SANDIA REPORT
SAND2003-8310
Unlimited Release
Printed July 2003

A Heuristic Re-Mapping Algorithm Reducing Inter-Level Communication in SAMR Applications

J. Steensland and J. Ray

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-AC04-94AL85000.

Sandia National Laboratories

SAND2003-8310
Unlimited Release
Printed July 2003

A Heuristic Re-Mapping Algorithm Reducing Inter-Level Communication in SAMR Applications

Johan Steensland and Jaideep Ray
High Performance Computing and Networking
Sandia National Laboratory
P.O. Box 969
Livermore, CA 94550-9915, USA

Abstract

This paper aims at decreasing execution time for large-scale structured adaptive mesh refinement (SAMR) applications by proposing a new heuristic re-mapping algorithm and experimentally showing its effectiveness in reducing inter-level communication. Tests were done for five different SAMR applications. The overall goal is to engineer a dynamically adaptive meta-partitioner capable of selecting and configuring the most appropriate partitioning strategy at run-time based on current system and application state. Such a meta-partitioner can significantly reduce execution times for general SAMR applications.

Computer simulations of physical phenomena are becoming increasingly popular as they constitute an important complement to real-life testing. In many cases, such simulations are based on solving partial differential equations by numerical methods. Adaptive methods are crucial to efficiently utilize computer resources such as memory and CPU. But even with adaption, the simulations are computationally demanding and yield huge data sets. Thus parallelization and the efficient partitioning of data become issues of utmost importance. Adaption causes the workload to change dynamically, calling for *dynamic* (re-) partitioning to maintain efficient resource utilization.

The proposed heuristic algorithm reduced inter-level communication substantially. Since the complexity of the proposed algorithm is low, this decrease comes at a relatively low cost. As a consequence, we draw the conclusion that the proposed re-mapping algorithm would be useful to lower overall execution times for many large SAMR applications. Due to its usefulness and its parameterization, the proposed algorithm would constitute a natural and important component of the meta-partitioner.

Acknowledgments

The authors wish to thank Manish Parashar and Sumir Chandra at the Center for Advanced Information Processing, Rutgers University, NJ, USA for scientific collaboration and Michael Thuné and Jarmo Rantakokko at Information Technology, Uppsala University, Sweden for comments on this manuscript leading to an improved paper.

Contents

This page intentionally left blank

1 Introduction

This paper presents a new heuristic re-mapping algorithm for the built-in partitioning techniques in `Nature+Fable` [35], and experimentally shows the effectiveness of this algorithm in reducing inter-level communication for five other vastly different structured adaptive mesh (SAMR) applications. Particular attention will be paid to a SAMR application that simulates the Richtmyer-Meshkov instabilities using an explicit time-stepping and a finite volume scheme.

The presented work is part of an ongoing research project [34, 35, 36, 10] 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) [37] 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 effectively 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 [42, 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. These implementations lead to interesting challenges in dynamic resource allocation, data-distribution, load-balancing, 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 other scenarios, the focus must be on lowering the interprocessor communication costs. Hence, a means to explicitly attack the amount of inter-level communication is crucial to the adaptive meta-partitioner.

The little support for tuning and choosing trade-off impacts growing more present 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, 43, 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. A 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 move towards the meta-partitioner by introducing a key component; a means for lowering the inter-level communication costs for SAMR applications. The key contributions are (1) a mathematical estimation of the compute and communication costs of SAMR simulations based on the common "time refinement" technique, (2) a fine-scale case-study of the Richtmyer-Meshkov instability, breaking down the communication amount into components, (3) a new, parameterized heuristic re-mapping algorithm that attempts to solve the problem posed above and is designed to operate as a component of the meta-partitioner, and (4) an experimental evaluation of this algorithm showing its effectiveness to reduce inter-level communication on five vastly different SAMR applications.

The rest of the paper is organized as follows. Section 2 introduces SAMR and presents related work on partitioning grid hierarchies. Section 3 motivates a means to explicitly attack inter-level communication and proposes the heuristic re-mapping algorithm. Section 4 explains the experimental methods, Section 5 displays the results, and Section 6 provides a discussion.

2 SAMR and Related Work

2.1 Introduction to SAMR

The numerical solution to a PDE is obtained by first discretizing the problem domain. One approach is to introduce a structured uniform Cartesian grid. The unknowns of the PDE are then approximated numerically at each discrete grid point. The resolution of the grid (or grid spacing) determines the local and global error of this approximation, and is typically dictated by the solution-features that need to be resolved. The resolution also determines computational costs and storage requirements. Dynamically adaptive solution techniques for PDE's provide a means for concentrating additional grid resolution and computational resources to regions in the application domain with large error. These techniques potentially lead to more efficient and cost-effective solutions to time-dependent problems exhibiting localized features, viz. shocks, discontinuities, or steep gradients.

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. The adaptive grid hierarchy corresponding to the SAMR formulation by Berger and Oliger [8] is shown in Fig. 1. A selection of snap-shots of the adaptive grid hierarchy for the Richtmyer-Meshkov 3D SAMR application is shown in Fig. 2.

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, 43] 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.

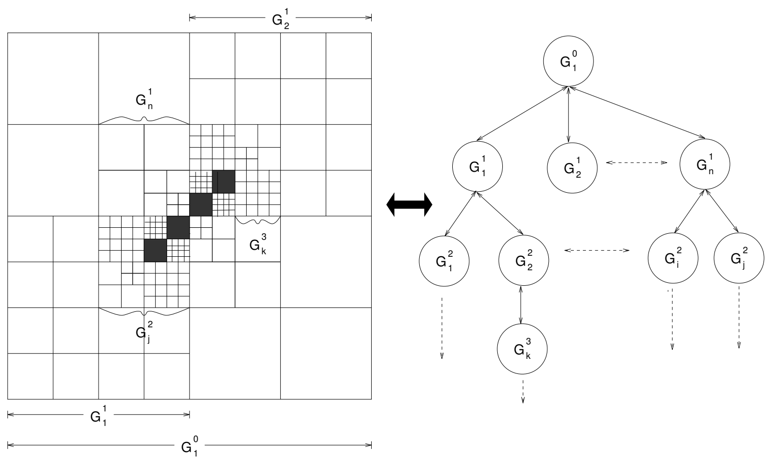


Figure 1: Berger-Oliger formulation of adaptive grid hierarchies for SAMR applications. Reprinted with permission from M. Parashar.

