is the best penalty, Y is the index of AdaptiveRepart (3), and Z is the cost ratio of the best adaptive and AdaptiveRepart.

ITR=HG_MULT=1.0

App.	Max, 0.25	Avg, 0.25	Max, 0.5	Avg, 0.5	Max, 1.0	Avg, 1.0
Quake32	(1,3)=49	(1,3)=69	(1,3)=50	(1,3)=70	(1,3)=51	(1,3)=72
Quake64	(1,3)=57	(1,3)=85	(1,3)=58	(1,3)=84	(1,3)=59	(1,3)=85
Mach8	(1,3)=98	(4,3)=108	(1,3)=98	(8,3)=104	(1,3)=97	(8,3)=103
Mach16	(2,3)=101	(8,3)=102	(2,3)=96	(8,3)=102	(8,3)=101	(8,3)=102
L-R8	(1,3)=97	(8,3)=96	(1,3)=97	(8,3)=98	(4,3)=103	(8,3)=102
L-R16	(4,3)=94	(1,3)=100	(4,3)=94	(1,3)=100	(4,3)=94	(1,3)=99
Conv8	(1,3)=92	(1,3)=95	(1,3)=91	(1,3)=99	(1,3)=91	(1,3)=94
Conv16	(1,3)=96	(1,3)=104	(1,3)=100	(1,3)=99	(1,3)=100	(1,3)=102
Sphe8	(4,3)=101	(4,3)=105	(4,3)=101	(8,3)=104	(4,3)=101	(1,3)=90
Sphe16	(2,3)=88	(2,3)=89	(2,3)=88	(4,3)=94	(4,3)=91	(2,3)=88
Shock8	(2,3)=108	(1,3)=98	(2,3)=107	(4,3)=100	(4,3)=106	(2,3)=97
Shock16	(1,3)=106	(4,3)=108	(1,3)=110	(4,3)=108	(2,3)=103	(2,3)=105

Table 10. Adaptive partitioning vs. AdaptiveRepart, using ITR=HG_MULT=1.0 and all CCRs and both max and avg data sets. Notation: For $(X,Y) = Z$, X is the best penalty, Y is the index of AdaptiveRepart (3), and Z is the cost ratio of the best adaptive and AdaptiveRepart.

References

[1] http://www.openchannelfoundation.org/projects/GeoFEST.

[2] http://www.openchannelfoundation.org/projects/pyramid.

[3] S.W. Bova, D.D. Copps, and C.K. Newman. Calore, a computational heat transfer program. Technical Report SAND2005-0551, Sandia National Laboratory, Albuquerque, NM, USA, 2005.

[4] K. Budge and J. Peery. Experiences Developing ALEGRA: A C++ Coupled Physics Framework, 1996. In M.E Henderson, C. R. Anderson, and S. L. Lyons, editors, Object Oriented Methods for Interoperable Scientific and Engineering Computing.

[5] G. F. Carey, M. Anderson, B. Carnes, and B. Kirk. Some aspects of adaptive grid technology related to boundary and interior layers. *J. Comput. Appl. Math.*, 166(1):55–86, 2004.

[6] Graham F. Carey, William Barth, Juliette A. Woods, Ben Kirk, Michael L. Anderson, Sum Chow, and Wolfgang Bangerth. Modeling error and constitutive relations in simulation of flow and transport. *International Journal for Numerical Methods in Fluids*, 46:1211–1236, 2004.

[7] B. R. Carnes and G. F. Carey. Estimating spatial and parameter error in parameterized nonlinear reaction-diffusion equations. *Communications in Numerical Methods in Engineering*, 23:834–854, 2007.

[8] Bruce Carter and et. al. Parallel FEM simulation of crack propagation – challenges, status, and perspectives. In *Lecture Notes in Computer Science 1800*, pages 443–449. Springer-Verlag, 2000.

[9] U.V. Catalyurek, E.G. Boman, K.D. Devine, D. Bozdag, R.T. Heaphy, and L.A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Proc. of 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*. IEEE, 2007. Best Algorithms Paper Award.

[10] M. I. Chen and T. G. Trucano. ALEGRA validation studies for regular, mach, and double mach shock reflection in gas dynamics. (SAND2002-2240), September 2002.

[11] G. F. Carey Chow, S. and M. L. Anderson. Finite element approximations of a glaciology problem. *Mathematical Modelling and Numerical Analysis*, 38(5):741–756, 2004.

