A Partitioner-Centric Model for SAMR Partitioning Trade-Off Optimization: Part I*

Johan Steensland and Jaideep Ray

11th September 2003

Abstract

Optimal partitioning of structured adaptive mesh applications necessitates dynamically determining and optimizing for the most time-inhibiting factor, such as load imbalance and communication volume. But any trivial monitoring of an application primarily evaluates the current partitioning, rather than the inherent nature of the grid hierarchy itself. We present an analytical model that given a structured adaptive grid determines, *ab initio*, to what extent the partitioner should focus on optimizing load imbalance or communication volume to reduce execution time. This model contributes to the meta-partitioner, our ultimate aim of being able to select and configure the optimal partitioner based on the mesh configuration, the simulation and computer characteristics. We validate the predictions of this model by comparing them with actual measurements (via traces) from five different adaptive simulations. The results show that the proposed model generally captures the inherent optimization-need in SAMR applications. We conclude that our model is a useful contribution, since tracking and adapting for the dynamic behavior of such applications potentially lead to a large decrease in execution times.

1 Introduction and Background

Significantly improving the scalability of large structured adaptive mesh refinement (SAMR) applications requires sophisticated capabilities for utilizing the underlying parallel computer's resources in the most efficient way. The *meta-partitioner* [34, 35] is a tool providing such capabilities. Previous research has offered design, proofs-of-concept and evaluation of major components. This paper presents a new model for classifying SAMR application and parallel computer system state to best provide a partitioner with information required for trade-off optimization, and validates this model by showing its effectiveness to accurately capture the dynamic behavior of five vastly different SAMR applications. Particular attention will be paid to dimension I in the classification space (characterizing the relative importance of achieving a good load-balance versus reducing the overall communication cost) and an application that simulates the Richtmyer-Meshkov instabilities using an explicit time-stepping and a finite volume scheme.

The presented work is part of the ongoing research project [34, 35, 36, 10, 37] with the overall goal of engineering a *dynamically adaptive* meta-partitioner for SAMR grid hierarchies capable of selecting the most appropriate partitioning strategy at runtime based on current system and application state. Such a meta-partitioner can significantly reduce the execution time of SAMR applications [13, 12, 11].

Dynamically adaptive mesh refinement (AMR) [39] methods for the numerical solution to partial differential equations (PDE's) [7, 8, 31] employ locally optimal approximations, and can yield highly advantageous ratios for cost/accuracy when compared to methods based on a static uniform mesh. These techniques seek to improve the accuracy of the solution by dynamically refining the computational grid in regions with large local solution error. Structured adaptive mesh refinement methods are based on uniform patch-based refinements overlaid on a structured coarse grid, and provide an alternative to the general, unstructured AMR approach. These methods are being widely used for adaptive PDE solutions in many domains, including computational fluid dynamics [2, 6, 28], numerical relativity [14, 30], astrophysics [1, 9, 23], and subsurface modeling and oil reservoir simulation [44, 25]. Methods based on SAMR can lead to computationally efficient implementations as they require uniform operations on regular arrays and exhibit structured communication patterns. Furthermore, these methods tend to be easier to implement and manage due to their regular structure. Distributed implementations of these methods offer the potential for accurate solutions of physically realistic models of complex physical phenomena. However, they also pose new challenges in dynamic resource allocation, data-distribution, load-balancing,

*This paper was presented at the 4th Annual Symposium of the Los Alamos Computer Science Institute (LACSI04) and appears in the proceedings on CD-ROM.

and runtime management. Critical among these is the partitioning of the adaptive grid hierarchy to balance load, optimize communication and synchronization, minimize data migration costs, and maximize grid quality (e.g. aspect ratio) and available parallelism.

The primary motivation for the research presented in this paper, as well as the research effort at large, is the observation that in the case of parallel SAMR, *no single partitioning scheme performs the best* for all types of applications and systems. Even for a single application, the most suitable partitioning technique depends on input parameters and the application's runtime state [29, 35]. This necessitates an adaptive management of these dynamic applications at runtime. This includes using application runtime state to select and configure the partitioning strategy to maximize performance. The goal of the adaptive *meta-partitioner* is to provide such a capability for parallel SAMR applications.

Large scale SAMR applications place vastly different requirements on the partitioning strategy to enable efficient utilization of computer resources and consequently good scalability. In some scenarios, this means focusing on optimizing load balance; in others, on lowering the interprocessor communication costs [37]. Hence, a means to classify these requirements in a way that conform to the partitioner is crucial.

The support for tuning and choosing trade-off impacts maturing in graph-based partitioning techniques [17, 32, 15] for *unstructured* AMR, is so far lacking in the field of *structured* AMR. Whereas recent research efforts have targeted the scalability of *specific* applications executing on *specific* parallel computers [5, 45, 27], our line of research is in the opposite direction; the development of a *general* partitioning tool enabling good scalability for *general* SAMR applications executing on *general* parallel computers. As a consequence, we carefully engineer the components of the adaptive meta-partitioner with the basic requirement is that the components should be able to adapt to changing requirements derived from the monitoring of system and application state.

In this paper, we advance towards the meta-partitioner by introducing a key component: a model for the classification of application and system state. The key contributions are (1) a partitioner-centric classification space, designed to conform to the partitioner, (2) a detailed mathematical model for sampling and translating these samples of the given application parameters (such as the grid hierarchy, the number of processors and so forth) and system parameters (such as CPU speed and communication bandwidth) into dimension I of the partitioner-centric classification space, and (3) an experimental evaluation and validation of this mathematical model showing its effectiveness to accurately capture the dynamic behavior of five vastly different SAMR applications.

2 SAMR and Related Work

2.1 Introduction to SAMR

In the case of SAMR methods, dynamic adaptation is achieved by tracking regions in the domain that require higher resolution and dynamically overlaying finer grids on these regions. These techniques start with a coarse base grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain with large solution error, requiring additional resolution, are identified and refined. Refinement proceeds recursively so that the refined regions requiring higher resolution are similarly tagged and even finer grids are overlaid on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy.

Software infrastructures for SAMR worth mentioning are e.g. Paramesh [21, 22], a FORTRAN library for parallelization of and adding adaption to existing serial structured grid computations, SAMRAI [18, 45] a C++ object-oriented framework for implementing parallel structured adaptive mesh refinement simulations, and GrACE [26] and CHOMBO[3], both of which are adaptive computational and data-management engines for enabling distributed adaptive mesh-refinement computations on structured grids.

2.2 Partitioning SAMR Grid Hierarchies

Parallel implementations of SAMR methods present interesting challenges in dynamic resource allocation, data-distribution, load-balancing, and runtime management. The overall efficiency of parallel SAMR applications is limited by the ability to partition the underlying grid hierarchies at runtime to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load. A critical requirement when partitioning these adaptive grid hierarchies is the maintenance of logical locality, both across different levels of the hierarchy under expansion and contraction of the adaptive grid structure, and within partitions of grids at all levels when they are decomposed and mapped across processors. The former enables efficient computational access to the grids and minimizes the parent-child (inter-level) communication overheads, while the latter minimizes overall communication and synchronization overheads. Furthermore, application adaptation results in grids being dynamically created, moved and deleted at runtime, making it necessary to efficiently repartition the hierarchy "on the fly" so that it continues to meet these goals.

Partitioners for SAMR grid hierarchies can be classified as patch-based, domain-based, or hybrid.[1]

In the case of *patch-based partitioners* [5, 19], distribution decisions are independently made for each newly created

[1]Note that this paper focuses exclusively on partitioning techniques for adaptive structured grids. Similar classification and comparative studies for unstructured-grid/mesh partitioning and dynamic load-balancing have been investigated in the literature [41, 43].

grid. A grid may be kept on the local processor or entirely moved to another processor. If the grid is too large, it may be split. Grids may also be distributed uniformly over all processors. The SAMR framework SAMRAI [18, 45] (based on the LPARX [4] and KeLP [16] model) fully supports patch-based partitioning. The distribution scheme maps the patches at a refinement level of the AMR hierarchy across processors. The advantages are manageable load imbalance and re-partitioning at re-griding could be avoided. Shortcomings inherent in patch-based techniques are communication serialization bottlenecks, inability to exploit available parallelism both across grids at the same level and different levels [35].

Domain-based partitioners [24, 29, 40, 34] partition the physical domain, rather than the grids themselves. The domain is partitioned along with all contained grids on all refinement levels. The advantages are elimination of inter-level communication and better exploiting of all available parallelism. The disadvantages are intractable load imbalance for deep hierarchies and the occurrence of "bad cuts" leading to increased overhead costs [35].

