A PATCH-BASED PARTITIONER FOR PARALLEL SAMR APPLICATIONS

Henrik Johansson
Dept. of Information Technology, Scientific Computing
Uppsala University
Box 337, S-751 05 Uppsala, Sweden
E-mail: henrik.johansson@it.uu.se

Abbas Vakili
Dept. of Information Technology, Scientific Computing
Uppsala University
Box 337, S-751 05 Uppsala, Sweden
E-mail: abbas.vakili@it.uu.se

Abstract

To achieve good parallel efficiency, applications using structured adaptive mesh refinement (SAMR) need to repeatedly repartition and redistribute the underlying dynamic grid hierarchy. However, no single partitioner works well for all application and computer states.

This paper presents the implementation and evaluation of a patch-based partitioner for SAMR grid hierarchies. The partitioner results in a good and stable load balance, on average 3.1%. Space-filling curves are used to reduce the high communication volumes that are inherent in this type of partitioner.

The partitioner will be a part of the Meta-Partitioner, a partitioning framework that automatically selects, configures and invokes good-performing partitioners for general SAMR applications. Access to a large number of complementing partitioners is essential for the Meta-Partitioner. The presented partitioner will help to significantly decrease run-times for SAMR applications where load balance is the main priority.

Keywords: Load balancing, structured adaptive mesh refinement, partitioning.

1 Introduction

Structured adaptive mesh refinement (SAMR) is widely used to decrease the run-time of simulations based on partial differential equations in domains like computational fluid dynamics [3, 6], numerical relativity [7], astrophysics [4, 14] and hydrodynamics [13]. Good parallel efficiency for large SAMR applications is generally challenging. It requires that the underlying dynamic and adaptive grid hierarchy is repeatedly partitioned for efficient use of the parallel computer's resources; the arithmetical workload must be evenly distributed over the processors, the communication and synchronization between processors must be minimized, and the data migration between two consecutive partitions should be minimized.

In this paper, we present a patch-based partitioner for SAMR grid hierarchies. The partitioner is part of a larger research effort that aims to improve the performance for general SAMR applications executing on general parallel computers. SAMR applications exhibit vastly different partitioning needs depending on the state of the application and the computer system. None of the common partitioning approaches perform well for all application and computer states [8, 19]. To reduce the execution time, the partitioner needs to be dynamically selected and invoked during run-time. We have earlier proposed and implemented the Meta-Partitioner, a framework that selects, configures, and invokes the partitioner that results in the best predicted performance with respect to the current application and computer state [8–10, 20, 21]. To be effective, the Meta-Partitioner requires access to a large number of complementing partitioning algorithms based on all common partitioning approaches.

Our patch-based partitioner is specifically targeted at achieving a good and stable load balance. The results show that the load balance is significantly better than for all other common types of partitioners. By using space-filling curves, the large communication volumes associated with patch-based partitioners are reduced by 17%.

Incorporating the patch-based partitioner in the Meta-Partitioner will help to significantly decrease the run-time of SAMR applications, especially where a low load imbalance is the priority.

2 Partitioning approaches for SAMR grid hierarchies

A simulation using SAMR starts with a coarse and uniform grid. The resolution of the grid conforms to the lowest acceptable accuracy. The grid is then recursively refined in areas where the accuracy is too low, resulting in an adaptive grid hierarchy where grid patches are created, moved, and deleted [3].

Partitioners for SAMR grid hierarchies can be classified as patch-based, domain-based, or hybrid. For *patch-based partitioners*, the distribution decisions are independently made for every refinement level or grid patch [2, 12, 17]. In theory, the patch-based approach results in perfect load balance. In practice, some load imbalance can be expected due to integer divisions and constraints on the patch size. Because overlaid parts of the grid are often assigned to different processors, patch-based partitioners generally results in large communication volumes

Domain-based partitioners partition the physical domain rather than the grid patches [15, 16, 20, 22]. The do-

main is partitioned along with all contained grids on all refinement levels. Because the base grid impose restrictions on the minimum patch size, the load imbalance can often be intractable. The inter-level communication is eliminated as all overlaid parts of the grid reside on the same processor.

