Dynamic Structured Partitioning for Parallel Scientific Applications with Pointwise Varying Workloads

Sumir Chandra[1], Manish Parashar[1], and Jaideep Ray[2]

[1]ECE Dept./CAIP Center
Rutgers University
Piscataway, NJ 08854, USA
{sumir, parashar}@caip.rutgers.edu

[2]Advanced Software R&D
Sandia National Laboratories
Livermore, CA 94551, USA
jairay@ca.sandia.gov

Abstract

Parallel implementations of scientific applications involving the simulation of reactive flow on structured grids are challenging, since the underlying phenomena include transport processes with uniform computational loads as well as reactive processes having pointwise varying workloads. As a result, traditional parallelization approaches that assume homogeneous loads are not suitable for these simulations. This paper presents "Dispatch", a dynamic structured partitioning strategy that has been applied to parallel uniform and adaptive formulations of simulations with computational heterogeneity. Dispatch maintains the computational weights associated with pointwise processes in a distributed manner, computes the local workloads and partitioning thresholds, and performs in-situ locality-preserving load balancing. The experimental evaluation of Dispatch using an illustrative 2-D reactive-diffusion kernel demonstrates improvement in load distribution and overall application performance.

Keywords: *Dynamic load balancing, structured grids, pointwise processes, computational heterogeneity.*

1. Introduction

Simulations of complex physical phenomena, modeled by systems of partial differential equations (PDEs), play an important role in science and engineering. Dynamic structured adaptive mesh refinement (SAMR) [1] methods have emerged as attractive formulations of these simulations on structured grids. SAMR techniques have been used to solve complex systems of PDEs that exhibit localized features in various application domains including computational fluid dynamics, numerical relativity, astrophysics, combustion simulation, subsurface modeling and oil reservoir simulation.

Structured grids usually employ regular data structures that are easier to partition and lead to regular access and communication patterns. Consequently, structured formulations of parallel scientific simulations can result in relatively simpler, and more efficient and scalable implementations. Parallelization of these applications typically consists of partitioning the structured grid into uniform blocks and allowing processors to compute on these blocks in parallel. Large-scale parallel implementations for structured uniform grids have been widely reported. The parallelization of a SAMR application, however, tends to be more complex due to the dynamic and heterogeneous nature of the adaptive grid hierarchy [2]. Several existing structured grid infrastructures, such as GrACE [12], SAMRAI [7], Chombo [4], and Paramesh [8], address SAMR partitioning challenges and support parallel adaptive implementations. Furthermore, these frameworks typically assume that the computational effort at each grid point is the same and the workload at any level on the structured grid is uniformly distributed.

However, there exists a class of scientific applications involving reactive flows, such as the simulation of hydrocarbon flames with detailed chemistry [13], where the physical models include transport/structured processes with uniform computational loads as well as reactive/pointwise processes having varying workloads. In these simulations, the solution of pointwise processes at each iteration requires a different number of subcycles to complete the computation at each grid point within a single global timestep. As a result, the computational load varies at different grid points and is only known locally, and at runtime. Therefore, traditional parallelization approaches are not suitable for

1-4244-0054-6/06/$20.00

these simulations, and their parallel/distributed implementations present significant partitioning and runtime challenges. Research efforts addressing related issues include the load balancing mechanism by Moon et al. [11] with no spatial couplings between structured and pointwise processes for simulations of hydrocarbon flames, and the measurement-based dynamic load balancing in Adaptive MPI [6] using virtual processes and object migration for benchmark applications.

This paper presents the *Dispatch* dynamic structured partitioning strategy that has been applied to parallel uniform and adaptive formulations of scientific applications with computational heterogeneity. *Dispatch* maintains the computational weights associated with pointwise processes in a distributed manner. The *Dispatch* partitioner is invoked at periodic intervals during application execution and uses in-situ global load balancing to determine workload transfers and data remapping among processors. The *Dispatch* strategy consists of the following four steps: (1) it maps the computational weights onto the current grid structure (represented by distributed grid functions) that may be organized uniformly on a single level or as a multi-level SAMR hierarchy with dynamic refinement regions; (2) it generates intermediate workloads using interpolation for new refinements in adaptive meshes that correspond to pointwise processes; (3) it computes the local workloads (in parallel) and partitioning thresholds to determine processor allocations proportional to the computational weights; and (4) it performs workload and data redistribution that preserves application locality. The experimental evaluation of *Dispatch* is performed using an illustrative 2-D reactive-diffusion (R-D) kernel and demonstrates improvement in load distribution and overall application performance.

The rest of the paper is organized as follows. Section 2 outlines parallel formulations of simulations on structured grids. Section 3 illustrates the partitioning challenges for applications with heterogeneous workloads using the R-D kernel, and presents related work. Section 4 details the design and operation of the *Dispatch* strategy. Section 5 describes the experimental evaluation of *Dispatch* for uniform and adaptive R-D implementations. Section 6 presents concluding remarks.