Hybrid partitioners [24, 40, 20] combining patch-based and domain-based approaches, can be used for coping with the shortcomings present in these techniques. They use a 2-step partitioning approach. The first step uses domain-based techniques to generate meta-partitions, which are mapped to a group of processors. The second step uses a combination of domain and patch based techniques to optimize the distribution of each meta-partition within its processor group.

Developed at Uppsala University, Sweden and Rutgers University, New Jersey, USA, `Nature+Fable` (**Natur**al **Re**gions **+** **F**r**a**ctional blocking and **b**i-**le**vel partitioning) [35] aims to be the best possible tool for partitioning SAMR grid hierarchies. It hosts a variety of hybrid partitioning options. All involved parts are engineered to be components of the meta-partitioner. Thus, they offer carefully designed parameters to steer component behavior enabling adaptation to varying partitioning requirements. As `Nature+Fable` matures, it is intended to transform it into the meta-partitioner. Hence, the partitioning tool `Nature+Fable` is a step towards a complete implementation.

`Nature+Fable` separates homogeneous, un-refined (Hue) and complex, refined (Core) domains of the grid hierarchy and clusters refinement levels into *bi-levels* [35]. The Hues contain the portions of the grid hierarchy without refinements; consequently they contain only parts of the base grid (refinement level 0). The Cores contain the portions of the grid where refinements are present. The Cores are separated from the Hues in a strictly domain-based fashion, meaning that each Core contains a portion of the base grid and all its overlaid, refined grids. Expert blocking algorithms are used for the Hues. The Cores are subjected to a coarse partitioning, creating "easy-to-block" bi-levels. Then the same expert algorithms operating on the Hues are used for these bi-levels.

3 Previous Approaches

This section provides a problem description and a survey of relevant previous research efforts including the *octant approach* [35] and the *ArMADA framework* [13]. While conceptually similar to the octant approach in that it strives to capture the application and system state for optimizing the partitioning, our model is quantitative and is rigorously derived from a set of assumptions, primarily borne out by observations in SAMR simulations.

The PAC-triple defines the SAMR application (A), parallel computer systems (C), and the partitioner (P). While the A and C components are highly dynamic entities, the P component, however, is usually selected once and for all. A static P component, i.e., not exploiting the dynamic nature of the A and C components, can seriously inhibit the scalability and the prospect of reducing execution time. The octant approach and the meta-partitioner are means for allowing fully dynamic PAC:s. That is, it provides capabilities for

$$P_t = f(A_t, C_t),$$

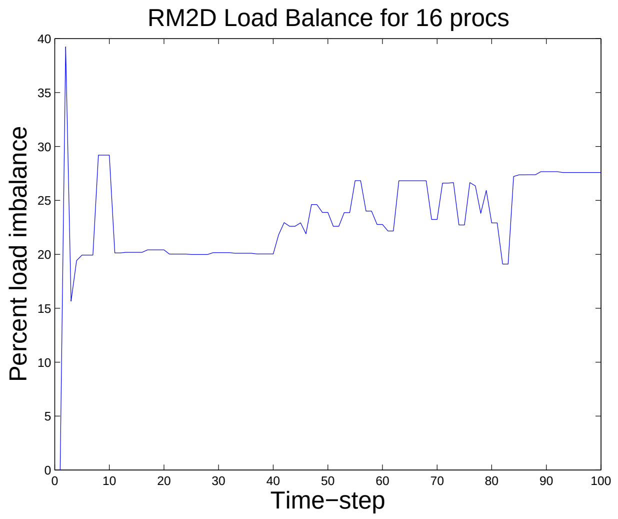
meaning that the partitioner P at a particular time t should be a function (f) of the application A and computer system C state at that particular time.

To illustrate the dynamic behavior of SAMR applications, consider a static choice of P for the RM2D application (further described in section 5) illustrated in Figure 1. This figure exhibits load imbalance and communication amount as a function of time. Clearly, with a dynamic selection of P (a fully dynamic PAC) appropriately reflecting the inherent dynamics of the application, the total execution time could have been reduced.

The octant approach is depicted in Figure 3 (left). It is an extension of the quadrant approach [38] and constitutes an important part in the conceptual meta-partitioner, illustrated in Figure 2. The octant approach is a discrete classification space and a set of rules for selecting and configuring the most appropriate partitioning technique, based on application and system state. The model consists of the following: (a) classifying application state, (b) classifying system state, (c) combining the results in (a) and (b), (d) translating the result in (c) into an octant, and finally (e) mapping from octant onto partitioning technique.

This model has evolved over the years to reflect and incorporate our growing intuition and experience. It requires determining the classification space, i.e., the actual axes in the cube (of octants), and the characterization of involved partitioners.

However, the model does not say *how* — only *what* and *why*; it outlines what seems to be a promising concept. For example, it does not define a "state". It does not offer ways to compute it, nor does it list the necessary variables. It does not define how to translate the information regarding state into an



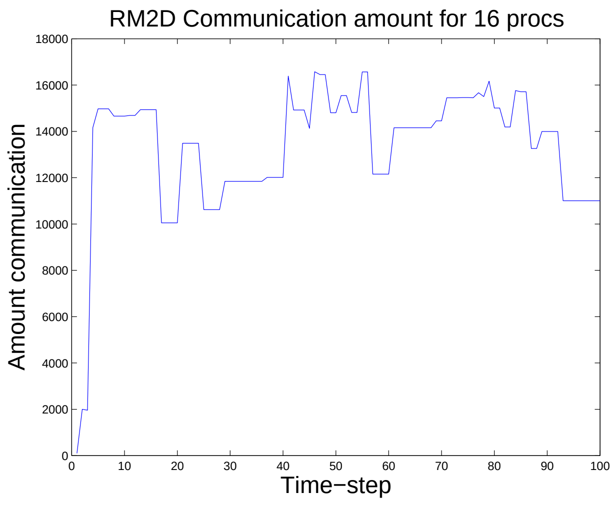


Figure 1: The dynamic behavior of the RM2D SAMR application illustrated by a static selection of partitioner (P). Clearly, with a dynamic selection of P (a fully dynamic PAC) appropriately reflecting the inherent dynamics of the application, the total execution time could have been reduced.

octant. The model in itself does not provide the specifics about the mapping from octant onto partitioning technique, but such a mapping has been derived for a set of partitioners [36].

The ArMADA framework offers a first attempt at an actual implementation of the model. ArMADA disregards the system component and uses simple box operations like e.g. volume to surface ratio on the grid hierarchy to determine the corresponding octant. The classification is relative to the previous state (octant) and the mappings used were those previously derived [36] . The project provided an important proof of concept: even with such a simple model, execution times were reduced.

In the following, we examine each of the dimensions of the classification space in the octant approach in detail. The original thought was to capture the current state of the application executing on the system, with the selection and configuration of the best partitioning technique determined by this state. We show below that this space is inadequate for the purpose.

3.1 The Refinement Pattern

The first dimension of the cube, *localized* — *scattered*, reflects the nature of the refinement pattern. It has been shown for unstructured meshes that diffusion schemes are suited to scattered patterns while scratch/re-map work well for strongly localized patterns. Evidently, this quantity plays a role. But how does it affect partitioning of structured grids?

Assuming a strictly domain-based partitioning technique, the refinement pattern is crucial. A small base-grid, many processors, and many levels of refinement cause domain-based techniques to generate intractable amounts of load imbalance. However, the case is better with scattered refinement, and worsens with strongly localized refinement. Consequently, in the case of an already stretched domain-based, badly load-balanced scenario, refinement pattern is a crucial quantity to sample. However, in other cases, information about the refinement pattern might be of little consequence.

3.2 Time Domination

The second dimension of the cube, *computation dominated* — *communication dominated*, reflects whether run-time of an application executed on a machine (at a given moment) is dominated by communication or by computation. The idea is to distinguish between scenarios where load balance should be the target for optimization (computation dominated) and where communication pattern/amount (communication dominated) should be the main target.

This is problematic. First, the other axes in the cube (refinement pattern and activity dynamics) reflect the state of the application and system *independently* of partitioning technique and current distribution of the hierarchy. It is impossible to determine the time domination axis without involving assumptions about how the grid is distributed.