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 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, 43] (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, 38, 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, 38, 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 **+ Fra**ctional blocking and **bi**-**le**vel partitioning) [35] is aimed 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`, depicted in Fig. 3, is a step towards a complete implementation. It consists of the following units:

1. *A pre-partitioning unit* `Nature`, which divides the domain into *natural regions* suitable for attack by expert algorithms.

[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 [39, 41].

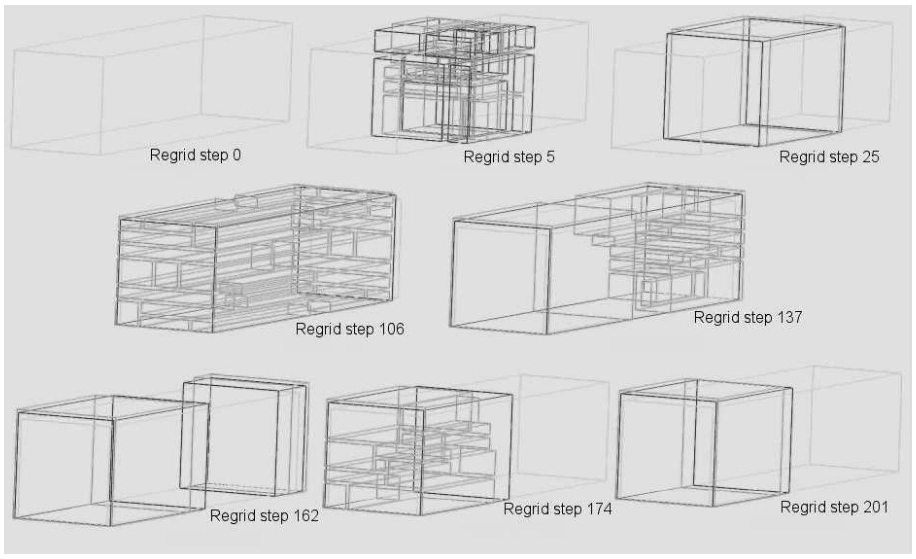


Figure 2: A sequence of dynamic adaptive grid hierarchies for a 3-D Richtmyer-Meshkov SAMR simulation. Note the dynamics of the grid hierarchy as the application evolves. Reprinted with permission from M. Parashar.

2. *A set of expert algorithms for blocking homogeneous and unrefined parts of a grid*, which is implemented as one part of `Fable`.

3. *An dynamic expert algorithm for partitioning complex and multi-level refined portions of a grid*, which is implemented as the other part of `Fable`.

4. *An Organizer*, which functions as the heart of, and interface to, `Nature+Fable`.

`Nature+Fable` separates homogeneous, unrefined (Hue) and complex, refined (Core) domains of the grid hierarchy and clusters refinement levels into *bi-levels*. The expert blocking algorithms used for the Hues are then after a coarse partitioning step re-used for the Cores.

With the components listed above, `Nature+Fable` is a flexible tool with ability to operate as a part of the meta-partitioner. `Nature+Fable` is the result of our research about the partitioning requirements of large parallel SAMR applications. Hence, it is designed to cope with the scenarios known to cause problems for previous approaches. The aim is that `Nature+Fable` will provide good scalability for general large-scale SAMR applications with deep hierarchies executed on general parallel computers. Moreover,the generality and adaption ability of `Nature+Fable` would make it suitable for applications in distributed computing, where resources are heterogeneous and even more dynamic.

3 A Heuristic Algorithm for Reducing Inter-Level Communication

The bi-level partitioning approach is appealing, since [35] (1) it has a strong rationale, and (2) it exhibits promising experimental results. We start this section by constructing a mathematical model of the compute and communication costs of a SAMR application and motivating the need to redistribute patches as bi-levels because of inter-level communication costs. This is then highlighted by a case-study of the Richtmeyer-Meshkow instability. The case-study establishes that even with the bi-level partitioning approach, there is need to explicitly attack inter-level communication costs. This section then outlines the inherent complexity in all re-mapping techniques and concludes by presenting our heuristic re-mapping algorithm.

3.1 Analytical Model

This section develops an analytical model for the compute and communication costs of a SAMR application and uses this model to motivate the need to partition

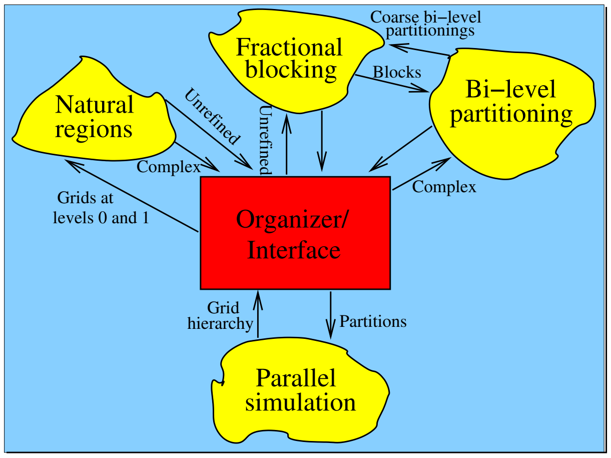


Figure 3: `Nature+Fable`, a partitioning tool implementing the greater part of the involved components of a meta-partitioner, with the goal to provide good scalability for general SAMR applications with deep, complex grid hierarchies executing on general parallel computers.

and redistribute a grid hierarchy as bi-levels (rather than individual patches).

Compute and communication loads in SAMR applications are tightly coupled with the numerical simulation algorithm. Most common algorithms are based on the "time refinement" approach [7] and consist of a series of identical processes called "time-steps" ; each time-step usually ends with a global reduction or some other operation that effectively synchronizes the processors. Within a time-step, patches are subjected to various numerical operations. If patches are refined by a factor of R, these numerical operations are done R times. Further, these operations proceed from the coarsest mesh to the finest in a recursive manner i.e., $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_2, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_2$ for a 3 level grid hierarchy with $R = 2$. $\mathcal{G}_l$ refers to the set of all patches on level l. Communications (ghost cell updates) are done after each of these numerical operations and follow the same pattern. At the culmination of the processing of each level, data is interpolated from the finer children patches onto the coarser parents.