2. Parallel Formulations of Simulations on Structured Grids

2.1. Unigrid Formulations

Structured unigrid [10] methods discretize the application problem space onto a single, regular Cartesian grid with uniform resolution. The unknowns of the PDE are then approximated numerically at each discrete grid point. The resolution of the grid (or grid spacing) determines the local and global error of the approximation, and is typically dictated by the features of the solution that need to be resolved. The resolution of the grid also defines the computational costs and storage requirements of the formulation. Parallel unigrid implementations uniformly decompose the structured grid, assuming homogeneous workloads, to balance the load across processors. Each processor then performs computations on its local patches and periodically synchronizes its boundaries with neighboring patches.

2.2. SAMR Formulations

Unigrid formulations can be wasteful for applications with localized features that require higher grid resolutions only in a small portion of the computational domain. In such cases, SAMR techniques yield highly advantageous ratios for cost/accuracy by dynamically refining the domain only in regions with large local solution error [1]. SAMR methods start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain requiring additional resolution are tagged and finer grids are overlaid on these tagged regions of the coarse grid. Refinement proceeds recursively so that regions on the finer grid requiring more resolution are similarly tagged and even finer grids are overlaid on these regions, thus resulting in a structured adaptive grid hierarchy.

The grid hierarchy in the SAMR formulation is refined both in space and time. Refinements in space create finer level grids, which have more grid points than their parents, and hence greater computational loads. Refinements in time mean that finer grids take smaller timesteps, and hence have to be advanced and synchronized more often. This results in a space-time heterogeneity in the SAMR adaptive grid hierarchy [2]. Note that even though finer-resolution grids have more grid points and consequently more work than coarser grids, the load associated with each grid point throughout the computational domain is the same since the formulation assumes homogeneous workloads.

Parallel implementations of SAMR applications typically partition the dynamic grid hierarchy across participating processors, with each processor operating on its portions of the grid in parallel. Processors perform computation, boundary synchronization, and interlevel communication recursively at each level of the grid hierarchy. Dynamic partitioning and regridding/-redistribution occurs at regular intervals and results in refined regions being created, moved, and/or deleted.

3. Parallel Formulations of Simulations with Heterogeneous Workloads

3.1. Partitioning Challenges for Pointwise Varying Workloads

Parallel structured implementations of PDE-based simulations typically assume a uniform workload per grid point, and use numerical schemes that require all points to march forward, together in time, in lock-step. Such an approach is exact since it preserves the spatial coupling in a straightforward manner. However, there exist certain classes of problems, such as reactive flows, which have physical point processes that are coupled to other similar point processes through a second process. Reaction/chemistry is such a point process – its models (and operators in the evolution equations for reactive flows) do not contain any spatial derivatives. Reactive processes at a point in space affect others around them through convective and diffusive processes, which are separate physical processes and usually operate at different timescales. Thus, over a time period that is far smaller than the transport (convection and diffusion) timescale, the reactive processes can be considered as approximately decoupled. This approximation is exploited in operator-split [9] integration methods.

Operator-split methods are used in PDEs where the physical processes can be approximately decoupled over a global timestep Δt_g. Consequently, the physical processes can be advanced in time over Δt_g using separate integrators since they do not have to be marched in lock-step, and are usually chained in a certain sequence for accuracy reasons. If one of these physical processes happens to be a purely point process, viz. reaction, then separate (systems of) ordinary differential equations (ODEs) for different points in space are obtained. While these ODEs are advanced up to Δt_g for all points, different points (decoupled point processes) can adopt various independent paths to reach there. Thus, points in the domain with little reaction take a few large timesteps to reach Δt_g, while points with significant reactive processes take miniscule timesteps in order to time-resolve the fastest reactive processes. Although such an approach is efficient as the non-reactive regions do not necessarily march in lock-step with explosive regions, it results in a highly uneven distribution of computational load as a function of space. Any domain partitioner that ignores this uneven load distribution often incurs a stiff load imbalance penalty.

Since the reactive processes are approximately decoupled in space (over Δt_g) and there are no spatial terms in the reaction operator, a conceptually simple solution exists. The grid points are distributed arbitrarily across all processors as long as the loads are equalized. Such a solution has no spatial couplings, and hence does not consider communication costs or preserve the connections between a point and its neighbors. As a result, this approach incurs significant communication costs as the data is redistributed at every global timestep. In combusting flows, where one strives to capture subtle effects of the simulation by preserving as many chemical species as possible (leading to 50-100 variables per grid point), this communication cost can be prohibitive. This motivates the requirement to calculate the reactive processes *in situ*, and achieve load balance by a prudent domain decomposition that incorporates the uneven nature of load distribution.

3.2. Related Work in Partitioning Heterogeneous Workloads

In [11], Moon et al. evaluate the performance of simulations of hydrocarbon flames using the Multiblock PARTI and CHAOS runtime libraries. The physical processes are classified as structured (e.g., heat conduction) or pointwise (e.g., radiation, chemistry, etc.) processes. As there is no spatial coupling, the block-partitioned data is redistributed across processors to balance the load of pointwise processes, and is then moved back into the original locations for the next structured process. This results in a substantial amount of communication as the application redistributes data at every timestep. Heuristic schemes partly alleviate redistribution costs by generating a load balancing plan and reducing the communication volume. The *Dispatch* scheme presented in this paper accounts for heterogeneous workloads in the current distribution and performs in-situ global partitioning, without additional data migration for load balancing.