This dependency accounts for a “circle” in the model. We are supposed to classify the state of the application / system to select the partitioner. But this classification is strongly influenced by the partitioner that was used to achieve the current domain decomposition in the first place. Consequently, in the worst case, the partitioner is classifying itself and the answer cannot be regarded to be very general. For example, assume the classification of the application / system is “communication dominated”, what this means is really that *this* particular partitioner generated lots of communication for the application / system. Thus this information is a classification of a

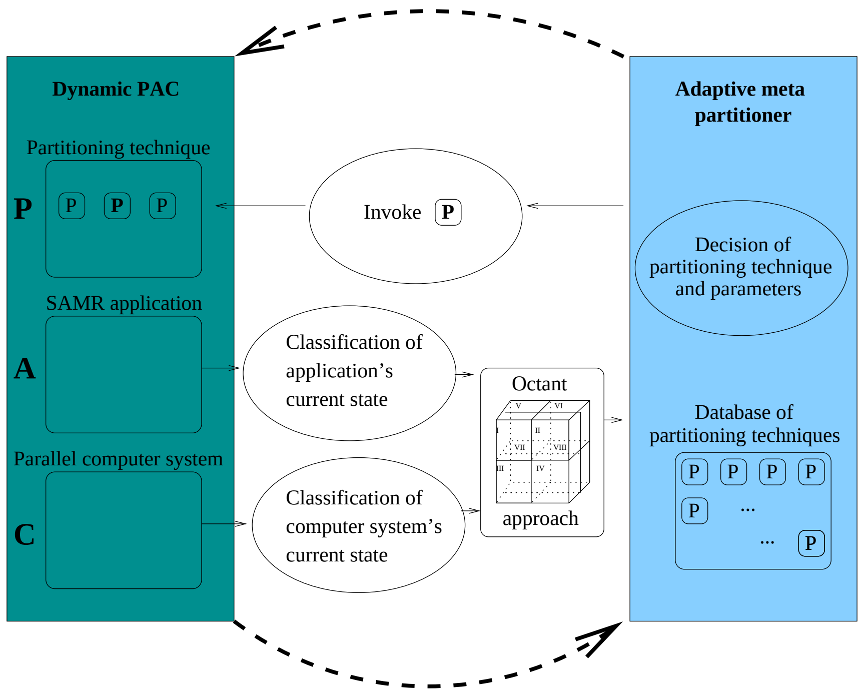

Dynamic PAC
Partitioning technique
P
P P P
SAMR application
A
Parallel computer system
C
Invoke P
Classification of application's current state
Classification of computer system's current state
Octant approach
Adaptive meta partitioner
Decision of partitioning technique and parameters
Database of partitioning techniques
P P P P
P ...
... P

Figure 2: The conceptual meta-partitioner incorporating the octant approach. The most appropriate partitioner is selected and configured, based on the current application and system state. Consequently, fully dynamic PAC:s are enabled.

PAC-triple, while the other dimensions in the cube really classify the AC-double. Moreover, the numerical algorithm has a direct impact on where the time is spent.

Furthermore, the original idea of interpreting a computation dominated simulation as one requiring an optimizing of load balance is flawed. A parallel SAMR application consists of a series of steps separated by synchronization points. Load imbalance among processors dictates the cost of (time spent at) these synchronizations. But this load imbalance is the *total* load imbalance, i.e. it is the sum of computational load imbalance and communicational load imbalance. Assume for example a perfectly (computationally) load-balanced application with a localized refinement. When the processors involved with the localized refinement start communicating, they will take more time than their counterparts working on the non-refined parts of the hierarchy. As a consequence, there will be lots of load imbalance (lots of time spent waiting at synchronization points) despite a perfect computational load balance.

To summarize the complexity of this dimension:

1. An application spending lots of time at synchronization points is badly load balanced. Based on this information only, there is no way of telling whether optimizing (computational) load balance or communication would be the best remedy.

2. Synchronization points is implemented by variants of MPI_Wait. Thus, they are part of a (global) communication. This means that a communication dominated application might be in great need of optimizing load balance.

We conclude that this dimension must be reformulated so as to explicitly expose the cause of the load imbalance i.e. whether the imbalance is due to unbalanced load distribution or unbalanced communication schedules. There are ways to steer the trade-off communication/load imbalance in tools like `Nature+Fable`. We must find a suitable model for giving these tools pertinent information to base their trade-offs on.

3.3 Activity Dynamics

The third dimension reflects how fast things are *changing* in the solution. The idea is that high activity dynamics implies more frequent regridding. Consequently, partitioning speed and low data migration costs are crucial. This is not necessarily true.

An SAMR application might have a very long compute cycle, meaning that there may be minutes between synchronization points and possible re-gridding. Such an application might easily show significant activity dynamics from time-step to time-step. However, if each time-step is computationally expensive (i.e. takes minutes to perform) a fairly expensive partitioner and significant data migration costs can be accommodated. Thus frequent partitioning do not automatically indicate a “cheap” partitioner - one needs to compare the partitioning and data migration costs with the time between successive repartitioning.

4 A New Model

This section derives a new continuous classification space and a model to characterize the "state" of a SAMR application. The section starts by investigating the fundamental requirements and suitable properties of such a classification space. Based on these requirements and properties, it proceeds with the designing of this space. Finally, it derives the methods for sampling application and system state and mapping this state onto the proposed classification space.

What the Partitioner Needs

This section discusses the fundamentals of grid hierarchy partitioning. Independent of partitioning approach, any serious partitioner should either (a) exhibit a strong preference for optimizing a specific metric, or (b) have parameters which allow one to bias its behavior so as to better optimize a given metric. Partitioners are generally confronted with the same problem-specific trade-offs and merely use different algorithms to achieve them. As a consequence, the classification space should be constructed to conform to these "universal truths" of SAMR partitioners - only then can the information present in application and system state be fully exploited.

A sophisticated partitioning tool offers various parameters for influencing the outcome. Most prominent is the trade-off between communication costs and load balance. For example, to focus on load balance in `Nature+Fable` we may choose a small *atomic unit*, select a large *Q*, choose *fractional blocking* and so forth. Working with other partitioners, we might migrate from domain-based techniques toward more elaborate patch-based techniques specializing in optimizing load balance.

Further, it is a fairly straight-forward idea to trade-off overall outcome quality for partitioning speed. For example, there are SAMR applications that compute for many minutes in between synchronization and re-griding. For these applications, it would make sense to spend more than fractions of a second to generate a more high-quality partitioning.

The third and last important trade-off occurs when there is need to optimize the amount of data migration. As opposed to the two trade-offs above, there is no unique or apparent trade-off to optimizing data migration. Depending on the current partitioning strategy and the state of the grid hierarchy, optimizing data migration may be obtained by e.g. invoking some kind of post mapping technique or switching methods to a more "diffusion-like" one, or investing more time in creating a more fully ordered SFC mapping. Depending on the circumstances, any of the metrics load imbalance, communication, speed or overall remaining quality might suffer. Note that the optimal amount of data migration is zero, translating to keeping all data where currently allocated. The trade-off for inducing longer waits between re-partitioning is of course the penalty of keeping the same partitioning during this time. Hence, attacking data migration this way trades-off whatever shortcomings the current partitioning is suffering from.

These are the three major and general trade-off possibilities, and it is imperative that *any* classification model (in the present context) should expose these fairly explicitly.

In view of the above, we propose that the partitioner-centric classification space should host exactly these three dimensions: (1) Communication versus load balance, (2) Speed versus overall quality, and (3) Data migration.

Illustrated in Figure 3 (right), this partitioner-centric classification space is obviously quite different from the octant approach. Moreover, we propose that the classification space is *absolute* and *continuous*, as opposed to *relative* and *discrete* in the ARMaDA framework. Consequently, a state sampling will generate a mapping onto a point defined in a continuous coordinate space within the classification space and the loci of all such points, as a simulation evolves, will be a curve in the same space. Thus, unlike ARMaDA, one does not discretely transition between octants, but rather follow a smooth curve. This enables not only a coarse grained partitioner selection, but also an extremely fine grained partitioner configuration.

The sub-sections below derive the theory for obtaining the necessary formulas for applying the model to any SAMR application.

4.1 Trade-off 1: Load Balance vs Communication

To best provide the partitioner with information quantifying the relative importance of achieving a good load balance versus reducing communication, we need a strategy independent of the partitioner currently in use. The information should preferably be based only on the status of the grid hierarchy (application) A and the system (computer) C.

To achieve this, we propose the following model where two extremes are used to span the classification space.

Assume there exists a way to estimate the load imbalance penalty $\beta_l(A, C)$ incurred for a domain decomposition obtained by invoking a simple, strictly domain-based partitioning strategy on the current hierarchy and system. We assume our simple technique will generate insignificant amount of (and only intra-level) communication. Further, assume there exists a way to estimate the communication penalty $\beta_c(A, C)$ incurred by a decomposition obtained by invoking a particular patch-based partitioning strategy (described later). We assume this strategy will generate insignificant load imbalance.

The patch-based technique referred to is the one where each patch is distributed evenly across all processors. This is known to generate perfect load balance, but a high amount of (intra and, most importantly, inter-level) communication.

The estimations constitute two extremes and provide information of the relative suitability of optimizing for load balance or communications. A comparison of $\beta_l(A, C)$ and $\beta_c(A, C)$ places us along the first axis in the cube. This model is not only independent of the current (or any) partitioning technique — it also incorporates the fundamentals of partitioning SAMR grid hierarchies.