By combining the patch-based and the domain-based approach, *hybrid partitioners* try to balance the properties of the partitioning approaches [11, 19, 22]. The hybrid partitioner Nature+Fable (N+F) clusters refinement levels into bi-levels. Each bi-level is then partitioned using the domain-based approach. N+F can a achieve a smaller load imbalance than domain-based algorithms since patches from at most two refinement levels are partitioned together. Because inter-level communication only exist between bi-levels, communication volumes are generally smaller than for patch-based algorithms.

3 Related Work

Several variations of patch-based partitioners have been developed.

An iterative bin-packing approach is taken by Balsara and Norton [2]. Initially, patches much larger than the average patch size are divided into smaller patches. Next, the patches are binned to the processors. The bins are then paired against each other and patches are exchanged between the bins if this reduces the load imbalance. Instead of repeatedly pairing the bins in sequence, a certain offset between the bins is used. This offset brings the bins in contact with each other much quicker, reducing the load imbalance faster than when the bins are paired in sequence. A load balancing iteration ends when all possible offsets have been encountered. Usually, only a few iterations are necessary to reduce the load imbalance to only a few percent. The final load imbalance depends on the ratio of patches to processors because patches are only divided before the load balancing iterations. When the number of patches are large, the load imbalance is generally lower.

The Balsara and Norton approach disregard the communication costs. By only focusing on the load imbalance, the locality both on the same refinement level and between different refinement level can be poor and result in large communication volumes. The partitioner were originally implemented for a shared memory computer, which decreases the impact of large communication volumes.

The default load balancing algorithm in the SAMR framework Chombo was recently changed from recursive bisection to an algorithm using space-filling curves. The new algorithm was found to reduce communication and improve scalability. The previous algorithm frequently resulted in very long and thin patches that required large amount of boundary data from other processors. The newly implemented algorithm seems to have many similarities with our patch-based partitioner. Both partitioners use space-filling curves to order the patches. The linearly ordered patches are then divided into intervals with equal workloads. We assume that the Chombo algorithm only divides the patches that are actually causing a processor to overload, but no details are given. Using the new algorithm, only a small fraction of the patches have neighbors located on other processors. Unfortunately, neither detailed implementation information nor performance data have been presented.

Lan, Taylor and Bryan [12] use an approach that must be categorized as patch-based because grid patches from all refinement levels are partitioned individually and simultaneously. The previous partition is used as a starting point. If the load imbalance is greater than a threshold, they try to move complete patches between the processors. If no patches can be moved without the recipient getting an overload, they start a new phase. Now, a patches causing an overload are split to exactly match the underload on another processor. During the partitioning, the patches are moved or divided without regard to the refinement level. Thus, processors can be assigned patches that exclusively belong to a single refinement level. If the assigned workload is unevenly distributed between the refinement levels, some processors will be probably be idle at any given moment. Processors will have to wait while necessary data from lower refinement levels are computed. When the data is ready, other processors might have finished all of their assigned work. The remedy is to partition one refinement level at a time, distributing it over all processors.

4 The patch-based partitioner

The presented partitioner is designed to be incorporated into the Meta-Partitioner. To avoid time-consuming conversions between differing grid hierarchy representations, we use the same grid representation as in the Meta-Partitioner [9]. The format only handles the physical properties of the grid patches — the solution data is contained in the SAMR frameworks that are connected to the partitioner.

To decrease the communication volumes, neighboring grid patches on the same refinement level should be assigned to the same processor. We use inverse space-filling curves (SFC) to increase the locality [18]. An SFC is a recursive mapping from 1-dimensional space to d-dimensional space i.e., $N^1 \rightarrow N^d$, such that each point in N^1 is mapped to a unique point in N^d. Furthermore, points close together in N^1 will be mapped to points close to each other in N^d. Using an inverse SFC, a d-dimensional grid is transformed to an one-dimensional list, while the locality is maintained. There exist several inverse SFC orderings. We use the common inverse Hilbert ordering for the patch-based partitioner. Instead of constructing an SFC at each repartitioning, we only construct a single SFC at the start of the execution. For the construction, we use a dummy grid with the same shape as the base grid. The size of the dummy grid is always a power of two, either equal to or slightly larger than the unrefined base grid. We limit the size of the dummy grid to reduce the construction time. The constructed SFC is then stored during the whole ex-

ecution of the SAMR application. At repartitioning, each grid patch is initially mapped onto the dummy grid and assigned its corresponding SFC-index. The grid patches on each refinement level are then sorted according to their respectively SFC-indices. During the mapping step we also compute the optimal processor workload for each refinement level.