Adaptive MPI (AMPI) [6] extends MPI to support processor virtualization and provides dynamic measurement-based load balancing strategies for automatic load redistribution, based on object/thread migration in CHARM++. AMPI is evaluated using an artificial benchmark involving non-uniform 2-D stencil calculations, where the load on 1/16 of the processors is much heavier than on the other 15/16 processors. Unlike AMPI, *Dispatch* addresses adaptive meshing, domain decomposition, and runtime support for scientific applications with computational heterogeneity.

3.3. Reactive-Diffusion (R-D) Application

Combustion applications modeling the properties of hydrocarbon flames [13] are highly complex. The physical processes in such simulations interact in a strongly

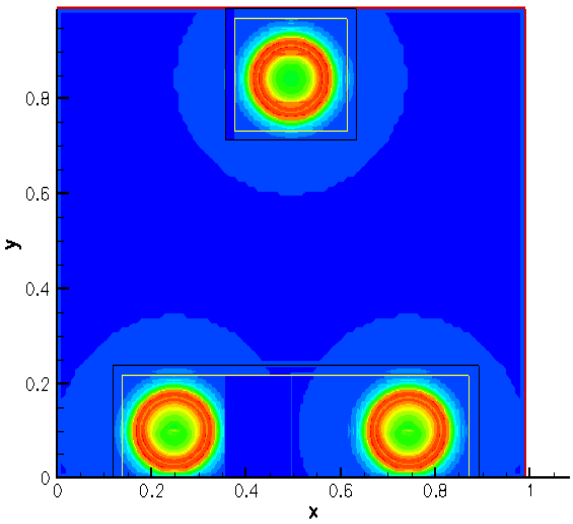

Figure 1. 2-D snapshot of R-D kernel's temperature field with 3 hot-spots at time t=100.

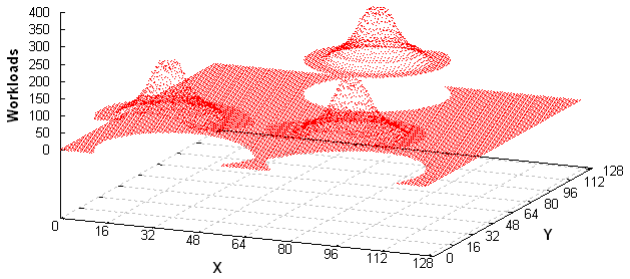

Figure 2. Distribution of heterogeneous loads for R-D kernel on a 128*128 structured grid.

non-linear fashion and accurate solutions can only be obtained using highly detailed models that include complex chemistry and transport processes [11]. The transport (or structured) processes have uniform computational workloads while the chemical/reactive (or pointwise) processes require different amounts of computation (i.e., "weights") at each point.

A model problem approximating the ignition of a CH_4-Air mixture is used as an illustrative example to evaluate the *Dispatch* partitioning strategy presented in this paper, and is referred to as the *reactive-diffusion* (R-D) application or kernel. Figure 1 is a 2-D snapshot of the R-D kernel that illustrates application dynamics after 100 timesteps during the ignition of a CH_4-Air mixture in a non-uniform temperature field with 3 "hot-spots". The application exhibits high dynamism, space-time heterogeneity and varying computational workloads, and is representative of the class of simulations targeted by this research. Figure 2 shows a sample distribution of the heterogeneous computational workloads associated with pointwise processes for the R-D kernel on a 128*128 structured grid. The reactive processes near the flame fronts have high computational requirements that correspond to large values of workloads at the 3 hot-spots, while the diffusive processes have uniform loads with a value of 1.

Briefly, the R-D kernel solves an equation of the form

$$\frac{\partial \mathbf{\Phi}}{\partial t} = \nabla^2 \mathbf{\Phi} + R(\mathbf{\Phi}) \qquad (1)$$

where $\mathbf{\Phi}$ is a vector consisting of the temperature and the mass fraction of 26 chemical species at a given point in space. $R(\mathbf{\Phi})$ models the production of heat and chemical species by 92 reversible chemical reactions [5]. ∇^2 is approximated using second-order central differences. Eq. 1 is evolved in the following manner:

1. Over a timestep of $\Delta t_g/2$, we advance $\mathbf{\Phi}^n$ (solution at timestep n) using $\mathbf{\Phi}_t = \nabla^2 \mathbf{\Phi}$ to $\mathbf{\Phi}'$ with Heun's method (second order Runge-Kutta scheme). For this step, the integration is done either on one level for a unigrid implementation or in a recursive manner for all levels in case of SAMR, so that the CFL condition is preserved on each patch of the SAMR hierarchy for the Berger-Oliger formulation [1].

2. Using $\mathbf{\Phi}'$ as initial condition, we solve $\mathbf{\Phi}_t = R(\mathbf{\Phi})$ over Δt_g to get $\mathbf{\Phi}''$. Since there are no spatial coupling terms, this system is solved on a point-by-point basis. At certain points, especially near flame fronts and ignition points, this ODE system exhibits very fast kinetics and has to be advanced using small timesteps (for accuracy reasons). This is done using BDF3 from the CVODE [3] package. This step does not require recursive integration in the case of SAMR, and accounts for the heterogeneity in the application workloads.