We suggest the comparison of $\beta_l(A, C)$ and $\beta_c(A, C)$ should be biased with the system characteristics. These characteristics are obtained once and for all for the current machine by a simple benchmark program. We represent the system characteristics by a single number, which is the compute-to-communicate ratio. The specifics of this procedure is yet to be determined, but they should not present significant research challenges.

4.1.1 Estimating the load balance penalty

For a grid hierarchy with refinement levels 0 through $l_{\max}$ and grids $G_1^l, G_2^l, \ldots, G_{n_l}^l$ on level l, let W_t be the total amount of work, defined as

$$W_t = \sum_{l=0}^{l_{\max}} \sum_{i=1}^{n_l} |G_i^l| r^l, \tag{1}$$

where $|G|$ denotes the number of points (size) of the grid G and r is the refinement factor, which is assumed to be equal in time and space.

Let the number of processors be p and the unit work $U = W_t/p$ be the average (optimal) load assignment to any processor. Then let W_m be the work for the heaviest loaded processor. Load imbalance is normally expressed as W_m/U. The greater the load imbalance, the poorer the parallel efficiency. *One* way of measuring the parallel efficiency for a SAMR application as follows. Assume a set of p_o processors ($p_o << p$) are significantly overloaded so that their average workload is XU, where $X > 1$. This implies a total workload for these p_o processors of $W_o = XUp_o$. Now, the remaining processors $p_r = p - p_o$ have to share the rest of the work $W_r = W_t - W_o$. The parallel efficiency due to this load imbalance can be expressed as

$$\begin{aligned} E &= \frac{W_r}{p_r}\frac{p}{W_t} = \frac{W_t - W_o}{W_t}\frac{p}{p - p_o} \\ &= \left(1 - \frac{W_o}{W_t}\right)\left(1 - \frac{p_o}{p}\right)^{-1} \end{aligned} \tag{2}$$

which, given the assumptions, is a number between zero and one. To obtain the load balance penalty β_l such that $0 \leq \beta_l \leq 1$, we suggest E be evaluated via Equation 2 in key areas in the grid hierarchy and then define $\beta_l = 1 - E$.

Let the Cores in an arbitrary grid hierarchy be denoted by C^i for $i = 1, 2, \ldots, n_c$, and let the size of C^i on level 0 be $|C_0^i|$, the work for C^i be W_c^i as defined by Equation 1. Let the optimal number of processors for C^i be $p_{\text{opt}}^i = pW_c^i/W_t$ and the maximum number of processors that can fit in C^i be $p_{\text{max}}^i = |C_0^i|/a^D$, where a^D is the atomic unit in D dimensions. Finally, let $q^i = p_{\text{opt}}^i/p_{\text{max}}^i$.

We start by the case where $q^i \leq 1$ for all i. The ratio q^i indicates the "tightness" of the problem. As $q \to 1$, it gets harder to balance the load.

Let C^h be the Core with the largest q. From experience, we know that even for a single grid roughly half of the processors p_{opt}^h will be assigned a load of UX each, where $X \approx q^h + 1$. The load imbalance is due to round-off errors caused by integer divisions and sub optimal grid aspect ratios. Generally, the tighter the problem (greater q) the greater the imbalance. The efficiency for C^h is according to Equation 2

$$E^h = \frac{1 - q^h}{2}\left(\frac{1}{2}\right)^{-1} = 1 - q^h.$$

Since C^h is the hardest case of all Cores, it is safe to assume that E^h has the lowest efficiency of all C^i. Hence, in this case $\beta_l = 1 - E^h = q^h$.

As an example consider $p_{\text{opt}}^h = 4$ and $p_{\text{max}}^h = 40$, which gives $q^h = 0.1$. For this case, we estimate that half of the processors (2 in this case) will end up with a 10 percent overload.

Now we derive β_l for the the case where at least one $q^i > 1$. This means that there is at least one Core which cannot be partitioned into the optimal number of processors due to granularity constraints. Consequently, the processors that could not be assigned to these Cores have to be assigned elsewhere. It is the (too little) work and the (too great) number of processors for this "elsewhere" that we focus on.

Let $\bar{W}$ denote the sum of the work *not* in "elsewhere" and $\bar{p}$ the number of processors *not* assigned to "elsewhere". That is

$$\bar{W} = \sum_{\forall i\{q^i > 1\}} W_c^i$$

and

$$\bar{p} = \sum_{\forall i\{q^i > 1\}} p_{\text{max}}^i.$$

The efficiency for "elsewhere" is then according to Equation 2

$$E^e = \left(1 - \frac{\bar{W}}{W_t}\right)\left(1 - \frac{\bar{p}}{p}\right)^{-1},$$

and consequently, $\beta_l = 1 - E^e$.

In a real case scenario, β_l is trivial to compute according to the formulas above, by using the partitioning tool `Nature+Fable`. Most of the entities occurring in the formula above, will be computed as a part of the partitioning process.

4.1.2 Estimating the communication penalty

Let the communication penalty β_c be the fraction of the total data needed to be communicated as inter-level communication. Let W_{fc} be the fraction of workload residing in the Cores and let f be the fraction of work within the Cores avoiding inter-level communication. Since all inter-level communication will occur within the Cores, we get:

$$\beta_c = (1 - f)W_{fc}.$$

For the worst possible $\beta_c = \hat{\beta}_c$, we get:

$$\hat{\beta}_c \to 1.$$

To compute β_c in a practical case, we proceed as follows. For an arbitrary parent-child pair $P^i = (P_p^i, P_c^i)$ in $1D$, let $x(P^i)$ be the fraction of the child P_c^i covering the parent P_p^i, and let $f(x(P^i), p)$ be the fraction of the child avoiding inter-level communication. Then,

$$\beta_c = W_{fc} \sum_{\forall P^i} \frac{(1 - f(x(P^i), p)|P_c^i|}{W_c}$$

$$= \frac{1}{W_t} \sum_{\forall P^i} (1 - f(x(P^i), p)|P_c^i|. \quad (3)$$

Thus, we need to derive the function $f(x(P^i), p)$. Assume all grids G_i are distributed evenly in logical rectangles, called patches, over all p processors and assume a linear mapping onto processors identical for all grids. For a parent-child grid pair (P_p, P_c) ,the probability for a patch $a \in P_p$ and a patch $b \in P_c$ ending up on the same processor goes to zero when p goes to infinity. When P_c is relatively small compared to P_p we go faster to zero.

Assume 1D and let h_1 be the size of the a parent grid P_p and h_2 the size of its child grid P_c. Assume we place P_c so its left border align the border of P_p (not optimal) as in Figure 4. Let $x = x(P)$ denote the ratio h_1/h_2 (consequently $x \in [0, 1]$). If $x < 1/2$, the fraction of h_1 where inter-level communication is eliminated is x/p as in the two processor case in Figure 4 (left). If $1/2 \leq x < 2/3$, the fraction is $x/p + (2x-1)/p$ as in the two processor case in Figure 4 (right). For $2/3 \leq x < 3/4$, we get $x/p + (2x-1)/p + (3x-2)/p$. Continuing the same pattern, we add the term $(ix - (i-1))/p$ for $i = 1, 2, 3 \ldots, k$, where

$$k = \min\left(\lceil \frac{1}{1-x} \rceil - 1, p\right).$$

The first argument to min is the condition for $(ix - (i-1))/p \leq 0$ and the second argument (p) is needed to limit the number of terms in the sum by the number of processors when x approaches 1.

Then $f(x(P), p)$, the fraction of P_c avoiding inter-level communication, is

$$\frac{1}{p}\sum_{i=1}^{k}(ix - (i-1)) = \frac{1}{2p}\left(xk(k+1) - k(k-1)\right).$$

For 2D, assume P_p and P_c are squares with sides h_1 and h_2 respectively, and let (as for the 1D case) $x = h_1/h_2$. Further, to obtain the p rectangles, we assign $\sqrt{p}$ processors to each dimension. For notation purposes, let

$$g(x, k) = xk(k+1) - k(k-1)$$

The fraction avoiding inter-level communication is then

$$\left(\frac{g(x, k_2)}{2\sqrt{p}}\right)^2 = \frac{g(x, k_2)^2}{4p},$$

where

$$k_2 = \min\left(\lceil \frac{1}{1-x} \rceil - 1, \sqrt{p}\right).$$

For the D dimensional case, we get

$$\frac{g(x, k_D)^D}{p2^D},$$

where

$$k_D = \min\left(\lceil \frac{1}{1-x} \rceil - 1, p^{\frac{1}{D}}\right).$$

For a realistic case, we need to allow for non-square grids. Consequently, we define $y(P)$ and $z(P)$ analogously to $x(P)$. In 3 dimensions, we get

$$f\left(x(P), y(P), z(P), p\right) =$$

$$= \frac{g(x(P), k_3)g(y(P), k_3)g(z(P), k_3)}{p2^3}, \quad (4)$$

where

$$k_3 = \min\left(\lceil \frac{1}{1-x} \rceil - 1, p^{\frac{1}{3}}\right),$$

and analogously for $2D$.

