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
{jsteens, jairay}@ca.sandia.gov

11th August 2003

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

Computer simulations are often based on solving partial differential equations by numerical methods. Adaptive methods are crucial to efficiently use computer resources. But even with adaptation, the simulations are computationally demanding and yield huge data sets. Parallelization and efficient data partitioning are necessary. Adaptation causes the workload to change dynamically, calling for *dynamic* (re-) partitioning to maintain efficient resource utilization.

The proposed algorithm reduced inter-level communication substantially. Since the complexity of the algorithm is low, this decrease was relatively inexpensive. We draw the conclusion that the algorithm could lower overall execution times for many large SAMR applications, and that, due to its parameterization, it would constitute a natural component of the meta-partitioner.

Keywords: Dynamic load balancing, structured adaptive mesh refinement, meta-partitioner, run-time management, communication cost.

1 Introduction

This paper presents a heuristic re-mapping algorithm for the built-in partitioning techniques in `Nature+Fable` [34], and experimentally shows the effectiveness of this algorithm in reducing inter-level communication for five vastly different structured adaptive mesh (SAMR) applications. Particular attention will be paid to simulating the Richtmyer-Meshkov instabilities using an explicit time-stepping and a finite volume scheme.

The presented work is part of an ongoing research project [33, 34, 35, 9] 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 [12, 10].

Methods based on SAMR for the numerical solution to partial differential equations (PDEs) [6, 7, 30] employ locally optimal approximations, and can yield highly advantageous ratios for cost/accuracy when compared to methods based on a static uniform mesh. These methods are being effectively used in many domains, including computational fluid dynamics [2, 5, 27], numerical relativity [13, 29], astrophysics [1, 8, 22], and subsurface modeling and oil reservoir simulation [41, 24]. 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 primary motivation for our research is that *no single partitioning scheme performs the best* for all types of SAMR applications and computer systems. For a given

*This paper is submitted to the *15th IASTED International Conference Parallel and Distributed Computing and Systems 2003 (PDCS 2003)*.

application, the most suitable partitioning technique depends on input parameters and the application's runtime state [28, 34]. This means sometimes focusing on optimizing load balance, and sometimes focusing on lowering the interprocessor communication costs. This necessitates adaptive management of these applications at runtime. This includes using application runtime state to select and configure the partitioning strategy to balance load, optimize communication and synchronization, minimize data migration costs, and maximize grid quality (e.g. aspect ratio) and available parallelism. The goal of the adaptive *meta-partitioner* is to provide such a capability for parallel SAMR applications. A means to explicitly attack the amount of inter-level communication is crucial to the adaptive meta-partitioner.

Partitioners for SAMR grid hierarchies can be classified as patch-based [4, 18], domain-based [23, 28, 37, 33], or hybrid [37, 19, 34].

`Nature+Fable` (**Natur**al **Re**gions + **Fra**ctional blocking and **bi**-**le**vel partitioning) [34] 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. They offer carefully designed parameters to steer component behavior enabling adaptation to varying partitioning requirements. As `Nature+Fable` matures, it is intended to transform into the meta-partitioner.

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.

2 SAMR and Related Work

In 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.

Existing software infrastructures include Paramesh [20, 21], a FORTRAN library for parallelization of and adding adaption to existing serial structured grid computations, SAMRAI [17, 42] a C++ object-oriented framework for implementing parallel structured adaptive mesh refinement simulations, and GrACE [25] and CHOMBO[?], both of which are adaptive computational and data-management engines for enabling distributed adaptive mesh-refinement computations on structured grids.

Parallel implementations of SAMR methods present interesting challenges in dynamic resource allocation since the overall efficiency 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 [34].

`Nature+Fable` (**Natur**al **Re**gions + **Fra**ctional blocking and **bi**-**le**vel partitioning) [34] 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. They offer carefully designed parameters to steer component behavior enabling adaptation to varying partitioning requirements. As `Nature+Fable` matures, it is intended to transform into the meta-partitioner.