3. Using $\mathbf{\Phi}''$ as initial condition, we solve $\mathbf{\Phi}_t = \nabla^2 \mathbf{\Phi}$ over $\Delta t_g/2$ to get $\mathbf{\Phi}^{n+1}$, exactly as in Step 1.

4. Dynamic Partitioning for Applications with Computational Heterogeneity

This section presents *Dispatch*, a dynamic structured partitioning strategy for scientific applications with pointwise varying workloads. *Dispatch* has been integrated with the GrACE [12] computational framework and enables parallel uniform and adaptive simulations. *Dispatch* augments the structured grid formulations outlined in Section 2 by combining an inverse

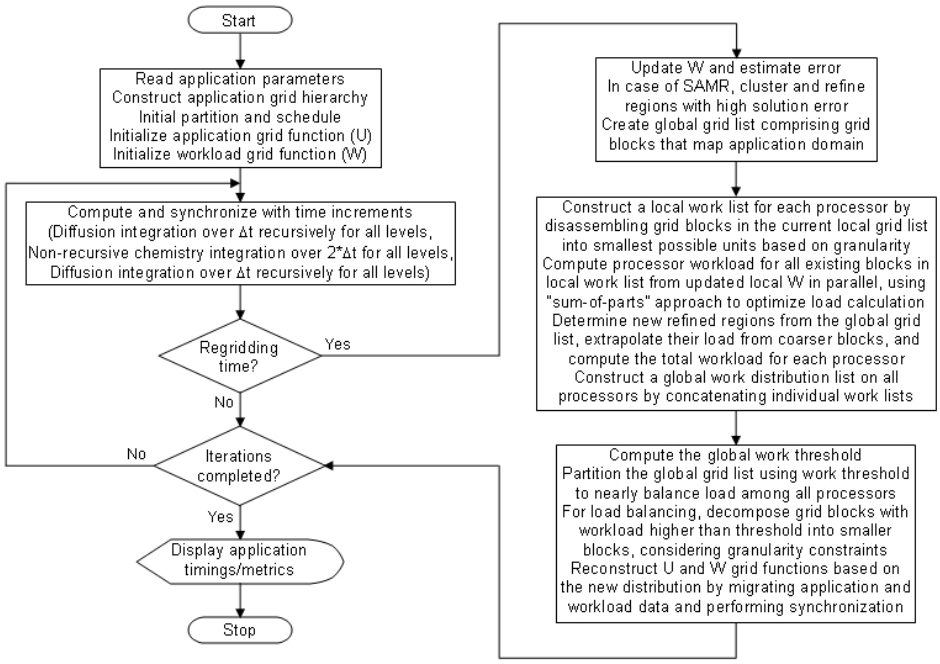

Figure 3. R-D kernel execution illustrates the Dispatch scheme for heterogeneous workloads.

space-filling curve based partitioner (ISP) [12] with in-situ weighted load balancing using global thresholds. The reactive-diffusion (R-D) kernel, presented in Section 3, is used to illustrate *Dispatch*. The parallel execution of the R-D application is shown in Figure 3 and described in the next section.

4.1. Parallel R-D Application Execution

The execution of the R-D application consists of three major phases: (i) initialization, (ii) computation and synchronization at each timestep, and (iii) periodic load balancing followed by redistribution. The structured grid domain/hierarchy is constructed at the start of the simulation based on input parameters. The initial partitioning is a simple geometric decomposition of the application domain among the available processors. The application grid function (U) and workload grid function (W) are then initialized. Grid functions are distributed entities that represent application variables denoting physical entities (e.g., pressure, temperature, density, etc.), and use the grid hierarchy as a template to define their structure and distribution.

The computation component consists of two diffusion integration methods over an application timestep Δt. These two integration methods are recursively invoked for each level of the (single level or adaptive) grid hierarchy and are separated by a non-recursive chemistry integration routine over $2*\Delta t$ for all levels. This is followed by boundary updates and timestepping. During the redistribution phase, computational weights in W corresponding to existing grid points are first updated. In case of SAMR, a truncation error estimate is used to identify regions requiring additional resolution, which are then clustered and refined. As described in Section 4.2, a global grid list mapping the entire application domain is created and the *Dispatch* strategy is invoked to dynamically partition the grid. Since unigrid can be viewed as a special case of SAMR with only one level, the description of *Dispatch* focuses on the general SAMR case.

4.2. Parallel Workload Computation

As depicted in Figure 4, traditional inverse space-filling curve based partitioners (ISP) [12] preserve application locality by indexing the grid blocks that map the structured grid domain in the order of traversal of the Hilbert space-filling curve (SFC) [14] to form a one-dimensional global grid list. The global grid list consists of simple or composite grid blocks representing portions of the application domain. Simple grid blocks are strictly base grid regions while composite grid blocks contain regions that span multiple levels of refinement in the grid hierarchy. Decomposing a grid block entails a geometrical bisection of the block along all axes as long as the minimum block dimension (granularity) constraints are satisfied. A grid block that at-

