ARMaDA: An Adaptive Application-Sensitive Partitioning Framework for SAMR Applications*

Sumir Chandra and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
94 Brett Road, Piscataway, NJ 08854 USA
E-mail: {sumir, parashar}@caip.rutgers.edu

ABSTRACT
Distributed implementations of dynamic adaptive mesh refinement techniques offer the potential for accurate solutions of physically realistic models of complex physical phenomena. However, configuring and managing the execution of these applications presents significant challenges in resource allocation, data-distribution and load-balancing, communication and coordination, and runtime management. This paper presents the design and evaluation of the ARMaDA framework for adaptive application-sensitive partitioning of dynamic structured adaptive mesh refinement applications. The ARMaDA framework has three components: application state characterization component, octant-partitioner mapping policy-base, and adaptive meta-partitioner component that dynamically selects and configures partitioning strategies at runtime based on current application state. Experimental results show that adaptive application-sensitive partitioning using the ARMaDA framework can improve application performance as compared to non-adaptive partitioning.

KEY WORDS
Adaptive application-sensitive partitioning, Dynamic SAMR applications.

1 Introduction

Dynamic adaptive mesh refinement (AMR) methods for the solution of partial differential equations that employ locally optimal approximations can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. Distributed implementations of these techniques offer the potential for accurate solutions of physically realistic models of complex physical phenomena and can provide new insights into complex systems such as interacting black holes and neutron stars, formations of galaxies, subsurface flows in oil reservoirs and aquifers, and dynamic response of materials to detonation. However, configuring and managing the execution of these dynamic applications presents significant challenges in resource allocation, data-distribution and load-balancing, communication and coordination, and runtime management.

The research presented in this paper is motivated by the observation that no single partitioning scheme performs the best for all types of applications and systems. Even for a single application, the most suitable partitioning technique and associated partitioning parameters depend on input parameters and the application's runtime state. This necessitates adaptive partitioning and runtime management of these dynamic applications using an application-centric characterization of domain-based partitioners.

This paper presents the design, implementation, and evaluation of "ARMaDA" - an adaptive application-sensitive partitioning framework for structured adaptive mesh refinement (SAMR) applications. The framework has three components: application state characterization component, octant-partitioner mapping policy-base, and adaptive meta-partitioner component. The ARMaDA framework described in this paper makes three key contributions: (1) It establishes mechanisms for characterizing the state of the adaptive application and abstracting current computational, communication and storage requirements, (2) It determines the appropriate partitioning strategy by using policies that map runtime state octants to suitable partitioners, and (3) It selects and configures the appropriate partitioner and associated partitioning parameters and dynamically switches these at runtime. The partitioners constitute a selection from popular software tools such as GrACE [5] and Vampire [7].

The rest of the paper is organized as follows. Section 2 outlines the related research in dynamic adaptive partitioning and load balancing. Section 3 briefly describes an adaptive meta-partitioner for SAMR applications. Section 4 details the design and implementation of the ARMaDA framework. The evaluation of the framework for varied SAMR kernels is presented in Section 5. Section 6 presents concluding remarks.

*The work presented here was supported in part by the National Science Foundation via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), and by DOE ASCI/ASAP (Caltech) via grant number PC295251. The authors would like to thank Johan Steensland for access to Vampire in this research.

2 Related Work

There is ongoing research in the field of dynamic adaptive partitioning and dynamic load balancing for AMR applications. Whereas dynamic adaptive partitioning techniques have been extensively investigated for unstructured meshes, such schemes for structured grids are relatively unexplored. Our goal is to adaptively manage dynamic applications on structured grids based on the runtime state using a characterization of the available options.

PLUM [4] is a dynamic load-balancing strategy for adaptive unstructured-grid computations that uses a cost-metric model for efficient mesh adaptation. The PLUM model uses computation, communication, and data-remapping weights to implement accurate metrics that estimate and compare the computational gain and the redistribution cost of having a balanced workload after each mesh adaptation step. The new partitioning and mapping are accepted if the computational gain is larger than the redistribution cost.

The Unified Repartitioning Algorithm [6] is a parallel adaptive repartitioning scheme for the dynamic load-balancing of scientific simulations. A relative cost factor is used to obtain a cost function to minimize the communication and data redistribution costs. In the initial partitioning phase, repartitioning is performed on the coarsest graph twice by alternative methods and the one with the lowest cost is selected.