3 A Heuristic Algorithm for Reducing Inter-Level Communication

The bi-level partitioning approach is appealing, since [34] (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 an 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

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 [6] 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

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

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,

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

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 processor* 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

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

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.

In the patch-based *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_{comp}^i and interpolation τ_{interp}^i 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

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

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.

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 the level-clustering approach 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.,

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

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.

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 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 1 (left) 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. Our proposed algorithm is 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.

Given these definitions, the re-mapping scheme can never manipulate individual boxes. The load balance will be unaffected by *any* re-mapping.

The greatest concern for all `Nature+Fable` algorithms besides their primary functionality is speed. All algorithms should have low complexity and operate on simple data structures. We now discuss the inherent complexity of the re-mapping problem.

Consider two sets of partitions, both having the same number of partitions and the same processor span. 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 as illustrated by Figure 1 (right). The processor assignments will now dictate the amount of inter-level communication.

For simplicity, 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. Moreover, most

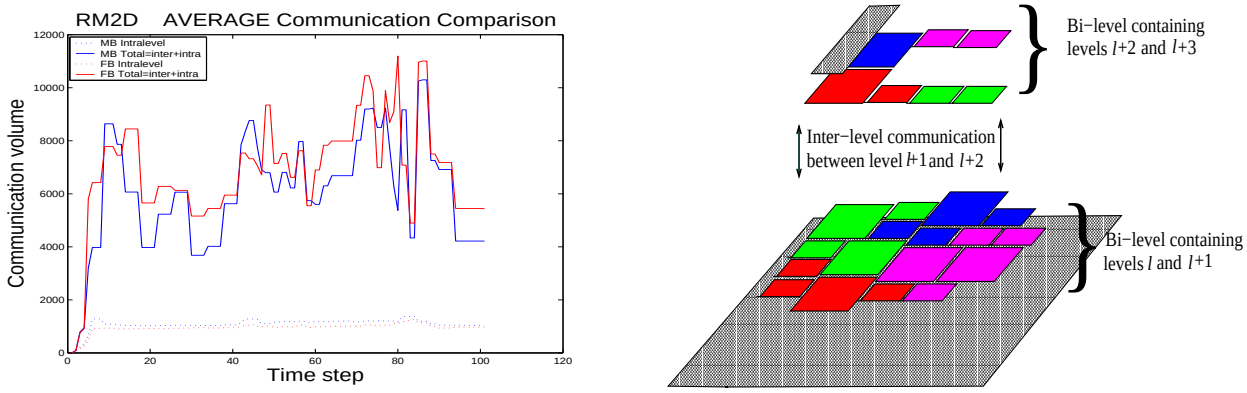

Figure 1: Left: Communication breakdown for RM2D application with 5 levels of refinements. Blue is *multiple blocking* and red is *fractional blocking*. $P = 16$ and $k = Q = 4$. Right: The re-mapping problem. Note how a permutation of processor assignments for the partitions would decrease the amount of inter-level communication.

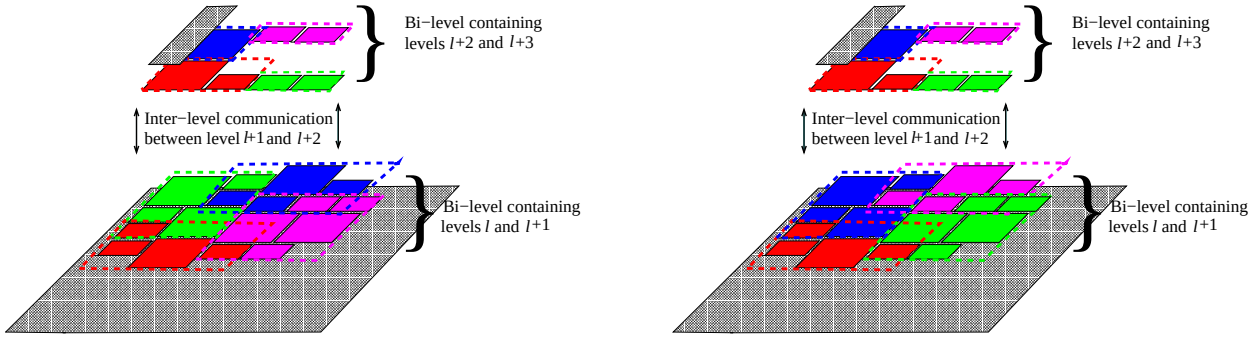

Figure 2: Left: The *Union* scheme. Note how a bounding box around each partition is created and set to represent its partition. Right: A successful mapping of the example (not considering the migration problem).