Figure 5 shows the fraction of P_c avoiding inter-level communication for the aligned case in 1D. Examining this plot, we see that when $x = h_1/h_2$ is small, the fraction of P_c avoiding inter-level communication is insignificant even for small values on p. Even when x approach 1, the fraction goes to zero relatively fast. Since all grids are distributed over all processors, it is obvious that the amount of inter-level communication for a large number of processors will be immense.

The overall communication penalty β_c could be computed by Equation 3 with f defined by Equation 4. Note that this is a relatively inexpensive operation. The square (or cube) root of p is computed once at simulation startup. Then, we compute k and k^2, which are used in the computation of $g(x, k)$. Consequently, the number of arithmetical operations are small and could be applied to each parent/child grid.

4.1.3 Comparing β_l and β_c

A useful comparison of β_l and β_c can be made only after the system characteristics have been taken into account. In the following, we assume that this is done and our β_l^* and β_c^* are appropriately biased to reflect the underlying parallel computer.

Using β_l^* and β_c^* to place us along the first dimension in the classification space between zero and one necessitates reducing a 2-dimensional input to 1-dimensional output. The dimension disregarded in this process will be used later (discussed below).

For comparing β_l^* and β_c^*, we propose

$$\text{trade-off 1} = \begin{cases} 1 - \beta_l^*/(2\beta_c^*) & \text{if } \beta_c^* > \beta_l^* \\ \beta_c^*/(2\beta_l^*) & \text{otherwise.} \end{cases} \tag{5}$$

4.2 Trade-off 2: Speed vs. Overall Quality

The trade-off between speed and overall quality translates to a comparison of two entities. They are quantifications of 1) how much time the partitioner would like to spend to obtain its goals, and 2) what time-slot size the application realistically can offer it.

Since both the other trade-offs (1 and 3) are derived from two penalties each, they can both provide the present trade-off (speed versus quality) with important information regarding the quantification of 1). That is, by reducing the two-dimensional data to one dimension for the other trade-offs, there are “unused” data yet to exploit. For the case of trade-off 1, β_l and β_c are compared by Equation 5. This equation disregard the “amplitude” of the inputs. For example, $\beta_l = \beta_c = 0.1$ would yield the same result as $\beta_l = \beta_c = 0.4$. But this information is of utmost importance to the present trade-off.

As the penalties approach 1, the more prominent is the need to optimize. Consequently, a quantification of 1) could be obtained by taking the average of the other penalties. An average close to one translates to the greatest request for partitioning time, and an average close to zero would translate to a simple partitioning case with no particular demands on the partitioning (any will do).

The quantification of 2) is more problematic. We need to know how long the compute cycles are, i.e., how much time is spent computing between global synchronization points. Placing timing calls in the code would facilitate an easy way of determining how much time we could afford to spend in the partitioning process. However, timing calls are not an option, since we strive for optimized (non-profiled) code.

The numerical algorithm can give information about how expensive, or computational intense, a cycle is. However, it is unclear how the partitioner would extract this information. Most importantly, we also need to know the re-partitioning frequency (not all synchronization points calls for re-partitioning). Thus, we provide methods for quantifying the *importance* of the partitioning, i.e., how much time the partitioner would like to spend to obtain its goal, but only future research will provide us with methods for quantifying how much time the partitioner has at its disposal.

4.3 Trade-off 3: Data migration

Trade-off 3 concerns *how much*, i.e., the amount of perturbation of the grid hierarchy since the most recent re-partitioning.

Keeping a history of grid hierarchies as a sliding window as proposed by Sumir Chandra in ARMaDA allows for the comparison of adaptation patterns at different time steps. By intersecting the boxes in the hierarchy at time-step t with those at time-step $t + 1$, we get a metric indicating how much the grid has changed during this time-step. A large intersection means little change, and a small intersection means a large change. Keeping a history of n states prevents thrashing and over-reacting to temporary changes. This dimension in the classification space should thus be relatively easy to determine.

5 Validation

This section explains the experimental process for validating the proposed model, and presents the results.

5.1 Methods — Experimental Setup

A trace file (described below) from each of the five SAMR applications (described below) is used in two different ways. First, the trace-file is processed by a program implementing our proposed model. This program outputs β_l and β_c for each time-step. Second, the trace-file is partitioned by `Nature+Fable` and processed by the SAMR simulator (described below). This program outputs the actual partitioning result in terms of load imbalance and communication amount for each time step. The two sets of output, viz. β_l vs actual load imbalance, and β_c vs actual communication amount, are then for each application plotted in the same figure to enable visual comparison. The idea is *not* that the plots should coincide — rather, the idea is to examine whether the model (i.e., β_c and β_l) succeeds in capturing the overall *behavior* of the different applications i.e. for a static and non-optimized partitioning setup, will the analytical model correctly capture the difficult-to-load-balance and difficult-to-reduce communications configurations of the grid hierarchy.

5.1.1 Five SAMR Applications

A suite of 5 “real-world” SAMR application kernels taken from varied scientific and engineering domains are used to evaluate the effectiveness of the proposed model to capture application behavior. These applications demonstrate different runtime behavior and adaptation patterns. Application domains include numerical relativity (Scalarwave), oil reservoir simulations (Buckley-Leverette), and computational fluid dynamics (compressible turbulence - RM, and supersonic flows - EnoAMR 2D). Finally, we also use TportAMR 2D which is a simple benchmark kernel that solves the transport equation in 2D and is part of the GrACE distribution. The applications use 5 levels of factor 2 refinements in space and time. Regridding and redistribution is performed every 4 time-steps on each level. The applications are executed for 100 time-steps and the granularity (minimum block dimension) is 2. The application kernels are described below.

The numerical relativity application (Scalarwave/SC) is a coupled set of partial differential equations. The equations can be divided into two classes: elliptic (Laplace equation-like) constraint equations which must be satisfied at each time, and coupled hyperbolic (Wave equation-like) equations describing time evolution. This kernel addresses the hyperbolic equations and is part of the Cactus numerical relativity toolkit[2].

The Buckley-Leverette model is used in Oil-Water Flow Simulation (OWFS) application for simulation of hydrocarbon pollution in aquifers. OWFS provides for layer-by-layer modeling of oil-water mixture in confined aquifers with regard to discharge/recharge, infiltration, interaction with surface water bodies and drainage systems, discharge into springs and leakage between layers. This kernel is taken from the IPARS reservoir simulation toolkit developed at the Center for Subsurface Modeling at the University of Texas at Austin[3].

The RM is a compressible turbulence application solving the Richtmyer-Meshkov instability. This application is part of the virtual test facility (VTF) developed at the ASCI/ASAP center at the California Institute of Technology[4]. The Richtmyer-Meshkov instability is a fingering instability which occurs at a material interface accelerated by a shock wave. This instability plays an important role in studies of supernova and inertial confinement fusion.

EnoAMR is a computational fluid dynamics application that addresses the forward facing step problem, describing what happens when a step is instantaneously risen in a supersonic flow. The application has several features including bow shock, Mach stem, contact discontinuity, and a numerical boundary. EnoAMR is also a part of the virtual test facility developed at the ASCI/ASAP Center at Caltech.

5.1.2 Partitioning Set-Up

All partitioning is done with `Nature+Fable` set-up with static "default" values [35, 37]. The goal is not to obtain a particularly good-quality partitioning, but rather to partition the applications with a static "neutral" setting so that behavior patterns in the applications are clearly visible.

5.1.3 Deriving Application Behavior

The derivation of load imbalance and communication amount is performed using software [33] developed at Rutgers University in New Jersey by The Applied Software Systems Laboratory, that simulates the execution of the Berger-Colella SAMR algorithm. This software is driven by an application execution trace obtained from a single processor run. This trace captures the state of the SAMR grid hierarchy for the application at the regrid (refinement and coarsening) step and is independent of any partitioning. The experimental process allows the user to select the partitioner to be used, the partitioning parameters (e.g. block size), and the number of processors. The trace is then run and the performance of the partitioning configuration at each regrid step is computed using a metric [35] with the components load balance, communication, data migration, and overheads.

Using the evaluation process described above, *communication* is the sum of the amount of inter-processor communication for each time-step. *Load imbalance* is the load of the heaviest loaded processor divided by optimal load for each time step.