In the dynamic load balancing scheme for SAMR [3], the moving-grid phase and splitting-grid phase execute in parallel. In the moving-grid phase, load balancing is triggered if there is imbalance after each adaptation. The splitting-grid phase is invoked if imbalance still exists, and it splits the grid along the longest dimension into two subgrids. Eventually, either the load is balanced or the minimum allowable grid size is reached.

Our previous work presents the application-centric characterization of domain-based partitioners for SAMR applications and investigates the design of an adaptive meta-partitioner [1, 8, 9]. In [2], we demonstrate application performance improvement by an experimental study of adaptive application-sensitive partitioning that uses a manually formulated adaptive policy for the dynamic switching of partitioners. The ARMaDA framework presented in this paper is based on and extends previous research to characterize application runtime state in an automated manner and dynamically select and configure the appropriate partitioning strategy to be used. Moreover, this automated approach does not require application characteristics to be known *a priori*.

3 Adaptive Meta-partitioner for SAMR Applications

Fig. 1 illustrates the design of an adaptive meta-partitioner for SAMR applications. Using the octant approach, the runtime environment is characterized based on the application state. The current octant is then used to select the most appropriate partitioning technique P from a database of available partitioning techniques. The selected partitioner is configured with appropriate parameters such as partitioning granularity and threshold, and then invoked (using a common interface) to partition the SAMR grid hierarchy. Such a mapping is derived based on an experimental characterization of partitioning techniques and application states [8, 9]. We believe that such a meta-partitioner will be able to significantly improve the efficiency and performance of parallel/distributed SAMR applications.

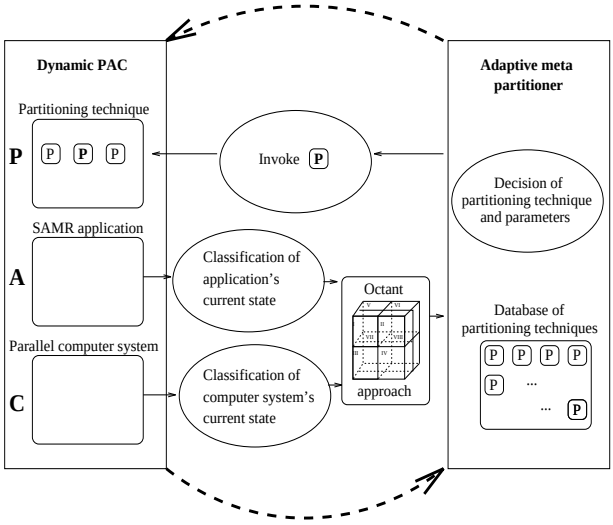

Figure 1. Adaptive meta-partitioner selects appropriate partitioner at runtime. Figure reproduced from [8]

4 ARMaDA Framework

The ARMaDA framework has three components: application state characterization component, octant-partitioner mapping policy-base, and adaptive meta-partitioner component. The state characterization component implements mechanisms that abstract the current application state in terms of the computation/communication requirements, application dynamics, and the nature of the adaptation. The policy-base provides an association for mapping octants to partitioners. Subsequently, the meta-partitioner component dynamically sets the appropriate partitioner and associated partitioning parameters at runtime.

Dynamic history of the application characteristics enables better selection of the appropriate partitioning strategy by using a "sliding history window" to avoid thrashing and to obtain scaled ratios of the quality metric. *Thrashing* may occur due to very frequent state changes and can result in high overheads due to rapid switching of partitioners. Moreover, fast algorithms are used for sensing and characterizing runtime state to minimize the ARMaDA framework overheads. Application

state thresholds (LOW_THRESH and HIGH_THRESH) determine when a change in partitioner and partitioning parameters is warranted. Application-specific metric weights fine-tune the sensitivity of the quality metric to closely match the needs of the dynamic application. Additionally, state-sensing optimizations help to improve the ARMaDA framework efficiency. If the application state remains unchanged for a specific duration (set by SAME_REGRID_LIMIT), the framework stops runtime state sensing for SKIP_SAME_REGRID steps. Also, when a change in runtime state causes a switch in partitioners, the application characteristics do not significantly change in the immediately succeeding stages. Hence, runtime state sensing is paused for SKIP_SWITCH steps.

4.1 Runtime Characterization of Application State

The current application state is abstracted from the existing grid hierarchy and newly defined clusters at runtime, and is expressed in terms of bounding boxes at various levels of refinement. The application state is characterized in terms of the computation/communication requirements, application dynamics, and the nature of the adaptation.