Next, we proceed to assign the sorted patches, one-by-one, to the processors. Eventually, the assignment of a patch will overload the current processor. If the resulting load imbalance is less than 5%, we disregard the overload and assign the whole patch to the current processor. For a small load imbalance, dividing the patch might not reduce the execution time. Instead, the better load balance can be offset by an increase in communication. The amount of allowed load imbalance was determined through experimentation. When the load imbalance is larger than 5%, we divide the grid patch that causes the overload. To minimize communication, we divide the patch parallel to its shortest side. We assign one part to the current processor and the other part to the next processor. Additionally, we aim for a slight overload on the current processor when we divide a patch. If we try to achieve an optimal workload for the current processor, integer divisions and restrictions on where the patch can be cut frequently results in a slight underload. Excess work will thus start to accumulate. Eventually, the excess workload will turn into a huge overload for the last processor.

Algorithm 1 Assign grids

```
for i=0 to maxLevel do
  while patches left on level do
    if (newProcWorkload <= 1.05*optProcWorkload) OR
    lastProc then
      Assign patch to processor
      if newProcWorkload >= optProcWorkload AND !last-
      Proc then
        procID++;
      end if
    else
      Divide patch along its shortest side
      Assign first part to processor
      Add second part to patchList
      procID++
    end if
  end while
end for
```

5 Experimental setup

We evaluate the patch-based partitioner using a suite of four real-world applications from the Virtual Test Facility (VTF). VTF, developed at the California Institute of Technology, is a software environment for coupling solvers for compressible computational fluid dynamics with solvers for computational solid dynamics [6].

Ramp simulates the reflection of a planar Mach 10 shock wave striking a 30 degree wedge. A complicated shock reflection occurs when the shock wave hits the sloping wall. The initial grid size is 480x120 grid points and the application uses three levels of refinement with refinement factors $\{2,2,4\}$. *ShockTurb* treats the interaction of two contacting gases with different densities. When the gases are subject to a shock wave, the interface between them becomes unstable and the result is called a Richtmyer-Meshkov instability. The initial grid size is 240x120 grid points and and the application uses three levels of refinement with a constant refinement factor of two. *ConvShock* simulates a Richtmyer-Meshkov instability in a spherical setting. The gaseous interface is spherical and sinusoidal in shape.The interface is disturbed by a Mach 5 spherical and converging shock wave. The initial grid size is 200x200 grid points and the application uses four levels of refinement with refinement factors $\{2,2,4,2\}$. In the *Spheres* application, a constant Mach 10 flow passes over two spheres placed inside the computational domain. The flow results in steady bow shocks over the spheres. The initial grid size is 200x160 grid points and the application uses three levels of refinement with a constant refinement factor of two.

Application trace files, containing the complete grid hierarchies for each time step, were partitioned for 16 processors. The partitioned grid hierarchies were used as input to a SAMR simulator that mimics the execution of the common Berger-Colella SAMR algorithm [3]. The simulator computes performance metrics like load imbalance, number of patches, and communication volumes [5].

The result from the patch-based partitioner is compared to the results from a domain-based and a hybrid partitioner. The domain-based partitioner was taken from the SAMR framework AMROC [1]. As the hybrid partitioner, we use the partitioning framework Nature+Fable [19]. It is possible to configure Nature+Fable using a large parameter space. In this evaluation, we used around 800 parameter settings. For each combination of time step and metric, we present the result from the configuration that resulted in the best performance. Thus, the results for different metrics and/or time steps probably origin from different hybrid partitioning algorithms. To examine the impact of the SFC, we also present results where the SFC is disabled.

6 Results

For the evaluation, we use a number of performance metrics. We define the load imbalance as the highest workload assigned to a single processor divided by the optimal workload. For each of three metrics for communication, we present the highest the number of communicated grid points for a processor. We also include the maximum number of patches assigned to a single processor.