Consider a 2D domain distributed over a number of processors after being subjected to a purely domain-based decomposition. Let a sub-domain on a processor have N_0 grid points on the coarsest level. Let a fraction of this coarse level be refined into level 1 patches, contained in the set $\mathcal{G}_1$. Thus, the number of grid points in $\mathcal{G}_1$, (i.e., the load associated with $\mathcal{G}_1$) is $\mathcal{L}_1 = \sum_{i=0}^{M_1-1} \alpha_i^1 N_0 R$ where M_l is the number of patches in $\mathcal{G}_l$ and α_i^l is the fraction of the sub-domain that exists in patch i on level l. Patches on a level are indexed from 0 to $M_l - 1$. Let $\mathcal{G}_1$ be recursively refined into $\mathcal{G}_2, \mathcal{G}_3, \ldots$.

Let L denote the index of the finest level. During a time-step, the compute load T_{comp} is

$$\begin{aligned} T_{comp} &= t_{comp}(\mathcal{L}_0 + R\mathcal{L}_1 + R^2\mathcal{L}_2 + \ldots) \quad (1) \\ &= t_{comp} \sum_{l=0}^{L} R^l \sum_{i=0}^{M_l-1} \alpha_i^l N_0 R^l \\ &= t_{comp} N_0 \sum_{l=0}^{L} R^{2l} \sum_{i=0}^{M_l-1} \alpha_i^l \end{aligned}$$

where t_{comp} is the computation time / load of a unit operation.

Let us assume that patches are roughly square and that the number of "ghost cells" or "guard cells" per patch $\propto \sqrt{\alpha_i^l N_0 R^l}$. Since the *intra-level* communication follows a similar pattern as the computation,

$$T_{comm} = t_{comm} \sum_{l=0}^{L} \sum_{i=0}^{M_l-1} 4\beta_i^l \sqrt{\alpha_i^l N_0 R^l} R^l \qquad (2)$$

where t_{comm} is a unit communication time and β_i^l is the fraction of the perimeter of patch $G_{l,i}$ that abuts another patch on the same level but in a different sub-domain (and thus requires communication time to get updated). The fraction of patch a $G_{l,i}$ that abuts another patch on the same level i but on *the same pro-*

cessor incur the cost of memory copies. This cost is relatively low and is thus ignored.

At a given level, data is interpolated from children patches to the parent at the end of each numerical operation. If t_{interp} is the unit interpolation cost, then the total time spent in interpolation, T_{interp}, is

$$\begin{aligned} T_{interp} &= t_{interp} \sum_{l=0}^{L-1} R^l \sum_{i=1}^{M_{l+1}-1} \alpha_i^{l+1} N_0 R^{l+1} \quad (3) \\ &= N_0 t_{interp} \sum_{l=0}^{L-1} R^{2l+1} \sum_{i=0}^{M_{l+1}-1} \alpha_i^{l+1}. \end{aligned}$$

Note that T_{interp} is entirely computational in a purely domain-based partitioning — since the child and parent reside on the same processor, these interpolated values do not have to be transported over a network and consequently they do not incur a communication cost.

Thus, the total time for an arbitrary processor to execute a time-step, T_{total}, is

$$T_{total} = T_{comp} + T_{interp} + T_{comm} + t_{wait} \quad (4)$$

where t_{wait} is the wait induced by load-imbalance amongst processors. In the following, we assume that the amount of intra-level communication is relatively small and could hence be neglected. We will now attempt to reduce t_{wait} by redistributing patches.

3.1.1 Strategy 1: A Patch-based Approach

In *Strategy 1* we move an arbitrary patch $G_{l,i}$ from a processor with $t_{wait} = 0$ to another with maximum t_{wait} i.e., the least loaded one. We ignore the migration cost, since this is incurred once and focus on the effect on performance.

We gain savings on compute τ^i_{comp} and interpolation τ^i_{interp} loads for the patch i. At the same time, we introduce the cost of *inter-level* communication, i.e., the cost of bringing back the interpolated data from the off-processor patch: $\alpha_i^l N_0 R^{2l-1} t_{comm}$.

Let γ denote the fraction of this communication time that could not be overlapped with computation. Then, the load change on the sending processor Δt_{send} is

$$\begin{aligned} \Delta t_{send} &= -\tau^i_{comp} - \tau^i_{interp} + \gamma \tau^i_{comm} \quad (5) \\ &= -\alpha_i^l N_0 R^{2l} t_{comp} - \alpha_i^l N_0 R^{2l+1} t_{interp} \\ &\quad + \alpha_i^l N_0 R^{2l-1} \gamma t_{comm} \\ &= \alpha_i^l N_0 R^{2l-1} (\gamma t_{comm} - t_{interp} - R t_{comp}). \end{aligned}$$

Thus, the requirement for $\Delta t_{send} < 0$ is

$$\gamma t_{comm} < t_{interp} + R t_{comp}. \quad (6)$$

Given the fast processors of today, t_{interp} and t_{comp} are far smaller than t_{comm} (usually an order of magnitude) and unless one achieves a high degree of overlap ($\gamma \approx 0$) the sending processor might actually end up taking *more* time as a consequence of the data movement, i.e., $\Delta t_{send} > 0$. The conclusion is that *Strategy 1* has its inherent shortcomings.

3.1.2 Strategy 2: A Level-Clustering Approach

The obvious way to render $\Delta t_{send} < 0$ is to increase the savings in compute and interpolations costs while keeping the communication overhead unchanged. Consequently, in *Strategy 2* we consider the case of moving all patches above level "*q*" off-processor in an effort to reduce T_{total}. This leaves the original processor with levels 0 to q along with a requirement of bring back interpolated values from the off-processor $\mathcal{G}_{q+1}$. Thus the change of load on the sending processor Δt_{send} can be calculated as the cost incurred in bringing back interpolated data from $\mathcal{G}_{q+1}$ minus the compute and interpolate savings due to removal of all patches in $\mathcal{G}_{q+1}, \mathcal{G}_{q+2}, \ldots$, i.e.,

$$\begin{aligned} \Delta t_{send} &= \sum_{\mathcal{G}_q} \tau^i_{comm} \quad (7) \\ &\quad - \sum_{\mathcal{G}_m, m>q} (\tau^i_{comp} + \tau^i_{interp}) \\ &= -\sum_{l=q+1}^{L} \sum_{i=0}^{M_l-1} \alpha_i^l N_0 R^{2l} t_{comp} \\ &\quad - \sum_{l=q}^{L-1} \sum_{i=0}^{M_{l+1}-1} \alpha_i^{l+1} N_0 R^{2l+1} t_{interp} \\ &\quad + \sum_{i=0}^{M_q-1} \alpha_i^q N_0 R^{2q} \gamma t_{comm}. \end{aligned}$$

The $\sum_{l=q+1}^{L} \sum_{i=0}^{M_l-1}$ above sums up the contributions of $\mathcal{G}_{q+1}, \mathcal{G}_{q+2}, \ldots$ as does $\sum_{\mathcal{G}_m, m>q}$. By choosing a value of q, i.e., deciding at which level to truncate the grid hierarchy, Δt_{send} could be made negative. We illustrate this with an example.