Computation/Communication Requirements:
The computation-to-communication ratio ("CCratio") determines whether the current application state is computationally-intensive or communication-dominated. Using a simple geometrical analogy, the volume of the application state bounding boxes at different levels of refinement gives a measure of the computation expressed in work units, whereas communication requirements are represented by the surface area of the bounding boxes. The ratio of the total volume to total surface area of the current state bounding boxes determines CCratio.

$$\text{CCratio} = \frac{\sum(\text{Volume of bounding boxes})}{\sum(\text{Surface area of bounding boxes})}$$

Application Dynamics:
The application dynamics ("Dynamics") indicates the speed of application refinement pattern changes and is extracted as the degree of overlap between current application state bounding boxes and those of the previous state. A greater overlap region across states indicates that the application has not changed significantly and has low dynamics. A smaller intersection region represents high activity dynamics.

$$\text{Dynamics} = \text{Size of}\big(\text{Current state} \cap \text{Previous state}\big)$$

Nature of Adaptation:
The nature of the adaptation ("Adapt") represents whether the refinements are scattered or localized. A simple implementation determines adaptation based on the geometrical domain space occupied as compared to overall domain under consideration as well as the number of coarse level bounding boxes. A higher number of coarse boxes is an indicator of possibly scattered refinements, typically giving rise to greater overheads. Since differences in adaptation overheads do not significantly affect partitioner performance, this implementation suffices the assessment of adaptation.

$$\text{Adapt} = \frac{\text{Coarse volume}}{\text{Domain volume}} * \text{No. of coarse boxes}$$

4.2 Determining Appropriate Partitioning Strategy

As outlined above, the ARMaDA framework abstracts measures of "CCratio", "Dynamics", and "Adapt" from the current application state. Let M denote any one metric that is abstracted from the current application state at regrid step r, and $currM_r$ and $prevM_r$ represent the current and previous metric measures. Note that $prevM_r$ = $currM_{r-1}$. The current state measures are normalized against previous state measures to obtain current state factors. Hence, current state factor for metric M (denoted as $currMfactor$) is given by

$$\text{currMfactor} = \frac{\text{currM}_r}{\text{prevM}_r} = \frac{\text{currM}_r}{\text{currM}_{r-1}}$$

The current state factors are used in conjunction with previous state factors to obtain the scaled metric ratios ($Mratio$).

$$\text{Mratio} = \frac{\text{currMfactor}}{\text{prevMfactor}} = \frac{\text{currM}_r * \text{currM}_{r-2}}{(\text{currM}_{r-1})^2}$$

This equation indicates a *three regrid-step history* sliding window size incorporated within the ARMaDA framework for the dynamic application. The scaled metric ratio ensures that sudden glitches do not cause "run-away" metric measures and arbitrarily large or small ratio values.

The ARMaDA framework computes three metric ratios, namely computation/communication ratio (Cratio), application dynamics ratio (Dratio), and the adaptation ratio (Aratio). The three scaled metric ratios are then mapped directly to octant positions using a three-bit binary pattern. Let $Mratio$ and $Mbit$ represent the scaled ratio and the octant-bit value for a particular metric M. If $lowMwt$ and $highMwt$ denote the application metric weights for the low and high thresholds respectively, then

$$\begin{aligned} \text{Mbit} &= 0, \text{Mratio} < \text{lowMwt*LOW\_THRESH} \\ &= 1, \text{Mratio} > \text{highMwt*HIGH\_THRESH} \\ &\rightarrow \text{else, same as Mbit} \end{aligned}$$

The ARMaDA framework establishes the values for three bits - "Cbit", "Dbit", and "Abit", the combination of which represent a specific octant location (0-7). A low Cbit (Cbit=0) represents more communication while a high Cbit (Cbit=1) indicates greater computation. A low Dbit (Dbit=0) indicates greater activity whereas a high Dbit

(Dbit=1) represents lesser application dynamics. A low Abit (Abit=0) denotes more localized adaptation while a high Abit (Abit=1) indicates that the nature of adaptation is scattered.