Load Imbalance For all applications, the patch-based partitioner produced a stable and much lower load imbalance as compared to the domain-based and the hybrid partitioner. The patch-based partitioner resulted in an average

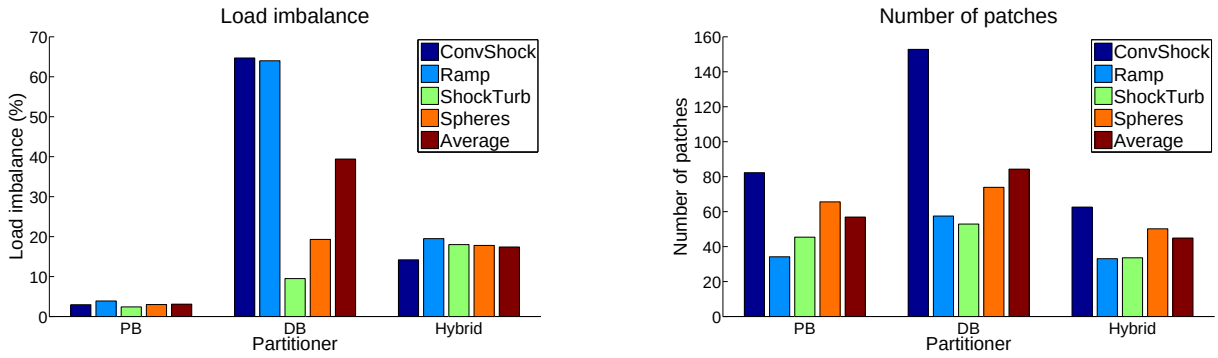

Figure 1. Results for the load imbalance and the number of patches.

load imbalance of only 3.1%, while the domain-based and hybrid partitioners resulted in load imbalances of 39.4% respectively 17.4%.

The patch-based partitioner has a great flexibility on where grid patches can be divided. The domain-based partitioner partitions all refinement levels at once, imposing great restrictions on where patches can be cut. The restrictions result in a much higher load imbalance for the domain-based partitioner. The hybrid partitioner also uses domain-based partitioning techniques, but on only two refinement levels at time. Consequently, the hybrid partitioner results in a load imbalance that is larger than for the patch-based partitioner but lower than for the domain-based partitioner.

At every time step, the load imbalance for the patch-based partitioner is less than five percent. As described in Section 4, we allow a load imbalance of up to five percent to avoid that excess workload are accumulated on the last processor. This mechanism seems to work as intended.

Number of grid patches The hybrid partitioner resulted in the smallest number of patches, followed by the patch-based partitioner and the domain-based partitioner. On average, the patch-based partitioner produced 32.5% fewer patches than the domain-based partitioner, but 21.1% more patches than the hybrid partitioner.

On each refinement level, the patch-based partitioner only divides the patches that actually cause a processor to become overloaded. The hybrid partitioner partitions two refinement levels at once. Depending on the layout of the patches, this strategy often results in fewer patches at the cost of a slightly higher load imbalance [19]. The domain-based partitioner divides all refinement levels with a single cut without checking the sizes of the created patches. This can potentially resulting in many small patches that may increase the communication volumes.

Intra-level communication Both the hybrid and the domain-based partitioner result in a lower amount of intra-level communication than the patch-based partitioner. The hybrid partitioner produced 50.4% less communication and the domain-based 28.1% less communication than the patch-based partitioner.

This result was expected, because the patch-based partitioner calculates the SFC-indices using a dummy grid that generally has a lower resolution than the refined grid hierarchy. Consequently, the patch-based partitioner can assign the same SFC-index to several patches, potentially degrading the locality. For the domain-based partitioner, each grid patch is assigned a unique SFC-index. The bi-level approach used in the hybrid partitioner also assigns unique SFC-indices to the patches. The amount of intra-level communication for the hybrid partitioner is probably further decreased by the small number of grid patches.

Inter-level communication We only present the inter-level communication for the patch-based and the hybrid partitioners. For the domain-based partitioner, overlaid grid patches are always assigned to the same processor, eliminating the inter-level communication.

The amount of inter-level communication for both partitioners is substantially larger than the intra-level communication, often by a factor ten or more. For the intra-level communication, only data located at processor boundaries need to be exchanged. Communication between refinement levels generally involves much larger amounts of data — large portions, not just the boundaries, of a grid patch often has to be exchanged between processors.