Consider the case of a sub-domain with a four-level grid hierarchy with $R = 2$. Assume that this sub-domain resides on the most loaded processor. Further, assume $t_{comm} = 10$, $t_{comp} = t_{interp} = 1$ corresponding to a computer with t_{comm} an order of a magnitude larger than $t_{comp} = t_{interp}$, and $\gamma = 0.4$ and $\sum_{\mathcal{G}_1} \alpha_i^l = 0.2$, $\sum_{\mathcal{G}_2} \alpha_i^l = 0.1$ and $\sum_{\mathcal{G}_3} \alpha_i^l = 0.05$ corresponding to generally observed values for SAMR applications. Trying to reduce the load by removing a patch individually (as per *Strategy 1* discussed above), Eq. 5 shows that the sending processor (the most loaded one) will actually *increase* its running time since we will get $\Delta t_{send} > 0$. Instead — as *Strategy 2* suggests — consider the case of moving levels 2 and 3 (levels are indexed from 0) to a different processor, i.e., setting $q = 1$. Eq. 7 gives

$$\begin{aligned} \Delta t_{send} &= N_0[-(0.1 \times 2^4 + 0.05 \times 2^6) \times t_{comp} \\ &\quad -(0.1 \times 2^3 + 0.05 \times 2^5) \times t_{interp} \\ &\quad +(0.2 \times 2^2) \times \gamma \times t_{comm}] \\ &= -4N_0. \end{aligned} \tag{8}$$

Thus, moving the bi-level as a whole to the least-loaded processor rendered $\Delta t_{send} < 0$, i.e., it reduced the load on the sending processor. The conclusion is that by migrating a parent-child pair as a unit, we reap the benefit of relieving the sending processor of substantial computational and interpolation costs while incurring the (smaller) communication cost of transferring the interpolated values back to the source processor. If this reduction of load on the sending processor is excessive i.e., the least loaded processor now becomes the bottleneck, then the bi-level could be partitioned and one of the partitions moved. This approach is used in `Nature+Fable`.

3.2 Case Study: Richtmyer-Meshkow Instability

Despite the bi-level partitioning approach, the inter-level component can for some applications account for about 80 percent of the total communication costs. Figure 4 displays a communication break-down for the Richtmyer-Meshkow instability with 5 levels of refinement for 100 time-steps, partitioned by the tool `Nature+Fable`. This example indicates that ways of attacking this component explicitly is imperative.

`Nature+Fable` clusters levels in pairs, called *bi-levels*. Within these bi-levels there is no inter-level communication, since they are partitioned in a strictly domain-based fashion. Nevertheless, inter-level communication may occur in between two bi-level partitions (that is, the top layer of the lower one, and the bottom layer of the upper one), if they are not mapped onto the same processor.

Bi-levels are mapped onto processors using a partially ordered space-filling curve (SFC). In practice, if the refinements on the higher levels have the same shape and size as on the lower levels, the ordering scheme will in many cases map parent and children onto the same processor.

The problem with the SFC mapping occurs when the shape and size of the higher refinement levels differs from that of the lower levels. The result can look like an arbitrary mapping, with few cases of parent/child boxes residing on the same processor.

This paper proposes a fast heuristic re-mapping algorithm as a remedy for bad default SFC mappings.

3.3 Re-Mapping Complexity

First, some necessary definitions.

Definition 3.1. A *list of boxes* is a set of boxes where each box has a processor assignment.

Definition 3.2. A *partition* is the subset of a set of boxes containing all boxes with the same processor assignment.

Definition 3.3. A *re-mapping* is a permutation of the processor assignments for a set of partitions. Partitions are therefore regarded as atomic by the re-mapping.

According to these definitions, the re-mapping scheme can never manipulate individual boxes. As a consequence, the load balance will be unaffected by *any* re-mapping.

The greatest concern for all `Nature+Fable` algorithms besides their primary functionality is speed. Thus, all algorithms should have low complexity and

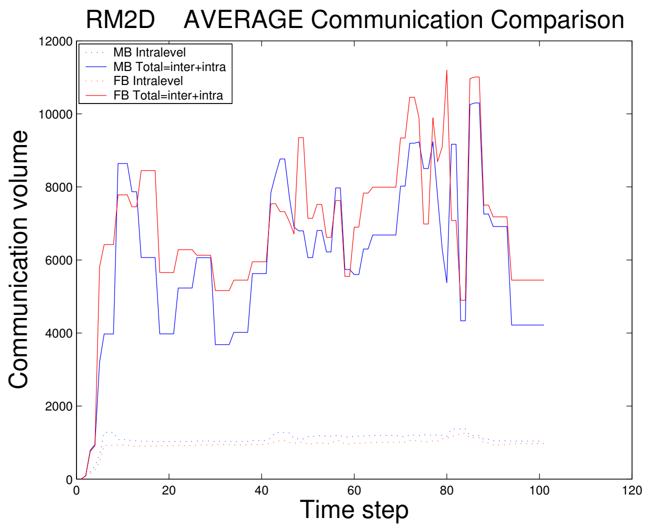


Figure 4: Communication breakdown for RM2D application with 5 levels of refinements. Blue is *multiple blocking* and red is *fractional blocking*. $P = 16$ and $k = Q = 4$.

operate on simple data structures. We start this section by discussing the inherent complexity of the remapping problem.

Assume two sets of partitions where both sets have the same number of partitions and the same processor span. Further, let one of these sets belong to refinement level $l + 1$ and the other to level $l + 2$ and place them "on top" of each other. The processor assignments will now dictate the amount of inter-level communication. Figure 5 illustrates the idea.