C D A	ARMaDA octant	Octant position	Suitable partitioner
0 0 0	0	V	pBD-ISP
0 0 1	1	VI	pBD-ISP
0 1 0	2	I	pBD-ISP, G-MISP+SP
0 1 1	3	II	pBD-ISP
1 0 0	4	VII	G-MISP+SP
1 0 1	5	VIII	G-MISP+SP, SFC
1 1 0	6	III	G-MISP+SP, SP-ISP, SFC
1 1 1	7	IV	G-MISP+SP, SP-ISP, SFC

Table 1. ARMaDA framework octant implementation

Table 1 shows the framework implementation octants representing various application states and their mapping to the original octant positions and suitable partitioners based on runtime state, as described in the octant approach [8, 9]. After octant characterization, the ARMaDA framework selects and invokes the appropriate partitioner through a common interface.

5 Framework Evaluation

The evaluation of the ARMaDA framework is performed for varied SAMR application kernels on different computing platforms. The experiments consist of measuring application execution times for different partitioner configurations using the ARMaDA framework. Except for the partitioning strategy and the associated partitioning granularity, all other application-specific and refinement-specific parameters are kept constant. The different partitioners in the ARMaDA framework include SFC (GrACE), G-MISP+SP (Vampire), pBD-ISP (Vampire), and adaptive combinations of these partitioners based on runtime state. In the adaptive run of the ARMaDA framework, the initial partitioner is set by the user along with other partitioning parameters and thresholds through a "param" file. The framework uses this initial partitioner as the starting-point and then determines the adaptive runtime strategy to be applied.

5.1 TportAMR-2D on "Discover"

"Discover" is a 16-node Linux Beowulf cluster at Rutgers University. The experimental tests on Discover aim to evaluate the average performance of the ARMaDA adaptive partitioning framework and ascertain the effect of initial partitioner selection on the performance of the framework.

The Transport AMR 2D (TportAMR-2D) application is a benchmark kernel solving the transport equation and is primarily communication-dominated with high adaptation overheads. The evaluation is performed for 8 processors and the application uses a base grid of size 64*64 and 3 levels of factor 2 space-time refinements. Regriding is performed every 4 time-steps at each level and the application runs for 40 iterations. The experimental tests compare individual partitioner runs with the performance of the adaptive configurations based on the corresponding initial partitioner. The application execution times for TportAMR-2D application on 8 processors for different partitioning configurations are listed in Table 2.

Partitioner	SFC alone (no ARMaDA)	ARMaDA with SFC start
Execution time (sec)	352.091	349.286
Partitioner	G-MISP+SP alone (no ARMaDA)	ARMaDA with G-MISP+SP start and thrashing
Execution time (sec)	360.946	353.44
Partitioner	pBD-ISP alone (no ARMaDA)	ARMaDA with pBD-ISP start
Execution time (sec)	344.558	342.646

Table 2. ARMaDA evaluation for TportAMR-2D application on 8 processors on "Discover"

The TportAMR-2D application has a small domain and is not computationally intensive but requires considerable data movement. Since the pBD-ISP partitioner is suited to strongly communication-dominated application states and reduces communication and data migration costs [8, 9], it is expected to perform better than the other partitioners. The ARMaDA partitioner with pBD-ISP start fares the best (same as pBD-ISP scheme alone), conforming to the above theory. If a different initial partitioner is chosen, the ARMaDA adaptive partitioner dynamically switches the initial partitioner and uses the pBD-ISP strategy instead. For initial partitioners other than pBD-ISP, the ARMaDA adaptive technique performs better than if the corresponding scheme is used alone.

The framework overhead for this evaluation is in the range of 40-100 milliseconds for the TportAMR-2D application run with duration of around 350 seconds. Thrashing also does not severely affect the ARMaDA framework performance because the framework design and state-sensing optimizations minimize the overheads of switching partitioners. On account of small domain size, lesser computational grid-points, and fewer application features, the various partitioners of the ARMaDA framework do not demonstrate significantly different execution times for this evaluation.

The choice of the initial partitioner affects the performance of the ARMaDA framework - the framework is good, in the worst case, and excellent when a suitable initial partitioner is selected. When applied to adaptively manage SAMR applications whose runtime characteristics and functional behavior are not known *a priori*, the ARMaDA framework proves especially beneficial.

5.2 VectorWave-2D on "Discover"

The VectorWave-2D application is a coupled set of partial differential equations and forms a part of the Cactus 2-D numerical relativity toolkit, solving Einstein's and gravitational equations. The experimental evaluation on Discover is performed for 16 processors and the application uses a base grid of size 64*64 and 3 levels of factor 2 space-time refinements. Regriding is performed every 4 time-steps at each level and the application runs for 60 iterations. The VectorWave-2D application execution times for individual partitioner runs and the adaptive ARMaDA partitioner with G-MISP+SP start are presented in Table 3.