5.1.4 Model Evaluation

To evaluate the ability of our penalties β_l and β_c to accurately capture the behavior of the applications, we plotted each application's load imbalance as a function of time and super imposed β_l, scaled so as to best allow for visual comparison. The same was done for communication versus β_c. No numerical results, e.g. in terms of error norms, were derived. The purpose of this experimental process was to examine whether our model indeed reflects the inherent and dynamic optimization-need in the applications. This was most easily examined visually.

5.2 Results

Figure 6 through 10 display the results. Examining the plots, it seems that the proposed model generally captures the essence of application behavior i.e., a larger β_l generally corresponds to a greater load imbalance and a larger β_c generally corresponds to larger communication amount. The trends are similar, and in case of oscillatory behavior, the model captures the time period of the oscillation. Note that the values obtained from the simulations are governed by the domain decomposition achieved by the actual (hybrid) partitioner used, while the model predictions assume a simple purely domain-based or purely patch-based partitioner.

Below, we discuss the results for each application.

RM2D The load imbalance was relatively static for this application. The penalty β_l successfully captured this by displaying a smooth curve increasing occasionally to indicate a general increase in load imbalance. The communication volume changed seemingly randomly and β_c followed the pattern both on the small and large scale fairly well.

BL2D Both the load imbalance and the communication volume exhibited oscillatory behavior for this application. Both β_l and β_c followed the time periods and accurately showed the same "peaks" and "valleys".

ENO2D This was the most static of the applications, exhibiting only a few transitions. We were unsure whether this was due to a bug in the application that generated this particular trace. The penalty β_l captured the load balance behavior fairly well, but it failed to accurately reflect the last small transition at time step 40. The penalty β_c followed the pattern of the communication volume fairly well.

[2]Cactus Computation Toolkit - http://www.cactuscode.org

[3]IPARS: A New Generation Framework for Petroleum Reservoir Simulation - http://www.ticam.utexas.edu/CSM/ACTI/ipars.html

[4]Center for Simulation of Dynamic Response of Materials - http://www.cacr.caltech.edu/ASAP/

SC2D This application exhibited oscillatory behavior both in load imbalance and communication volume. Both β_l and β_c followed the time periods and accurately showed the same "peaks" and "valleys". The penalty β_l did not accurately capture the behavior of the load imbalance at a larger scale.

TP2D This application exhibited seemingly random load imbalance and communication volume dynamics. The penalty β_l captured only some of the features of the load imbalance on a small and a large scale. The penalty β_c reflected large scale and small scale behavior of communication fairly accurately.

6 Conclusions and Future Work

We have developed a model that, *ab initio*, predicts the suitability of a structured adaptive mesh for purely domain-based or patch-based decomposition. This information is used determine one of the parameters used to configure a partitioner or choose the optimal one for a given problem. The predictions were validated against data obtained from five different SAMR simulations.

From the results we draw the conclusion that our model could be useful for decreasing execution time for large SAMR applications. Most such applications exhibit a highly dynamic behavior and consequently the partitioning requirements change at run-time. To accurately track and adapt for this dynamic behavior potentially lead to a large decrease of execution times.

In the future, we will address the remaining two parameters that complete the set which uniquely determine the optimal partitioner (setting). These parameters were defined in this paper, but not validated. Once done, this model will form the core of the *meta-partitioner*, an expert system that chooses (or configures) partitioners for optimality.

Acknowledgments

The authors thank Manish Parashar and Sumir Chandra at the Center for Advanced Information Processing, Rutgers University, NJ, USA, and Michael Thuné and Jarmo Rantakokko at Information Technology, Uppsala University, Sweden for scientific collaboration. 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.

References

[1] The ASCI allience. http://www.llnl.gov/asci-alliences/asci-chicago.html, University of Chicago, 2000.

[2] The ASCI/ASAP center. http://www.carc.caltech.edu/ASAP, California Institute of Technology, 2000.

[3] CHOMBO. http://seesar.lbl.gov/anag/chombo/, NERSC, ANAG of Lawrence Berkeley National Lab, CA, USA, 2003.

[4] Scott B. Baden, Scott R. Kohn, and S. Fink. Programming with LPARX. Technical Report, University of California, San Diego, 1994.

[5] Dinshaw Balsara and Charles Norton. Highly parallel structured adaptive mesh refinement using language-based approaches. *Journal of parallel computing*, (27):37–70, 2001.

[6] M. Berger, et al. Adaptive mesh refinement for 1-dimensional gas dynamics. *Scientific Computing*, 17:43–47, 1983.

[7] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82, 1989.

[8] Marsha J. Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Jounal of Computational Physics*, 53:484–512, 1984.

[9] G. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, pages 46–53, 1999.

[10] S. Chandra and M. Parashar. An evaluation of partitioners for parallel SAMR applications. *Lecture Notes in Computer Science*, 2150:171–174, 2001. Euro-Par 2001.

[11] S. Chandra, J. Steensland, and M. Parashar. An experimental study of adaptive application sensitive partitioning strategies for SAMR appliations, 2001. Research poster presentation at Supercomputing Conference, November 2001.

[12] S. Chandra, J. Steensland, M. Parashar, and J. Cummings. An experimental study of adaptive application sensitive partitioning strategies for SAMR appliations. Santa Fe, NM, USA, 2001.

[13] Sumir Chandra. ARMaDA: a framework for adaptive application-sensitive runtime management of dynamic applications. Master's Thesis, Graduate School, Rutgers University, NJ, USA, 2002.

[14] Mattew W. Choptuik. Experiences with an adaptive mesh refinement algorithm in numerical relativity. *Frontiers in Numerical Relativity*, pages 206–221, 1989.

[15] Karen Devine et al. Design of dynamic load-balancing tools for parallel applications. Technical report, Sandia national Laboratories, Albuquerque, NM, USA, 2000.

[16] Stephen J. Fink, Scott B. Baden, and Scott R. Kohn. Flexible communication mechanisms for dynamic structured applications. In *Proceedings of IRREGULAR '96*, 1996.

[17] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998.

[18] Scott Kohn. SAMRAI homepage, structured adaptive mesh refinement applications infrastructure. http://www.llnl.gov/CASC/SAMRAI/, 1999.

[19] Z. Lan, V. Taylor, and G. Bryan. Dynamic load balancing for structured adaptive mesh refinement applications. In *Proceedings of ICPP 2001*, 2001.

[20] Z. Lan, V. Taylor, and G. Bryan. Dynamic load balancing of SAMR applications on distributed systems. In *Proceedings of Supercomputing 2001*, 2001.

[21] Peter MacNeice. Paramesh homepage, 1999. sdcd.gsfc.nasa.gov/ESS/macneice/paramesh/-paramesh.html.

[22] Peter MacNeice et al. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer physics communications*, (126):330–354, 2000.

[23] M. Norman and G. Bryan. Cosmological adaptive mesh refinement. *Numerical Astrophysics*, 1999.

[24] M. Parashar and J. C. Browne. On partitioning dynamic adaptive grid hierarchies. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, 1996.

[25] M. Parashar, J.A. Wheeler, G. Pope, K.Wang, and P. Wang. A new generation EOS compositional reservoir simulator: Part II - framework and multiprocessing. *Proceedings of the Society of Pertroleum Engineerings Reservoir Simulation Symposium, Dallas, TX*, June 1997.

[26] Manish Parashar and James Browne. System engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement. *IMA Volume on Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, pages 1–18, 2000.

[27] S.G. Parker. A component-based architecture for parallel multiphysics PDE simulations. In *Proceedings of ICCS 2002*, number 2331, pages 719–734. Springer Verlag, 2002.

[28] R. Pember, J. Bell, P. Colella, W. Crutchfield, and M. Welcome. Adaptive cartesian grid methods for representing geometry in inviscid compressible flow, 1993. *11th AIAA Computational Fluid Dynamics Conference*, Orlando, FL, July 6-9.

[29] Jarmo Rantakokko. *Data Partitioning Methods and Parallel Block-Oriented PDE Solvers*. PhD thesis, Uppsala University, 1998.

[30] Hawley S. and Choptuic M. Boson stars driven to the brink of black hole formation. *Physic Rev*, D 62:104024, 2000.

[31] Jeffrey Saltzman. Patched based methods for adaptive mesh refinement solutions of partial differential equations, 1997. Lecture notes.

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

[33] 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.

[34] Johan Steensland. Domain-based partitioning for parallel SAMR applications, 2001. Licentiate thesis. Uppsala University, IT, Dept. of scientific computing. 2001-002.

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

[36] 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.

[37] Johan Steensland and Jaideep Ray. A heuristic re-mapping algorithm reducing inter-level communication in SAMR applications. Technical Report SAND2003-8310, Sandia National Laboratory, Livermore, CA, USA, 2003.