often there will be a *set* of boxes in each partitioning. Evaluating an assignment permutation will involve substantial computational cost (checking each box for possible overlap with other boxes). Consequently, we conclude that innovative heuristics are imperative.

3.4 Re-Mapping Algorithms

We build our algorithms on some facts and experience, viz: a) Since most work is done at the higher refinement levels, we let the top bi-level be untouched. We recursively move down the refinement levels and attack bi-level by bi-level, b) The partially ordered SFC mapping is probably a good initial guess to an acceptable solution. We take advantage of this (do not start from scratch), which lowers the complexity substantially, c) Greedy approaches are often successfully used in partitioning contexts. We use a greedy approach for processor re-assigningment.

Our 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 (e.g. arrays of lists of boxes), the first step creates the simpler scenario where each partition consists of exactly one box. For each partition, one box will represent (or approximate) its partition. Depending on the given partitioning scenario, different strategies are useful. We propose two schemes for creating approximating boxes:

- *Union.* This is used when *multiple blocking* is used. A bounding box around all the partition's boxes (a box-union of them) approximates the partition as illustrated by Figure 2 (left).
- *Largest.* This is used when *fractional blocking* is used. *Union* will not approximate the partition when processors are assigned more than one box (most processor are assigned exactly one box).The set of boxes may be spatially spread out over the computational domain. A bounding box will give

little information about the partition. We propose to to represent/approximate the partition with its largest box.

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

3.4.2 Removing "Good-Enough" Pairs

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 established by the linearization/approximation. We further lower the complexity by acknowledging that the default SFC mapping probably is a good initial guess to a high-quality solution. Given this, we can reduce the required work by reducing the size of the input. We remove pairs of partitions that as a result of the default mapping are already "good-enough".

Consider two lists A and B of approximating boxes representing partitions on level $l+1$ and level $l+2$ as in the example above. Let *threshold* $\in [0, 100]$. 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 pairs 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

We use a greedy approach to re-map the partitions remaining in list A and B efter the previous algorithmic steps. We start with the first box in list A, and intersect it with all boxes in list B. The greatest intersection volume is considered the best choice, and the box in A is greedily assigned to the processor of the matching box in B. If no best choice was found, we assign it to the processor of the first box in B. Last, we remove the corresponding 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 2 (right).

4 Methods — Experimental Setup

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 (Scalar-wave), 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 evaluation is performed using software [32] 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 [34] with the components load balance, commuication, data migration, and overheads.

Using the evaluation process described above, *communication* is the sum of the amount of inter-processor 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.

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. 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.

5 Results

The effect of the proposed re-mapping algorithm with *threshold=0* for RM is presented in Table 1. Figure 3 (left) shows a more detailed view for FB. Figure 3 (right) illustrates the impact of the parameter *threshold* for the TP application. This result was typical so the plots for the remaining applications are not shown. Figure 4 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 4).

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

Table 1: 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.