Partitioner	Execution time (sec)
SFC	192.848
G-MISP+SP	174.263
pBD-ISP	188.199
ARMaDA with G-MISP+SP start	172.703

Table 3. ARMaDA evaluation for VectorWave-2D application on 16 processors on "Discover"

The VectorWave-2D application is computationally-intensive and requires good load-balancing and reduced data migration overheads. G-MISP+SP and even pBD-ISP are the preferred partitioners for such application states, as verified in Table 3. In this evaluation, the ARMaDA partitioner with G-MISP+SP start gives the best performance among the different partitioning configurations. The state sensing and framework overhead in this case is 0.236 seconds, which is negligible. The execution times are not significantly different, which can be attributed to the small problem domain size and fewer number of processors.

5.3 RM2D & RM3D on "Blue Horizon"

The NPACI IBM SP2 "Blue Horizon" is a teraflop-scale Power3 based clustered SMP system at the San Diego Supercomputing Center. It supports shared-memory symmetric multi-processing (via OpenMP or PThreads) within a node and message passing (via MPI) between nodes. The machine contains 1152 processors arranged as 144 SMP compute nodes and 512 GB of main memory, with AIX as the operating system.

The evaluation on Blue Horizon is performed on 64 processors using the 2-D (RM2D) and 3-D (RM3D) versions of the compressible turbulence application kernel solving the Richtmyer-Meshkov instability. The 2-D version has a base grid of size 128*32 and runs for 60 iterations whereas the RM3D evaluation uses a base grid of size 128*32*32 and executes for 100 iterations. Both evaluations use 3 levels of factor 2 space-time refinements and regriding is performed every 4 time-steps at each level.

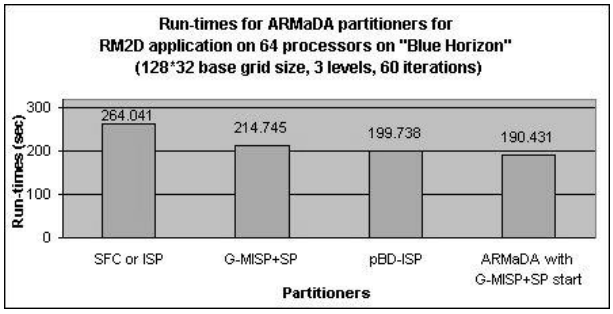

Figure 2. RM2D execution times for ARMaDA partitioners on 64 processors on "Blue Horizon"

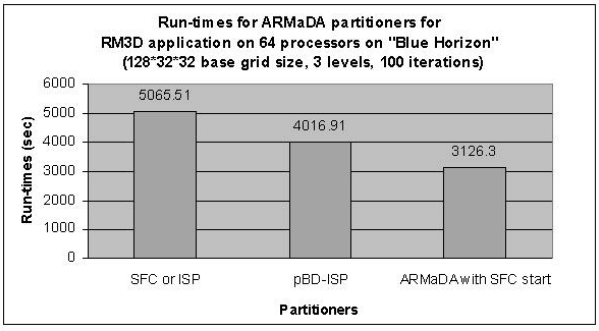

Figure 3. RM3D execution times for ARMaDA partitioners on 64 processors on "Blue Horizon"

The application execution times for RM2D and RM3D on 64 processors for different partitioning configurations in the ARMaDA framework are shown in Figs. 2 and 3 respectively. The application starts out with greater computation requirements, low activity dynamics, and more scattered adaptation, placing it in octant IV. As the application evolves, its communication requirements assume greater significance. Moreover, the application exhibits different dynamics and adaptations at different stages during its execution. The pBD-ISP scheme improves communication and data migration metric and is seen to perform better than the SFC and G-MISP+SP partitioners. The ARMaDA adaptive partitioner starts out with the G-MISP+SP or SFC partitioner as specified in the "param" file. This partitioner is better suited for the initial stages of the application. The adaptive policy formulated and used by the ARMaDA framework for RM3D evaluation is listed

in Table 4.