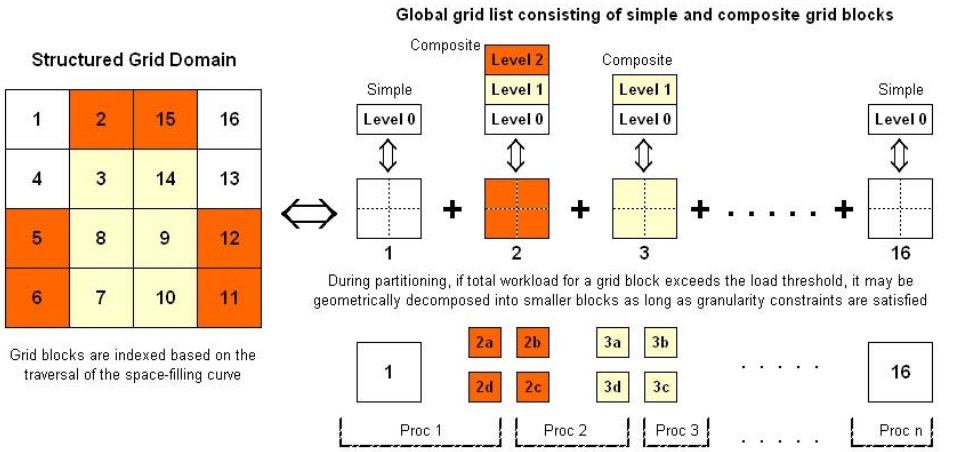

Figure 4. The structured grid domain is mapped to a global grid list for load balancing in Dispatch.

tains minimum block dimension is called a "grain". As an example, a 2-D grid block, if divisible, will decompose into 4 smaller blocks. Similarly, a 64*64 size grid block, when recursively decomposed, will result in 256 grains for a minimum application granularity of 4.

Each simple/composite grid block in the global grid list is assigned a cost corresponding to its computational load, which is determined by the load at the grid points contained in the block at each level and the level of the block in the SAMR grid hierarchy. Since the load per grid point is uniform for homogeneous simulations, the total computational work is proportional to the number of grid points and is relatively easy to calculate. However, in the heterogeneous case such as the R-D kernel, the load of a grid block at a level is obtained as the sum of the computational weights in the workload grid function (W) corresponding to the grid points in the grid block at that level.

Since the computational weights for existing grid blocks in W as well as the global grid list may have been updated during regridding, the *Dispatch* strategy first compares the current and previous global grid lists to identify existing grid blocks and new refinement regions. Each processor then operates on its local grid list (portions of the global grid list owned by it) in parallel, and constructs a local work list comprised of grains obtained by disassembling the simple/composite grid blocks in the local grid list. This decomposition of the local grid list is performed to speed up load calculation using the "sum-of-parts" approach (overall load of the grid block is simply the sum of the loads of all its grains) and to fine-tune the load balancing algorithm (decomposing a grid block during partitioning does not involve recalculation of loads of the constituent parts). The owner computes the updated workload for each grain in the local work list from its corresponding local W, and determines the workloads for existing grid blocks in the list. When a grain contains a newly refined level, the load at each grid point for that level is interpolated as the average workload of its parent (the immediate coarser level). Each processor stores the loads for all grains in its local work list and computes the total processor workload, in parallel. All processors then collectively construct a global work distribution list by concatenating individual local work lists.

4.3. Global Load Balancing

The global work threshold is locally computed at each processor from the global work distribution list. *Dispatch* performs domain decomposition by appropriately partitioning the global grid list based on the global work threshold so that the total computational load on each processor is approximately balanced. Processor allocation is based on a linear assignment of loads in the order of occurrence in the global grid list. This is done to preserve application locality which, in turn, reduces communication and data migration overheads during redistribution. If the workload for a grid block in the global grid list exceeds the processor threshold, the block is decomposed (possibly recursively) and replaced by smaller grid blocks whose loads are already known. If the load is still high, the block is assigned to the next available processor and the processor work thresholds are updated. The load imbalance generated during this phase is due to application granularity and aspect ratio constraints for each grid block that need to be satisfied.

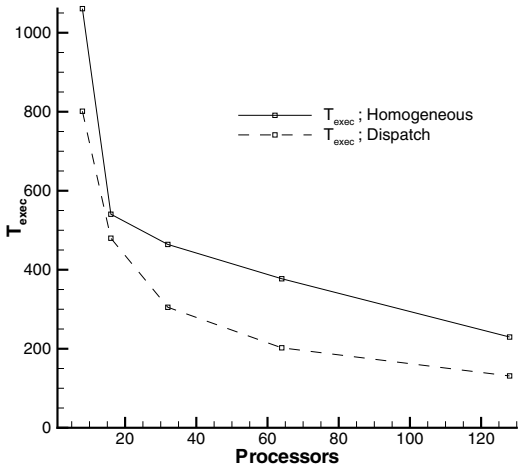

Figure 5. Execution time for 200 timesteps of a simulation on a 256×256 uniform grid plotted as a function of number of executing processors. All times are in seconds.