[38] Johan Steensland, Stefan Söderberg, and Michael Thuné. A comparison of partitioning schemes for blockwise parallel SAMR applications. In *Proceedings of PARA2000, Workshop on Applied Parallel Computing*, volume 1947 of *LNCS*, pages 160–169. Springer Verlag, 2001.

[39] Erlendur Steinthorsson and David Modiano. Advanced methodology for simulation of complex flows using structured grid systems. *ICOMP*, 28, 1995.

[40] M. Thuné. Partitioning strategies for composite grids. *Parallel Algorithms and Applications*, 11:325–348, 1997.

[41] N. Touheed, P. Selwood, P. Jimack, and M. Berzins. A comparison of some dynamic load-balancing algorithms for a parallel adaptive flow solver. *Journal of Parallel Computing*, 26:1535–1554, 2000.

[42] C. Walshaw and M. Cross. Multilevel mesh partitioning for heterogeneous communication networks. *Future generation computer systems*, 17:601–623, 2001.

[43] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, December 1997.

[44] P. Wang, I. Yotov, T. Arbogast, C. Dawson, M. Parashar, and K. Sepehrnoori. A new generation EOS compositional reservoir simulator: Part I - formulation and discretization. *Proceedings of the Society of Pertroleum Engineerings Reservoir Simulation Symposium, Dallas, TX*, June 1997.

[45] Andrew M. Wissink et al. Large scale parallel sctructured AMR calculations using the SAMRAI framwork. *In proceedings of Supercomputing 2001*, 2001.

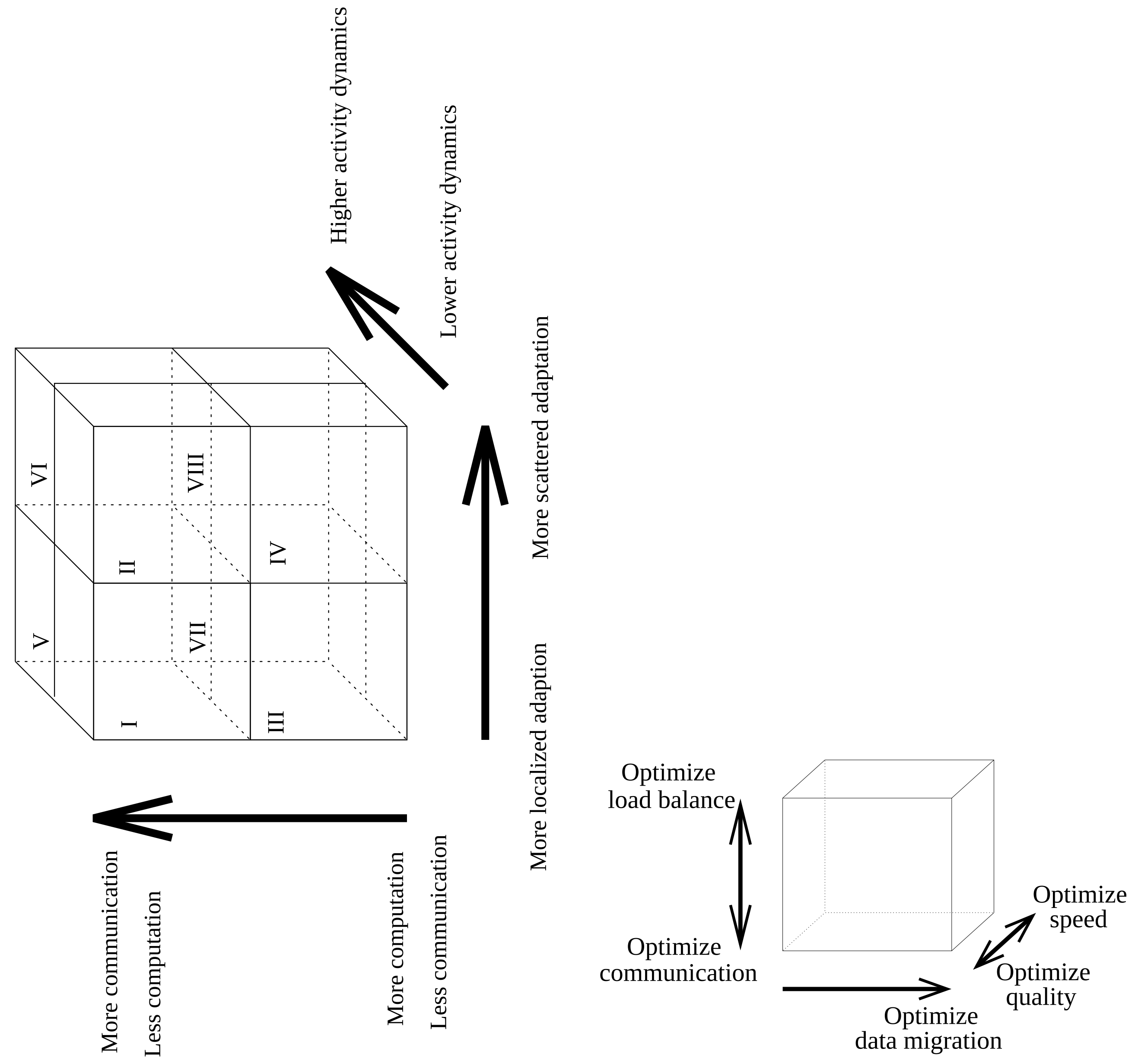


Figure 3: Left: The octant approach. The application and system is classified with respect to (1) communication / computation domination, (2) scattered / localized refinements, and (3) activity dynamics. Right: The absolute and continuous partitioner-centric classification space. Note the absence of unique trade-off for data migration, as opposed to the general trade-offs communication vs load balance and speed vs quality.

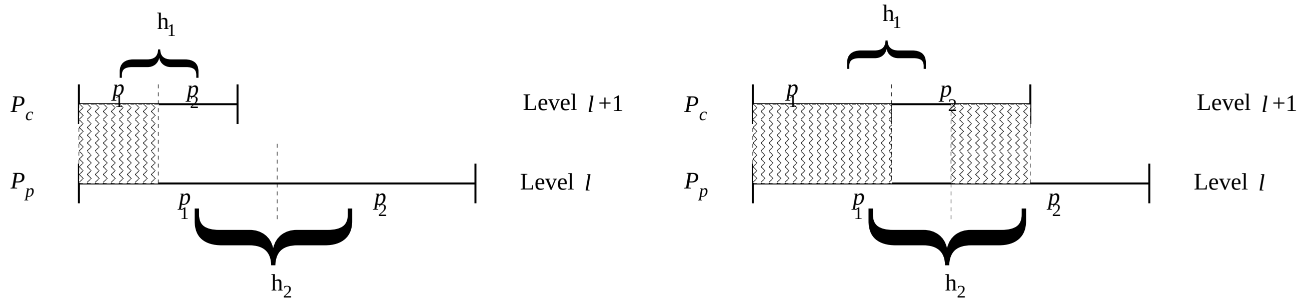


Figure 4: A part of a 1D grid hierarchy with an aligned parent-child grid pair (P_p, P_c) partitioned into 2 partitions ($p = 2$). Note (left) the fraction h_1/p of P_c avoiding inter-level communication, and (right) a larger P_c leading to a bigger fraction $(h_1/p + (2h_1 - h_2)/p)$ of P_c avoiding inter-level communication.

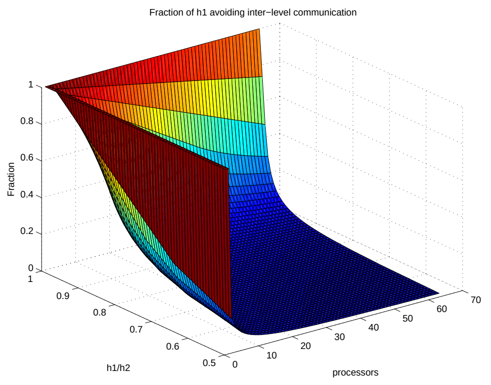


Figure 5: The fraction for $0.5 \leq x \leq 1$ and $0 < p < 64$ of P_c that in the aligned 1D case will eliminate inter-level communication. Note steepness for small x.