Iteration number	ARMaDA octant	Appropriate partitioner	Actual octant
1-6	7	SFC	IV
7-16	3	pBD-ISP	II
17-26	6	SFC	III
27-36	3	pBD-ISP	II
37-46	6	SFC	III
47-56	3	pBD-ISP	II
57-66	4	SFC	VII
67-84	1	pBD-ISP	VI
84-100	6	SFC	III

Table 4. ARMaDA adaptive policy for RM3D application on 64 processors on "Blue Horizon"

The overheads associated with the ARMaDA framework are low. For the RM2D and RM3D evaluations, the overhead is measured to be 0.415 seconds and 1.85 seconds respectively, and is minimal compared to the overall execution times. For the RM3D evaluation, the default ARMaDA settings used are SAME_REGRID_LIMIT = 4, SKIP_SAME_REGRID = 4, SKIP_SWITCH = 4, LOW_THRESH = 0.6, HIGH_THRESH = 1.4, and metric weights (Cwt, Dwt, Awt) = 1.0.

These results demonstrate that adaptive application-sensitive partitioning employed by the ARMaDA framework improves application performance resulting in reduced execution times. The improvement yielded by the ARMaDA framework for the RM2D application is 4.66%, 11.32%, and 27.88% over pBD-ISP, G-MISP+SP, and SFC partitioners respectively. In the case of RM3D application, the ARMaDA adaptive partitioner provides a speedup of 22.17% over the pBD-ISP partitioner and 38.28% over the SFC scheme.

6 Conclusions

This paper presented the ARMaDA framework for the adaptive application-sensitive partitioning of dynamic structured adaptive mesh refinement applications. The research was motivated by the observation that the choice of the "appropriate" partitioning technique and associated partitioning parameters depends on the nature of the application and its runtime state. The ARMaDA framework establishes mechanisms for characterizing the state of the adaptive application and abstracting the current computational, communication and storage requirements. The adaptive meta-partitioner component of the framework dynamically selects and configures partitioning strategies at runtime. The partitioners constitute a selection from popular software tools such as GrACE and Vampire. The experimental evaluation of the framework was presented using the 2-D Transport AMR, 2-D Vector Wave, and 2-D and 3-D Richtmyer-Meshkov CFD kernels.

The goal of ARMaDA is to realize an adaptive runtime environment that uses current runtime state to meet application requirements, thereby maximizing its efficiency and performance. The overarching motivation is to enable large-scale dynamically adaptive scientific and engineering simulations on distributed, heterogeneous and dynamic execution environments such as the computational "grid".

References

[1] S. Chandra and M. Parashar. An Evaluation of Partitioners for Parallel SAMR Applications. Proceedings of the *Euro-Par 2001*, Springer-Verlag Lecture Notes in Computer Science, Vol. 2150, pp. 171-174, August 2001.

[2] S. Chandra, J. Steensland. M. Parashar and J. Cummings. An Experimental Study of Adaptive Application Sensitive Partitioning Strategies for SAMR Applications. Proceedings of the *2nd Los Alamos Computer Science Institute Symposium* (also Best Research Poster at *Supercomputing Conference 2001*), October 2001.

[3] Z. Lan, V. Taylor and G. Bryan. Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications. Proceedings of the *30th International Conference on Parallel Processing*, Valencia, Spain, 2001.

[4] L. Oliker and R. Biswas. PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes. *Journal of Parallel and Distributed Computing*, 52(2), pp. 150-177, 1998.

[5] M. Parashar, J. Browne and et al. A Common Data Management Infrastructure for Adaptive Algorithms for PDE Solutions. Proceedings of the *Supercomputing Conference*, ACM/IEEE Computer Society, San Jose, CA, November 1997.

[6] K. Schloegel, G. Karypis and V. Kumar. A Unified Algorithm for Load-balancing Adaptive Scientific Simulations. Proceedings of the *International Conference on Supercomputing*, Dallas, TX, November 2000.

[7] J. Steensland. Vampire homepage, http://www.caip.rutgers.edu/~johans/vampire, 2000.

[8] J. Steensland, S. Chandra and M. Parashar. An Application-Centric Characterization of Domain-Based SFC Partitioners for Parallel SAMR. accepted for publication in *IEEE Transactions on Parallel and Distributed Systems*, April 2002.

[9] J. Steensland, S. Chandra, M. Thuné and M. Parashar. Characterization of Domain-based Partitioners for Parallel SAMR Applications. Proceedings of the *IASTED International Conference on Parallel and Distributed Computing and Systems*, Las Vegas, NV, pp. 425–430, November 2000.