For simplicity reasons, assume there is exactly one box per processor and that there are n boxes (and processors). Finding the optimal solution (trying and evaluating each) require $n!$ steps. This is unacceptable for larger values on n. Moreover, in most cases, there will be a set of boxes in each partitioning. Evaluating a assignment permutation will involve substantial computational cost (checking each box for possible overlap with other boxes).

As a consequence, innovative heuristics are imperative. The next subsubsections will describe how the problem is attacked, what heuristics are used and so forth.

3.4 Re-Mapping Algorithms

We build our algorithms on some facts and experience, viz:

a) Most work is done at the higher refinement levels. Therefore, we let the top bi-level be untouched. We recursively move down the refinement levels and attack bi-level by bi-level.

b) The solution given by the partially ordered SFC is probably a good initial guess to an acceptable solution. That is, we must take advantage of this, and not start from scratch. This will lower the complexity substantially.

c) Greedy approaches are often successfully used in partitioning contexts. This is a typical case where a greedy approach could be useful.

Our proposed algorithm is composed of three steps, viz. (1) Linearizing/approximation, and (2) Removing "good-enough" partitions in the default mappings, and (3) Re-mapping of the remaining partitions. The first two steps strives to reduce the data volume and algorithm complexity. The overall complexity is $O(m^2)$ where m is the number of remaining items in the list of boxes after the first two steps. The three steps will be described in the following sections.

3.4.1 Linearizing/Approximation

To avoid costly evaluation and non-linear, highly complex data structures (i.e. e.g. arrays of lists of boxes), the first step should be to create the simpler scenario

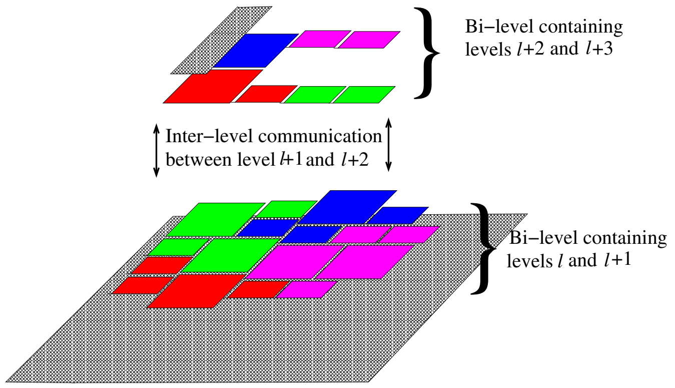

Bi−level containing levels l+2 and l+3
Inter−level communication between level l+1 and l+2
Bi−level containing levels l and l+1

Figure 5: The re-mapping problem. Note how a permutation of processor assignments for the partitions would decrease the amount of inter-level communication.

where each partition consists of exactly one box. That is, for each partition, one box is created to represent (or approximate) its partition. Depending on the given partitioning scenario, different strategies are useful. We propose two schemes for creating approximating boxes. The first is called *Union*, and the second is called *Largest*.

- *Union*. This is used in the case where `Nature+Fable` multiple blocking is used. A bounding box around all the partition's boxes (a box-union of them) would approximate the partition. Figure 6 illustrates the idea.
- *Largest*. This is used in the case where `Nature+Fable` fractional blocking is used. *Union* will not approximate the partition for the cases where processors are assigned more than one box (most processor are assigned exactly one box). In the multiple boxes per processor case, the set of boxes may be spatially spread out over the computational domain. A bounding box will therefore give little information about the partition. As a consequence, we propose to choose the largest box to represent/approximate the partition.

The complexity for both strategies are linear in the number of approximating boxes in either of the lists.

3.4.2 Removing of "Good-Enough" Partitions

It is not feasible to search for an optimal solution (it is not even clear what "optimal" means at this point) even for this simplified case of approximating boxes. Thus, we need to further lower the complexity. We do that by acknowledging that the default SFC mapping probably is a good initial guess to a high-quality solution. In view of this, we reduce the required work by reducing the size of the input. The idea is to remove pairs of partitions that as a result of the default mapping is already "good-enough".

Assume two lists A and B of approximating boxes representing partitions on level $l+1$ and level $l+2$ as in the example.

We start by introducing the parameter *threshold* $\in [0,100]$. Recall that the current mapping should be regarded as a good initial guess. Thus, intersect the boxes in list A with its counterpart in list B and remove all pairs with an intersection volume greater than *threshold* percent of the box in A. The deleted partitions are regarded as having a good-enough mapping and will not be considered further.

The complexity for this step is linear in the number of approximating boxes in either of the lists.

3.4.3 Re-Mapping of Approximating Boxes

Remaining in list A and B from the previous section are the partitions in need of re-mapping. We use a greedy approach to re-map these partitions. We start with the first box in list A, and intersect it with all boxes in list B. The one with the greatest intersection volume is considered the best choice. If no best choice was found, we assign it to the processor of the first box in B. If a best assignment was found, we greedily assign it. Last, we remove the corresponding

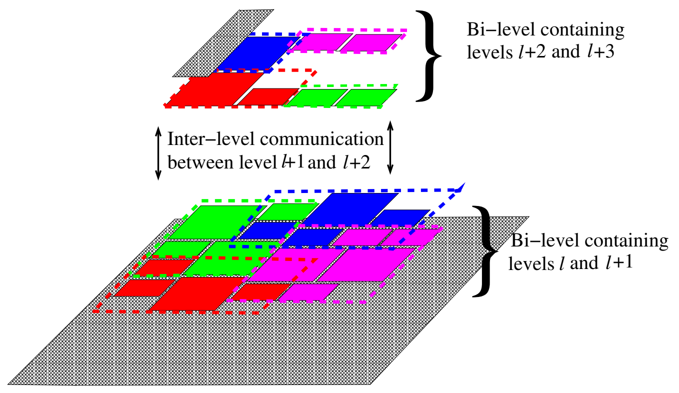

Bi–level containing levels l+2 and l+3
Inter–level communication between level l+1 and l+2
Bi–level containing levels l and l+1

Figure 6: The *Union* scheme. Note how a bounding box around each partition is created and set to represent its partition.

box from B. We then continue with the rest of the boxes in A.

This algorithm is quadratic in the number of remaining list items.

A successful re-mapping (not considering the data migration problem) of the example above is illustrated in Figure 7.

Each processor assignment considered in the remapping should be fast and simple to evaluate. Consequently, a model involving only one box-overlap operation is used.