[12] R. Deiterding, R. Radovitzky, L. Noels S. Mauch, J.C. Cummings, and D.I. Meiron. A virtual test facility for the efficient simulation of solid material response under strong shock and detonation wave loading. *Engineering with Computers*, 22 (3–4):325–347, 2006.

[13] Ralf Deiterding. AMROC. http://amroc.sourceforge.net/ VTF, California Institute of Technology, CA, USA, 2003.

[14] Ralf Deiterding. Detonation simulation with the AMROC framework. In *Forschung und wissenschaftliches Rechnen: Beiträge zum Heinz-Billing-Preis 2003*, pages 63–77. Gesellschaft für Wiss. Datenverarbeitung, 2004.

[15] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[16] Richard Drake et. al. Computational shock multiphysics. http://www.cs.sandia.gov/capabilities/-ComputationalShockMultiphysics/index.html, Sandia National Laboratory.

[17] Henrik Johansson and Johan Steensland. A characterization of a hybrid and dynamic partitioner for SAMR applications. Technical Report 2004-009, Uppsala University, IT, Dept. of scientific computing, Uppsala, Sweden, 2004.

[18] G. Karypis, K. Schloegel, and V. Kumar. PARMETIS - parallel graph partitioning and sparse matrix ordering library, version 2.0. Univ. of Minnesota, Minneapolis, MN, 1998.

[19] J. D. Murray, J. Cook, R. Tyson, and S. R. Lubkin. Spatial pattern formation in biology: I. dermal wound healing. II. bacterial patterns. *Journal of the Franklin Institute*, 335(2):303–332, 1998.

[20] C. D. Norton, G. A. Lyzenga, J. W. Parker, and E. R. Tisdale. Developing Parallel GeoFEST(P) using the PYRAMID AMR Library. In *Proceedings NASA Earth Science Technology Conference*, Palo Alto, CA, 2004. http://esto-doc.gsfc.nasa.gov/conferences/estc2004/papers/a2p2.pdf.

[21] J. W. Peterson, G. F. Carey, and B. T. Murray. Adaptive grid strategies for FEM simulations of double-diffusive convection in porous media. *Journal of Comm. Numer. Meth. Eng.*, Special Issue, In preparation.

[22] K. Schloegel, G. Karypis, and V. Kumar. A unified algorithm for load-balancing adaptive scientific simulations. In *Proceedings of Supercomputing 2000*, 2000.

[23] Larry A. Schoof and Victor R. Yarberry. Exodus II: A finite element data model. Technical Report SAND92-2137, Sandia National Laboratory, Albuquerque, NM, USA, 1994.

[24] Mausumi Shee. Evaluation and optimization of load balancing/distribution techniques for adaptive grid hierarchies. M.S. Thesis, Graduate School, Rutgers University, NJ, 2000 http://www.caip.rutgers.edu/TASSL/Thesis/-mshee-thesis.pdf, 2000.

[25] Johan Steensland. *Efficient partitioning of dynamic structured grid hierarchies*. PhD thesis, Uppsala University, 2002.

[26] Johan Steensland, Sumir Chandra, and Manish Parashar. An application-centric characterization of domain-based SFC partitioners for parallel SAMR. *IEEE Transactions on Parallel and Distributed Systems*, December:1275–1289, 2002.

[27] Johan Steensland and John Peterson. A study of dynamically adaptive partitioning for AMR. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07)*, pages 503–509. CSREA Press, 2007.

[28] The Virtual Test Facility. http://www.cacr.caltech.edu/asc/wiki, Oct. 2006.

[29] M. Vetter and R. Sturtevant. Experiments on the Richtmyer-Meshkov instability on a air/SF6 interface. *Shock Waves*, 4(5):247–252, 1995.

[30] C Walshaw. Jostle homepage, http://www.gre.ac.uk/~c.walshaw/jostle/. 2000.

[31] Chris Walshaw. *Variable partitioning inertia: graph repartitioning and load-balancing for adaptive meshes*. Wiley book series on parallel and distributed computing. Wiley, 2007. To appear.

Distribution:

1	Johan Steensland Vassvägen 1, SE-645 44 Strängnäs Sweden	
1	Jaideep Ray, 08964	MS 9159
1	Technical Library, 99536	MS 0899
2	Central Technical Files 08945-1	MS 9018

Sandia National Laboratories