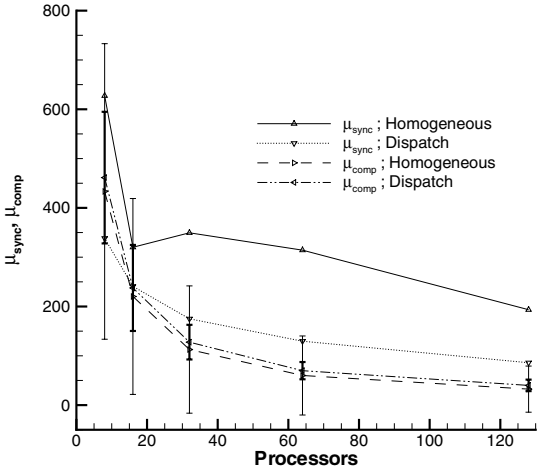

Figure 6. Comparison of compute (μ_{comp}) and synchronization (μ_{sync}) times averaged across all processors. Error bars (dark: *Dispatch*; light: *Homogeneous*) are standard deviations of the compute times. All times are in seconds.

The application (U) and workload (W) grid functions are reconstructed after the dynamic partitioning phase based on the new distribution. This step involves data migration, communication, and synchronization among participating processors, which updates the structure of the grid hierarchy. The new distribution is used in subsequent computation stages until the next regridding stage.

5. Experimental Evaluation

The experimental evaluation of the *Dispatch* strategy is performed using unigrid and SAMR implementations of the 2-D reactive-diffusion (R-D) kernel. The evaluation is performed on the IBM SP4 "DataStar" [15] at the San Diego Supercomputer Center (SDSC). Datastar is SDSC's largest IBM terascale machine that is especially suited for data intensive computations, and has 272 (8-way) P655+ compute nodes with 16-32 GB of memory. The experiments consist of comparing the performance of the *Dispatch* scheme and the default GrACE partitioner (*Homogeneous*) by measuring overall application execution time, load imbalance, synchronization time, and redistribution overheads. The *Homogeneous* strategy assumes that all grid points have the same workload requirements, and hence does not consider computational heterogeneity during load balancing.

5.1. Unigrid Evaluation

The unigrid (1-level) evaluation for the R-D application is performed using *Homogeneous* and *Dispatch* strategies on 8-128 processors on DataStar using 2-D application base grids with resolutions of 256*256 and 512*512. The application executes for 200 iterations, with all other application-specific parameters kept unchanged. Figure 5 plots the total execution time $\mathrm{T_{exec}}$ for *Homogeneous* and *Dispatch* schemes. The *Dispatch* scheme improves overall application execution time by 11.23% for 16 processors up to 46.34% for 64 processors. To further analyze load distribution and application runtime behavior, Figure 6 plots the average (across all processors) compute (μ_{comp}) and synchronization (μ_{sync}) times for 8-128 processor runs. The standard deviation σ_{comp} in compute time is plotted as error bars. The average compute times are roughly similar for both strategies while the *Dispatch* scheme achieves smaller average synchronization times than the *Homogeneous* scheme. *Dispatch* considers the weights of pointwise processes while performing load balancing and achieves a consistently smaller σ_{comp}. This leads to reduced synchronization times (since processors finish computation closer together in time) and ultimately improved execution times, as compared to the *Homogeneous* strategy.

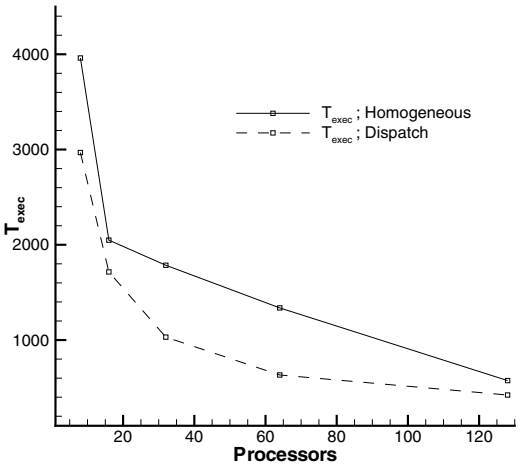

Figure 7. Execution time for 200 timesteps of a simulation on a 512×512 uniform grid plotted as a function of number of executing processors. All times are in seconds.

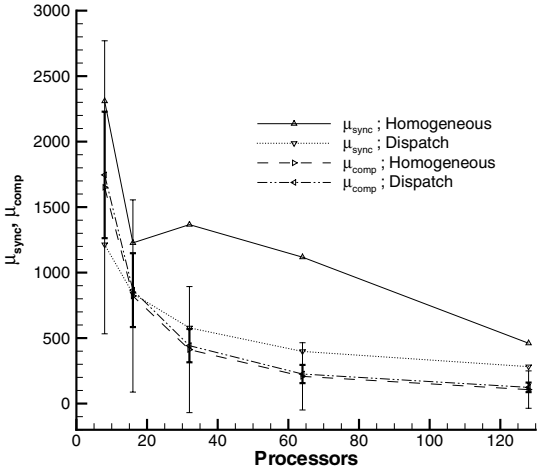

Figure 8. Comparison of compute (μ_{comp}) and synchronization (μ_{sync}) times averaged across all processors. Error bars (dark: *Dispatch*; light: *Homogeneous*) are standard deviations of the compute times. All times are in seconds.