4 Methods — Experimental Setup

4.1 Five SAMR Applications

A suite of 5 “real-world” SAMR application kernels taken from varied scientific and engineering domains are used to evaluate the effectivness of the heuristic algorithm to reduce communication. 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 mate-

[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/

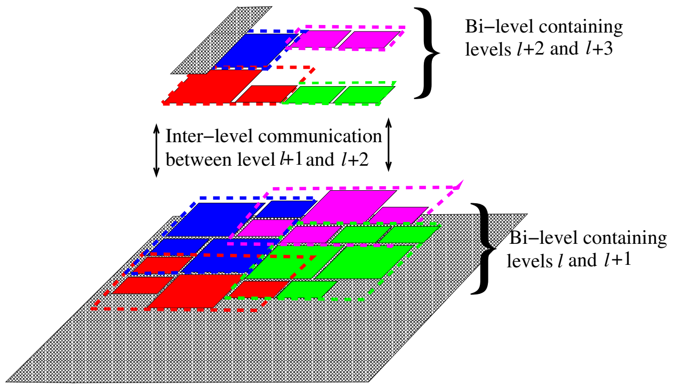

Bi–level containing levels l+2 and l+3
Inter–level communication between level l+1 and l+2
Bi–level containing levels l and l+1

Figure 7: A successful mapping of the example (not considering the migration problem).

rial 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.

4.2 Deriving the Amount of Component Communication

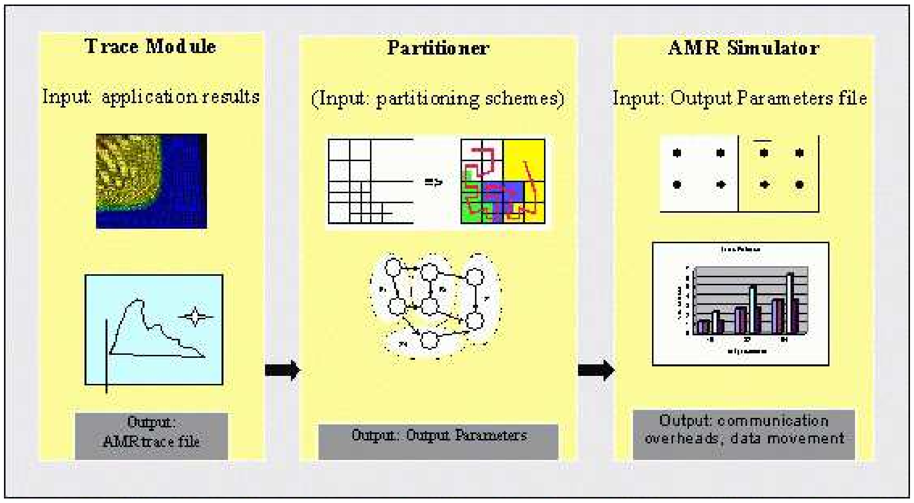
The evaluation is performed using software [33] developed at Rutgers University in New Jersey by The Applied Software Systems Laboratory, that simulates the execution of the Berger-Colella SAMR algorithm. This software is driven by an application execution trace obtained from a single processor run. 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, commuication, data migration, and overheads. The process is illustrated in Figure 8.

Note that this simulation process differs from approaches where a *parallel computer* with certain characteristics is simulated [40]. In our approach, it is rather the *SAMR algorithm* that is simulated. The simulator computes information about e.g. *the amount* of communication caused by a given partitioning — not *the cost* of this communication on a given parallel computer. Thus, our approach is independent of computer characteristics. This allows for a “fair” comparison and determination of partitioner characteristics.

Using the evaluation process described above, *communication* is the sum of the amount of interprocessor communication taken over all time-steps. *Data migration* is the sum of the total number of data points forced to migrate as a result of re-partitioning, taken over all time-steps.

4.3 Partitioning Set-Up

The blocking methods, *multiple blocking* (MB) and *fractional blocking* (FB) in `Nature+Fable` were used. Multiple blocking creates Q blocks per processor and bi-level. Fractional blocking generates exactly one block per processor for the greater part of the procesors, and some *fractional blocks* where needed in critical areas. The size of the fractions is a multiple of u/Q, where u is the “unit load” that should be assigned to each processor for establishing perfect load balance. These methods are described in full in [35]. The parameter *threshold* was set to $0, 10, 20, ..., 100$ for each application and blocking scheme. Due to the relatively small trace-file grids, the number of processors was 16. The entire parameter setting for `Nature+Fable` is listed in Table 1.



Trace Module
Input: application results
Output: AMR trace file
Partitioner
(Input: partitioning schemes)
Output: Output Parameters
AMR Simulator
Input: Output Parameters file
Output: communication overheads, data movement

Figure 8: An overview of the experimentation process. An application refinement trace from a single processor run is used to drive the selected partitioner. The resulting partitioning is then evaluated by the software that simulates the Berger-Colella SAMR algorithm. Reprinted with permission from M. Parashar.

5 Results

The effect of the proposed re-mapping algorithm with *threshold=0* for RM is presented in Table 2. Figure 5 shows a more detailed view for FB. Figures 10 through 12 illustrate the impact of the parameter *threshold* for each of the applications, and figure 13 displays a summary of the best-achieved mapping for each application and the impact on data migration.

Viewing the results, we may list a number of observations:

- The proposed re-mapping algorithm reduces the *total* communication volume with up to 30 percent (see Figure 13).
- The positive impact of the proposed re-mapping algorithm is greater for FB than for MB (see Figure 13).
- The result of the parameter *threshold* on total communication is predictable and "well-behaved" for FB but unpredictable for MB (see Figures 10 through 12).
- MB produced less communication than FB for the un-mapped cases for 3 out of 5 applications (see Figure 13).
- FB produced less communication than MB for the mapped cases for *all* applications (see Figure 13).
- FB struggles with data migration, and it gets a few percent worse as an effect of the mapping (see Figure 13).

6 Discussion, Conclusion, and Future Work

The proposed heuristic algorithm reduced inter-level communication substantially. The decrease is even larger than the numbers for *total* communication presented, and depends on the relative contribution to total communication volume. Since the complexity of the proposed algorithm is low, this decrease came at a relatively low cost. As a consequence, we draw the conclusion that the proposed re-mapping algorithm would be useful to lower overall execution times for many large SAMR applications. Due to its usefulness and its parameterization, the proposed algorithm operating within `Nature+Fable` would constitute a natural and important component of the meta-partitioner.