To reduce the amount of inter-level communication, overlapping areas of the grid hierarchy should be assigned to the same processor. The hybrid partitioner use the domain-based approach on two refinement levels at a time, eliminating the communication between these levels. The patch-based partitioner only considers one refinement level at a time, resulting in an average inter-level communication volume that is 8.3 times larger than for the hybrid partitioner.

Total communication The total communication is the sum of intra- and inter-level communication. Because the intra-level communication is much lower than inter-level communication, the total communication is similar to the volume of inter-level communication for both he patch-based and hybrid partitioner.

The domain-based partitioner has a low amount of total communication since the inter-level communication is

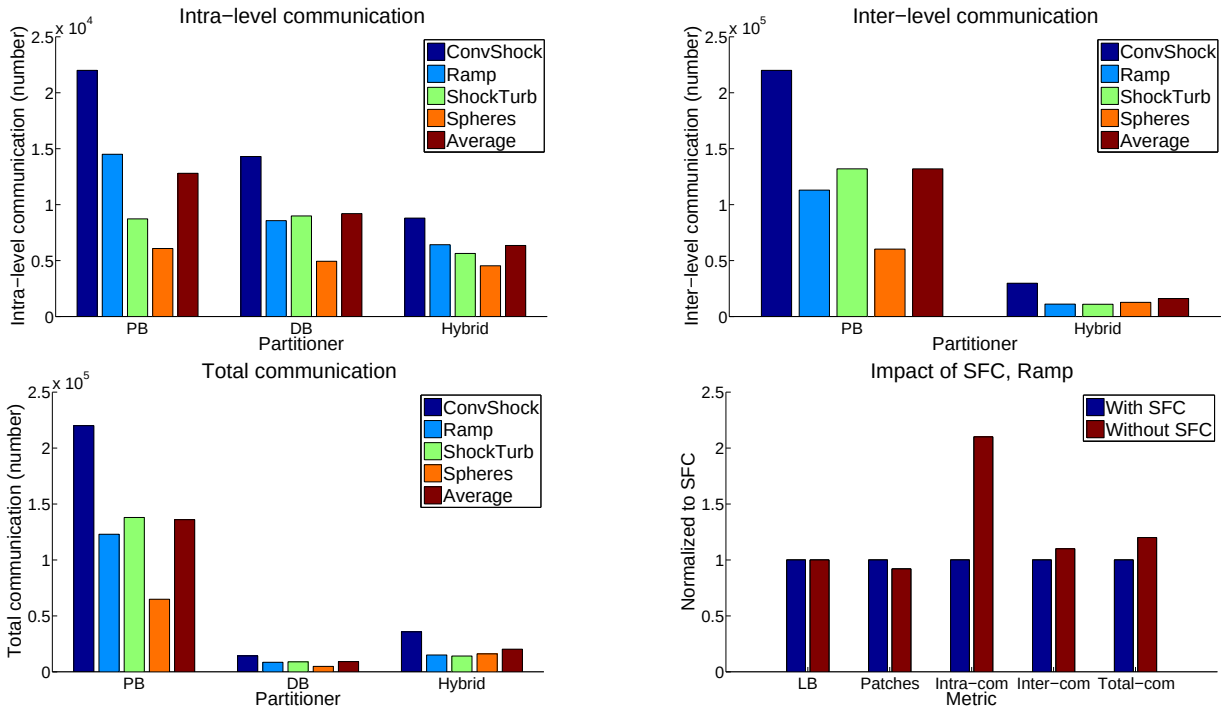

Figure 2. Results for the communication and the impact of space-filling curves.

eliminated. The total communication for the hybrid partitioner is on average 4.5 times larger than the total communication for the domain-based partitioner. For the patch-based partitioner, the difference is a factor 6.7 in favor of the domain-based partitioner.

Impact of SFC To increase locality and reduce the communication volumes, the patch-based partitioner uses an inverse space-filling curve to order the grid patches. In this section we examine the impact of the SFC ordering. We only present the results for the Ramp application since the general behavior was similar for all applications.