The *Homogeneous* scheme does not repartition the base grid beyond the initial decomposition since it does not consider computational heterogeneity. Using the *Dispatch* strategy does lead to increased partitioning overheads that include the costs to extract the existing pointwise weights and interpolate new ones, compute the weights of local grid blocks for each processor, determine the global workload, and perform an appropriate domain decomposition based on the heterogeneous loads. However, the cumulative partitioning overheads are of $O(10)$ seconds, which is an order of magnitude smaller than the application execution time. Hence, the partitioning overheads for *Dispatch* are considered negligible compared to the performance improvement in the uniform grid case. Figures 7 and 8 plot the same metrics for a 512×512 uniform grid run, and similar performance improvement is observed as for the 256*256 uniform grid case. *Dispatch* improves application execution times by about 15-50% and provides better load balance. The partitioning overheads are, once again, negligible compared to the overall performance gain. Note that application execution times for 512*512 uniform grid in Figure 7 are approximately 4 times larger than the corresponding times for the 256*256 uniform grid, since the domain has been scaled by a factor of 4. Increasing the resolution on uniform grids to obtain greater accuracy can be expensive, and hence SAMR methods are suitable for this class of applications with localized features.

5.2. SAMR Evaluation

2-level SAMR: The *Dispatch* strategy is evaluated using SAMR implementation of R-D kernel on 8-128 processors of DataStar for an application base grid of resolution 512*512 with 2 levels of factor 2 space-time refinements. The application uses a timestep (Δt) value of 2.1875e-9 and executes for 200 iterations, performing 10 regrids. All other application-specific and refinement-specific parameters are kept constant. The minimum block size for this set of experiments is set to 4. Figures 9 and 10 respectively plot the execution time T_{exec} and the compute μ_{comp} and synchronization times μ_{sync} averaged across processors. Error bars in Figure 10 are the standard deviation of the compute times across processors. The *Dispatch* strategy improves application execution time (upto 32 processors) and achieve a more uniform load balance, though the overall improvement obtained using *Dispatch* reduces as the computation-to-communication ratio (roughly, the ratio of average compute to average synchronization times) for the R-D application decreases. Since the R-D application has localized spiked loads, *Dispatch* generates more patches to yield a better load balance. If there is not enough computation per grid block on each processor and the application is communication-dominated, the *Dispatch* strategy can, in fact, perform

Table 1. Dispatch 2-level SAMR granularity evaluation for 512*512 base grid R-D application on 64 and 128 processors on DataStar.

Number of processors	Grain size	*Homogeneous* Time (sec)	*Dispatch* Time (sec)	Grain size	*Homogeneous* Time (sec)	*Dispatch* Time (sec)
64	4	1155.53	1167.35	32	1742.11	1436.36
128	4	691.13	814.32	16	765.63	664.15

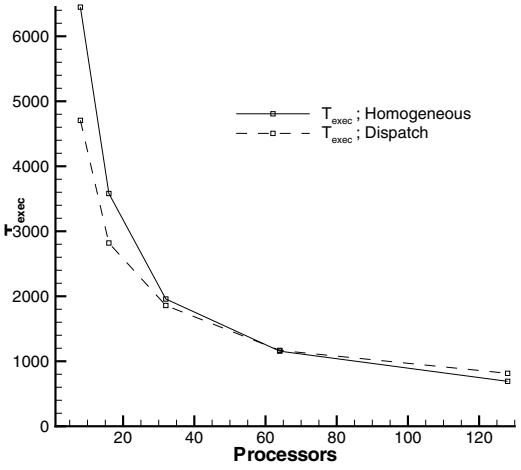

Figure 9. Execution time of the R-D kernel on a 512×512 base grid with two refinement levels plotted as a function of number of executing processors. All times are in seconds.

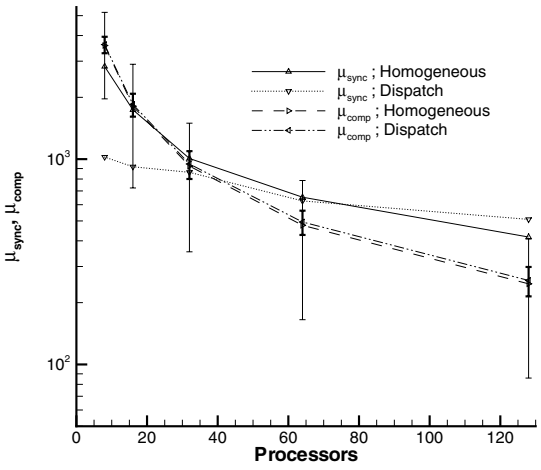

Figure 10. Comparison of compute (μ_{comp}) and synchronization (μ_{sync}) times averaged across all processors. Error bars (dark: *Dispatch*; light: *Homogeneous*) are standard deviations of the compute times. All times are in seconds.

worse than the *Homogeneous* scheme. This is seen in the case of 64 and 128 processors due to the large number of grid blocks created, since the granularity is 4. However, if an appropriate granularity is chosen so that the domain does not contain a large number of tiny blocks, the *Dispatch* strategy can be expected to perform better. Note that this performance variation is a direct result of the application and workload characteristics and the "computation-communication tradeoff" and not a restriction for the *Dispatch* strategy. Furthermore, Table 1 shows that *Dispatch* gives better runtime performance as compared to *Homogeneous* for the 2-level SAMR evaluation on 64 and 128 processors with a base grid resolution of 512*512, if the granularity is set to 32 or 16 instead of 4.