To lower the complexity of the re-mapping algorithm, the parameter *threshold* should be set as low as possible. A lower setting means that more approximating box pairs will be removed from the list of possible mappings. The present results show that a fairly

Parameter	Setting	Description in [35]
`smoothing`	2	Chapter 6
`growing`	2	Chapter 6
`NRmode`	Strategy 2: Off Smoothing: On Connect regions: On	Chapter 6
`actualLevels`	Off	Chapter 8
`goodEnough`	20.0	Chapter 8
`whiteSpace`	0.5	Chapter 8
`atomicUnit`	2	Chapter 4
`bMode`	MULT/FRAC	Chapter 7
`Q`	4	Chapter 7
`maxNRLoadImb`	0.10	Chapter 9
`maxVirtualPFactor`	1	Chapter 9
`mapping`	on/off	Present paper
`mapThresHold`	0-100	Present paper

Table 1: Parameter settings for `Nature+Fable` in the experiments.

Scheme	avg comm	max comm	avg migration	max migration
FB	24	14	-7	-5
MB	5	4	-3	-4

Table 2: RM2D: Results in percent improvement when using the re-mapping with *threshold=0*. FB=fractional blocking and MB=multiple blocking. Note the significant improvement on average communication for FB and the marginal improvement for MB.

low setting is sufficient for FB to generate good results. Since FB has the two advantages over MB, viz. (1) it generates fewer boxes per processor, and (2) it is much faster to compute, we conclude that the present results are sufficient to choose FB over MB in all cases where data migration does not have a significant impact on overall application execution time.

We also draw the conclusion that data migration has to be improved for the FB scheme, and we outline a model for achieving this below. Increased data migration is a major problem for re-mapping algorithms reducing inter-level communication. That is, the SFC ordering ensures that data will not move much (loosely speaking). Destroying this ordering (which is what a re-mapping will do), for one or more bi-levels, will affect to what extent data tend to stay/migrate.

Therefore, a re-mapping scheme aiming to reduce inter-level communication should be constrained to move data as few hops as possible and maintain data locality, which is important considering the hierarchical nature of parallel computers (0 hops is the local processor, 1 hop could be on the same processor board, 2 hops could be in the same SMP, and so forth). Moreover, in the next partitioning step, the same data will end up "close" to the local processor again. Having moved this data many hops, decreases the possibility for having any luck regarding not having to move the data again.

Models incorporating a weighted metric for quality will be investigated further in future research. The suggestion for incorporating a penalty for the number of hops introduced by the re-mapping algorithm is as follows (assume two boxes a and b from the lists A and B):

$$\text{quality} = \alpha * \text{intersect}(a,b) + (1-\alpha) * \frac{1}{\epsilon + \text{nrOfHops}}$$

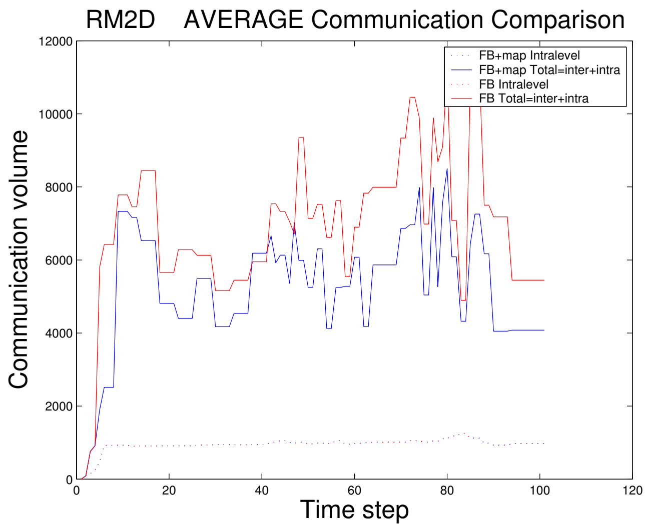


Figure 9: RM: Results for the fractional blocking scheme with *threshold=0*. The blue is with mapping and red is without. Note a 24 percent improvement of total communication.

where $0 \leq \alpha \leq 1$ and ϵ is a small number. This is a simple linear model allowing for the weighting of communication and data migration costs.

In this paper, we proposed a fast heuristic algorithm reducing the amount of inter-level communication in parallel SAMR applications partitioned by `Nature+Fable`. Reducing this component may be crucial to reduce execution times for many SAMR applications. The proposed algorithm was particularly useful for the FB partitioning method in `Nature+Fable`, for which the reduced amount of communication volume was established reliably and fast. With the proposed re-mapping algorithm, `Nature+Fable` takes another step towards a complete implementation of the meta-partitioner.

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.

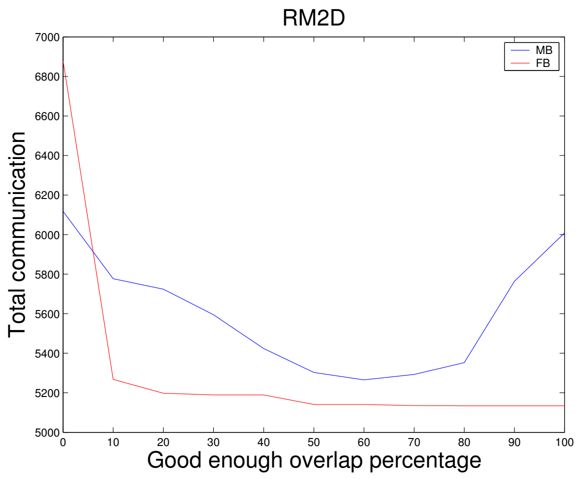
[7] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82, 1989.

[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.