On average, the use of SFC decreases the amount of intra-level communication with 53.1%. The largest observed decrease for a single time step was 66.7%. For the inter-level communication, we see a smaller decrease — in average 8.9%. The decrease is small because the ordering only considers patches on the same refinement level. However, the SFC always follows the same path for every refinement level. Thus, the probability that overlaid patches are assigned to the same processor is increased. The cumulative effect is a decrease of 16,9% in the total communication when the SFC is enable. The impact on the two other metrics, load imbalance and number of blocks, was minor and insignificant.

7 Conclusions and further work

In this paper we have presented the implementation and evaluation of a patch-based partitioner for SAMR grid hierarchies. The partitioner achieves a good and stable load balance for all experiments. The load imbalance is significantly lower than for all other common types of load balancing algorithms. The inherent drawback of patch-based partitioners is high communication volumes. By using space-filling curves to preserve locality, we are able to reduce the total communication with 17 percent. To reduce the communication further, overlaid patches must to a larger extent be assigned to the same processor. This will require a significant research effort.

The implemented partitioner will play an important part in the Meta-Partitioner, a partitioning framework that automatically selects, configures and invokes good-performing partitioners for SAMR applications. To be effective, the Meta-Partitioner need to contain complementing partitioners from as many partitioning approaches as possible. The patch-based partitioner presented in this paper will be especially valuable in situations where the load balance is the main priority.

8 Acknowledgments

The authors thank Ralf Deiterding, Oak Ridge National Laboratory, for providing the application trace files and the performance data for the domain-based partitioner. The authors also thank Johan Steensland and Jaideep Ray, Sandia National Laboratories, for scientific collaboration.

References

[1] AMROC - Blockstructured adaptive mesh refinement in object-oriented C++. http://amroc.sourceforge.net/index.htm, June 2008.

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

[3] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989.

[4] Greg L. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, pages 46–53, Mar-Apr 1999.

[5] Sumir Chandra, Mausumi Shee, and Manish Parashar. A simulation framework for evaluating the runtime characteristics of structured adaptive mesh refinement applications. Technical Report TR-275, Center for Advanced Information Processing, Rutgers University, 2004.

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

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

[8] Henrik Johansson. *Performance Characterization and Evaluation of Parallel PDE Solvers*. Licentiate thesis, Department of Information Technology, Uppsala University, November 2006.

[9] Henrik Johansson. Design and implementation of a dynamic and adaptive Meta-Partitioner for parallel SAMR grid hierarchies. Report 2008-017, Department of Information Technology, Uppsala University, Sweden, 2008. Available at http://www.it.uu.se/research/publications/reports/2008-017.

[10] Henrik Johansson and Johan Steensland. A performance characterization of load balancing algorithms for parallel SAMR applications. Report 2006-047, Department of Information Technology, Uppsala University, Sweden, 2006. Available at http://www.it.uu.se/research/reports/2006-047/.

[11] Zhiling Lan, Valerie E. Taylor, and Greg Bryan. Dynamic load balancing of SAMR applications on distributed systems. In *Proceedings of 30th International Conference on Parallel Processing*, 2001.

[12] Zhiling Lan, Valerie E. Taylor, and Greg Bryan. A novel dynamic load balancing scheme for parallel systems. *Journal of Parallel and Distributed Computing*, 62:1763–1781, 2002.

[13] Charles L. Mader and Michael L. Gittings. Modeling the 1958 Lituya Bay mega-tsunami, II. *Science of Tsunami Hazards*, 20(5):241–250, 2002.

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

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

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

[17] Jarmo Rantakokko. Partitioning strategies for structured multiblock grids. *Parallel Computing*, 26(12):1661–1680, 2000.

[18] Hans Sagan. *Space-filling curves*. Springer Verlag, 1994.

[19] Johan Steensland. *Efficent Partitioning of Dynamic Structured Grid Hierarchies*. PhD thesis, Department of Scientific Computing, Information Technology, Uppsala University, Oct. 2002.

[20] 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*, 13(12):1275–1289, Dec 2002.

[21] Johan Steensland, Michael Thun, Sumir Chandra, and Manish Parashar. Characterization of domain-based partitioners for parallel samr applications. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing Systems*, pages 425–430, 2000.

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