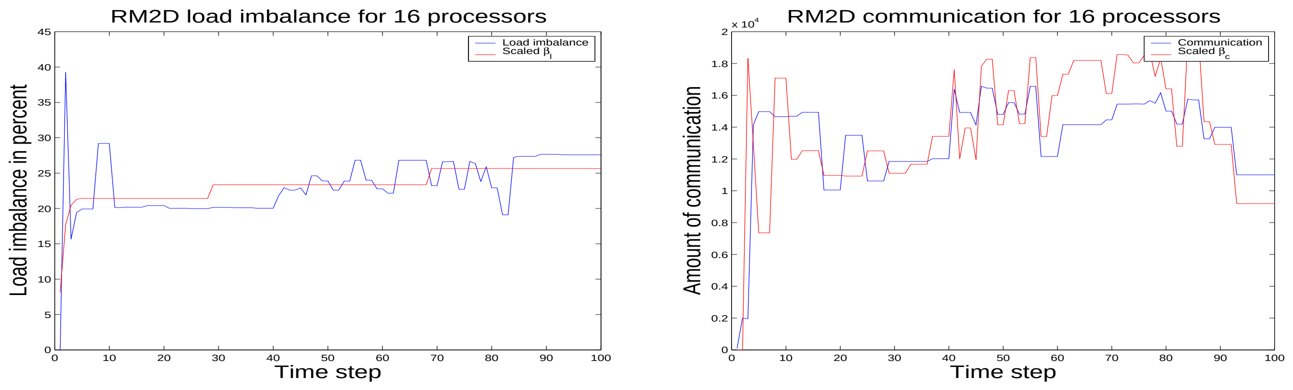


Figure 6: The ability of the penalties to predict application behavior for RM2D. Left, the actual load imbalance (in blue) and the penalty β_l (in red). Right, the actual communication volume (in blue) and the penalty β_c (in red).

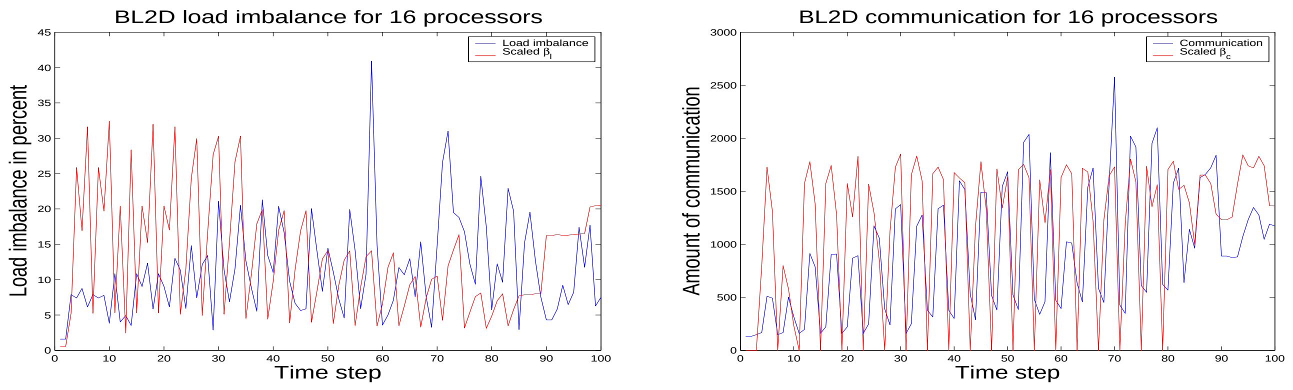


Figure 7: The ability of the penalties to predict application behavior for BL2D. Left, the actual load imbalance (in blue) and the penalty β_l (in red). Right, the actual communication volume (in blue) and the penalty β_c (in red).

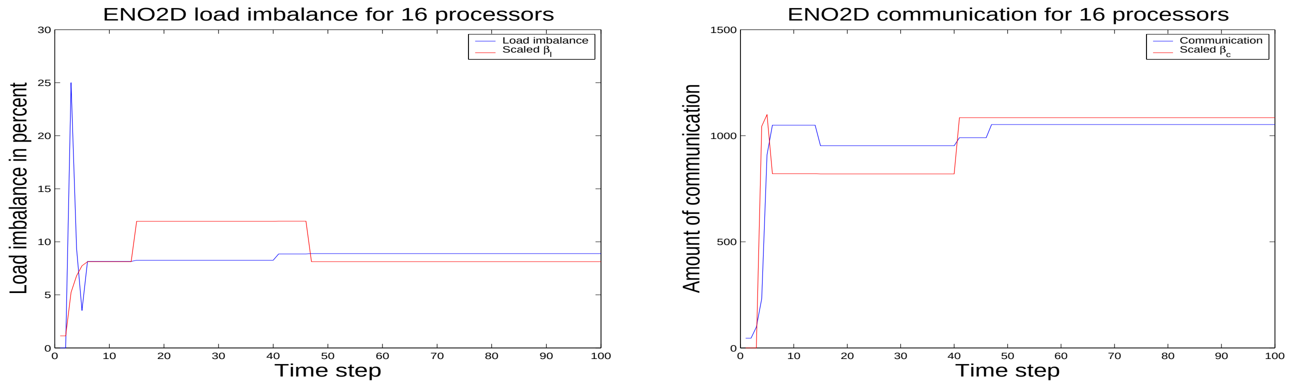


Figure 8: The ability of the penalties to predict application behavior for ENO2D. Left, the actual load imbalance (in blue) and the penalty β_l (in red). Right, the actual communication volume (in blue) and the penalty β_c (in red).

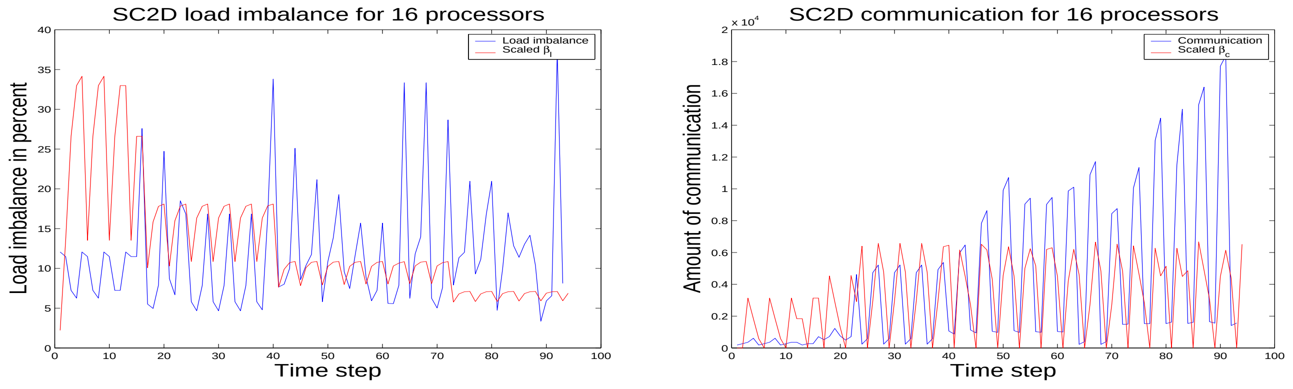
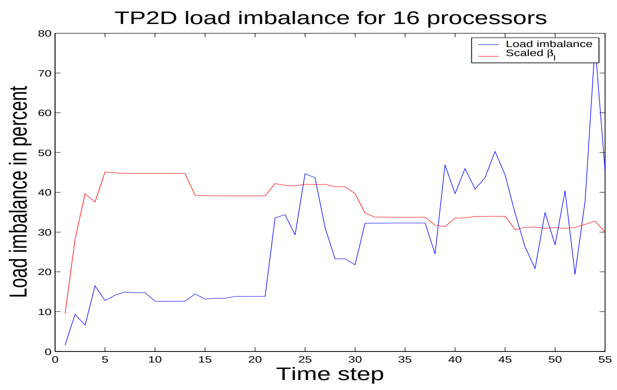


Figure 9: The ability of the penalties to predict application behavior for SC2D. Left, the actual load imbalance (in blue) and the penalty β_l (in red). Right, the actual communication volume (in blue) and the penalty β_c (in red).



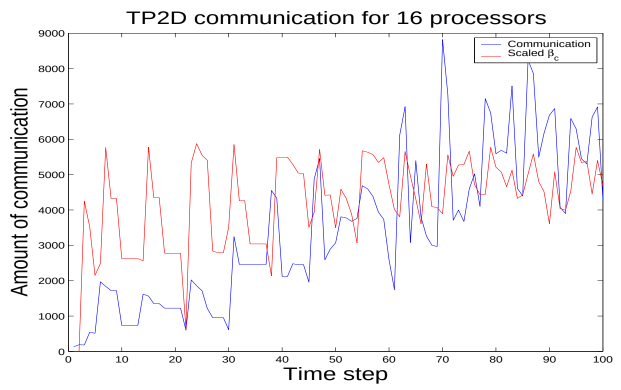


Figure 10: The ability of the penalties to predict application behavior for TP2D. Left, the actual load imbalance (in blue) and the penalty β_l (in red). Right, the actual communication volume (in blue) and the penalty β_c (in red). Note only 55 time-steps displayed for the load imbalance due to inability of `Nature+Fable` to cope with the remaining time-steps.