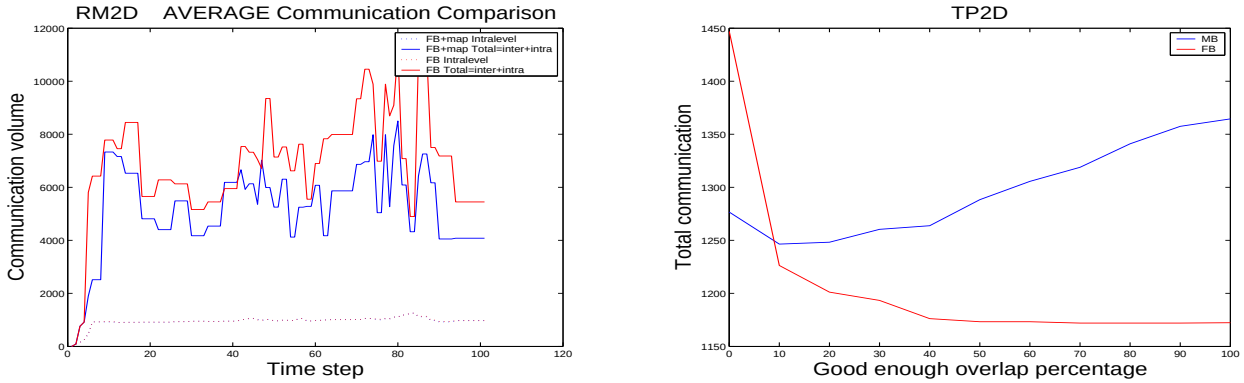

Figure 3: Left: RM: Results for the fractional blocking scheme with *threshold=0*. The blue is with mapping and red is without. Note a 24 percent improvement of total communication. Right: Impact of the parameter *threshold* for the application TP (result typical).

- The positive impact of the proposed re-mapping algorithm is greater for FB than for MB (see Figure 4).
- The result of the parameter *threshold* on total communication is predictable and "well-behaved" for FB but unpredictable for MB (see Figure 3 (right)).
- MB produced less communication than FB for the un-mapped cases for 3 out of 5 applications (see Figure 4).
- FB produced less communication than MB for the mapped cases for *all* applications (see Figure 4).
- FB struggles with data migration, and it gets a few percent worse as an effect of the mapping (see Figure 4).

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 our algorithm is low, this decrease was relatively inexpensive. We draw the conclusion that the algorithm could lower overall ex-

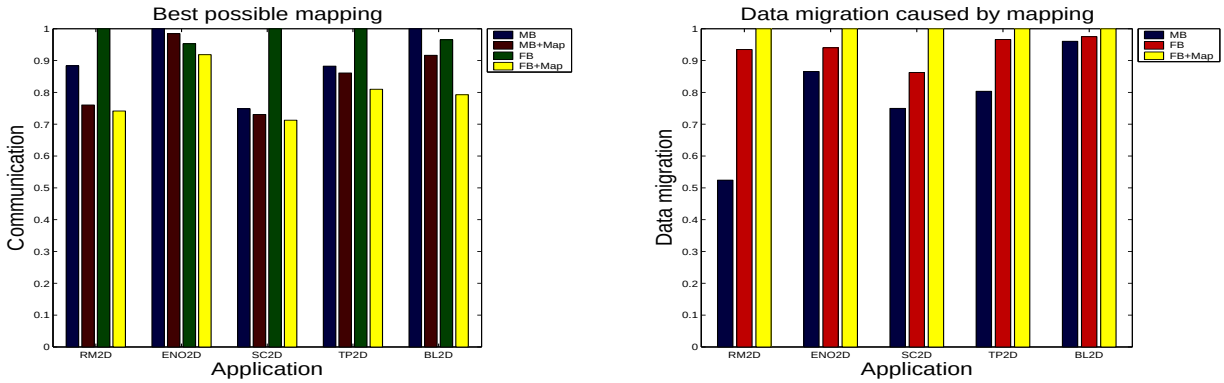

Figure 4: Best-achieved mappings for the all applications (left) and the impact on data migration (right).

ecution times for many large SAMR applications, and that, due to its parameterization, it would constitute a natural 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 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 significantly faster to compute, the present results support choosing FB over MB when 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. Models incorporating a weighted metric for quality will be investigated further in future research. A 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):

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

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 is crucial to reduce execution times for many SAMR applications. The algorithm was particularly useful for the FB method, for which the reduced amount of communication volume was established reliably and fast. With the re-mapping algorithm, `Nature+Fable` takes another step towards a complete implementation of the meta-partitioner.

Acknowledgments

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

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

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

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

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

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

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

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

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

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

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

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

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

[32] 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.

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

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

[35] 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.

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

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

[38] 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.

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

[40] 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.

[41] 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.

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