3-level SAMR: The *Dispatch* strategy is evaluated for the R-D application on 16 and 64 processors of DataStar using an application base grid of resolution 256*256 with 3 levels of factor 2 space-time refinements. The application uses a timestep (Δt) value of 8.75e-9 and executes for 200 iterations, performing 10 regrids. All other application-specific and refinement-specific parameters are kept constant. The minimum block size for this set of experiments is set to 4 for the 16 processor run and 16 for the 64 processor run. The application execution times are listed in Table 2. The improvements are lower for the 64 processor run due to the higher synchronization costs. However, the 16 processor evaluation has more computational load, and its performance is improved by the *Dispatch* strategy.

6. Conclusion

This paper presented *Dispatch*, a dynamic partitioning strategy for parallel structured scientific applications with computational heterogeneity. The

Table 2. Dispatch 3-level SAMR evaluation for R-D application with 256*256 base grid on 16 and 64 processors on DataStar.

Number of processors	Grain size	*Homogeneous* Time (sec)	*Dispatch* Time (sec)	Percentage Improvement
16	4	1324.57	1120.1	15.44
64	16	678.55	668	1.55

research was motivated by the observation that accurate solutions to simulations with pointwise varying workloads, such as reactive flows, require an appropriate domain decomposition that considers the load heterogeneity at each grid point. *Dispatch* maintains computational weights as distributed grid functions, computes local workloads and partitioning thresholds, and performs in-situ locality-preserving load balancing. The experimental evaluation of *Dispatch* using an illustrative 2-D reactive-diffusion kernel demonstrated improvement in load distribution and overall application performance. Future work aims to analyze the scalability of *Dispatch* for larger number of processors, and to develop decentralized load balancing and synchronization schemes for *Dispatch* that use local workload information within processor neighborhoods to adapt partitioning behavior during application execution.

Acknowledgements

We thank Dr. John Hewson, Sandia National Laboratories, Albuquerque, NM, for his help with the methane-air chemical mechanism and generating a self-supporting chemistry module for us. The research presented in this paper was supported by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826. This research was also supported by the United States Department of Energy and the Office of Basic Energy Sciences.

References

[1] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, vol. 53. pp. 484-512, 1984.

[2] S. Chandra, X. Li, and M. Parashar. Engineering an autonomic partitioning framework for Grid-based SAMR applications. In *High Performance Scientific and Engineering Computing: Hardware/Software Support*, L. T. Yang and Y. Pan (eds.), Kluwer Academic Publishers, pp. 169-187, March 2004.

[3] S. D. Cohen and A. C. Hindmarch. CVODE, a stiff/-nonstiff ODE solver in C. *Computers in Physics*, vol. 10(2), pp. 138-143, 1996.

[4] P. Colella et al. Chombo infrastructure for adaptive mesh refinement. http://seesar.lbl.gov/ANAG/-chombo/, 2005.

[5] J. C. Hewson and F. A. Williams. Rate-ratio asymptotic analysis of methane-air diffusion flame structure for predicting production of oxides of nitrogen. *Combustion and Flame*, vol. 117, pp. 441-476, 1999.

[6] C. Huang, O. Lawlor, and L. V. Kale. Adaptive MPI. In *16th International Workshop on Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, vol. 2958, pp. 306-322, 2003.

[7] S. Kohn. SAMRAI: Structured adaptive mesh refinement applications infrastructure. http://-www.llnl.gov/CASC/SAMRAI/, 2005.

[8] P. MacNeice, K. M. Olson, C. Mobarry, R. de-Fainchtein, and C. Packer. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, vol. 126, pp. 330-354, 2000.

[9] G. Marchuk. Splitting and alternating direction methods. In *Handbook of Numerical Analysis, Volume I*, P. Ciarlet and J. Lions (eds.), Elsevier Science Publishers B.V., North-Holland, Amsterdam, pp. 197-462, 1990.

[10] S. McCormick and J. Ruge. Unigrid for multigrid simulation. *Mathematics of Computation*, vol. 41(163), pp. 43-62, 1986.

[11] B. Moon, G. Patnaik, R. Bennett, D. Fyfe, A. Sussman, C. Douglas, J. Saltz, and K. Kailasanath. Runtime support and dynamic load balancing strategies for structured adaptive applications. In *7th SIAM Conference on Parallel Processing for Scientific Computing*, San Francisco, CA, February 1995.

[12] M. Parashar, J. C. Browne, C. Edwards, and K. Klimkowski. A common data management infrastructure for adaptive algorithms for PDE solutions. In *ACM/IEEE Conference on Supercomputing*, San Jose, CA, pp. 1-22, 1997.

[13] J. Ray, H. N. Najm, R. B. Milne, K. D. Devine, and S. Kempka. Triple flame structure and dynamics at the stabilization point of an unsteady lifted jet diffusion flame. *Proceedings of the Combustion Institute*, vol. 25(1), pp. 219-226, 2000.

[14] H. Sagan. Space filling curves. Springer-Verlag, 1994.

[15] San Diego Supercomputer Center. DataStar user guide. http://www.sdsc.edu/user_services/datastar/, 2005.