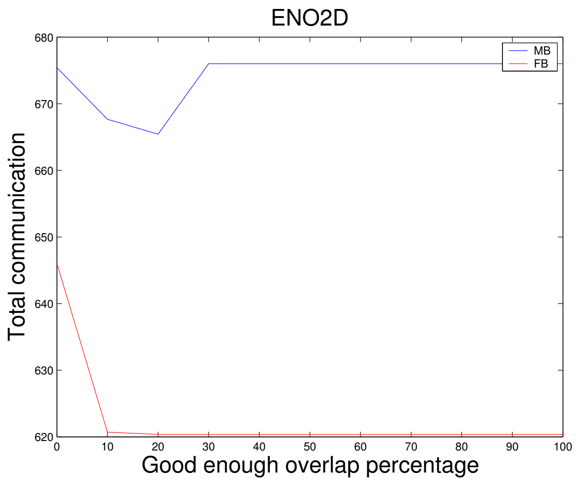
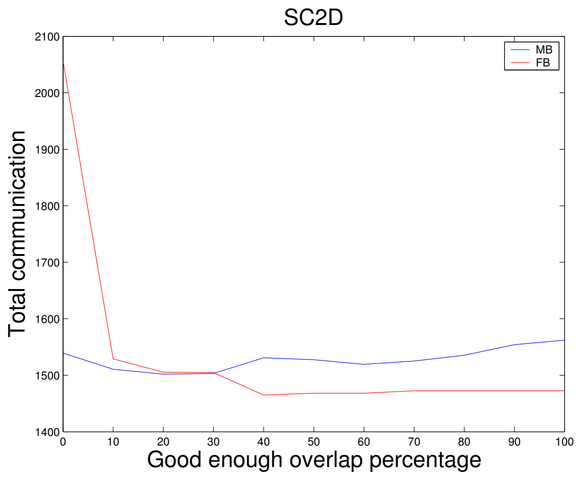


Figure 10: Impact of the parameter *threshold* for the applications RM (left) and ENO (right).



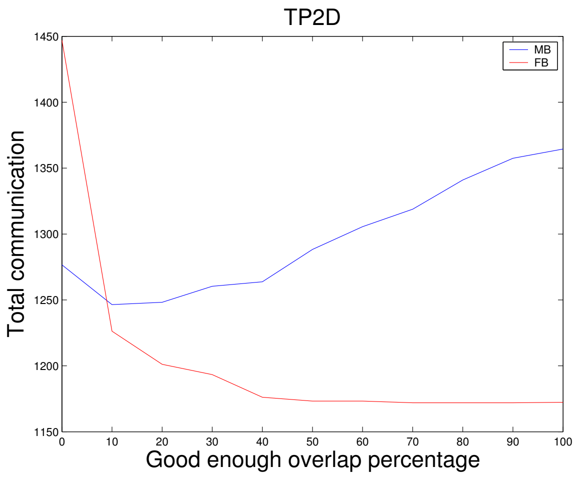


Figure 11: Impact of the parameter *threshold* for the applications SC (left) and TP (right).

[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.

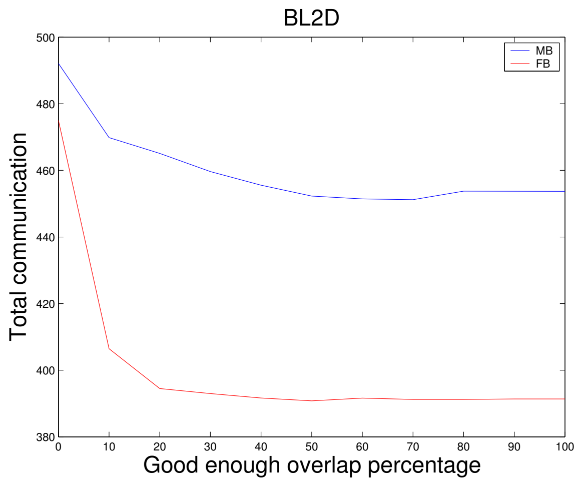


Figure 12: Impact of the parameter *threshold* for the application BL.

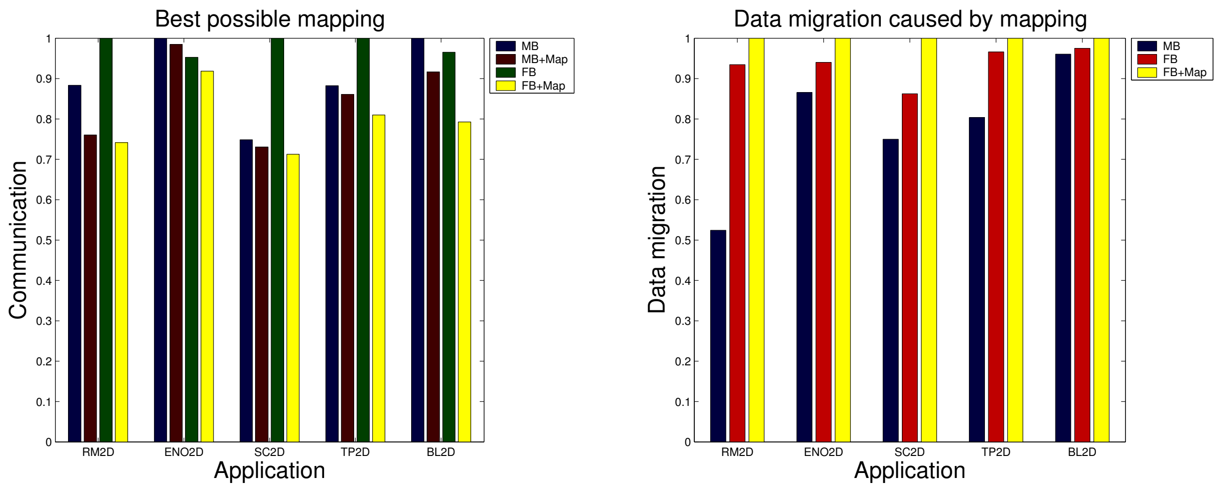


Figure 13: Best-achieved mappings for the all applications (left) and the impact on data migration (right).

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

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

[39] 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.

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

[41] 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.

[42] 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.

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

Distribution List

External Distribution

1	Manish Parashar Department of Electrical & Computer Engineering Rutgers, The State University of New Jersey 94 Brett Road, Piscataway, NJ 08854-8058
1	Sumir Chandra CAIP Center at Rutgers University CORE Building, Busch Campus 96 Frelinghuysen Rd. Piscataway, NJ 08855-1390
1	Michael Thuné Information Technology Department of Scientific Computing P.O. Box 337, SE-751 05 Uppsala, Sweden
1	Jarmo Rantakokko Information Technology Department of Scientific Computing P.O. Box 337, SE-751 05 Uppsala, Sweden

Internal Distribution

2	MS 9915	Johan Steensland, 8961
1	MS 9051	Jaideep Ray, 8961
3	MS 9018	Central Technical Files, 8945-1
1	MS 0899	Technical Library, 9616
1	MS 0612	Classification Office, 8511 for Technical Library, MS 0899, 9616 DOE OSTI via URL