To appear in the proceedings of Computational Methods in Water Resources XII, Crete, 1998.

Efficient parallel computation of spatially heterogeneous geochemical reactive transport

Mary Wheeler, Todd Arbogast, Steven Bryant & Joseph Eaton
**Center for Subsurface Modeling, Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin, Austin, Texas 78712, USA*
EMail: mfw@ticam.utexas.edu, arbogast@ticam.utexas.edu sbryant@ticam.utexas.edu, jeaton@ticam.utexas.edu

Abstract

In flow and non-reactive transport problems, an efficient division of labor for a parallel computer is given by a decomposition of the domain that assigns roughly an equal number of grid cells to each processor. Typical geochemically reactive transport problems exhibit reaction fronts or zones that travel through the domain and may be concentrated near heterogeneities in the domain, such as wells or sources of contaminants. Within these fronts, kinetic reactions may be governed by relatively stiff differential equations, and equilibrium reactions may be near poorly conditioned minima in the Gibbs free energy. Thus the geochemical computational workload can vary greatly with position and time. A domain decomposition tailored for flow and transport problems will result in computationally intensive reaction zones that fall heavily on certain parallel processors but lightly on the rest. We present an algorithm to balance the geochemical load while preserving the efficiency of the flow and transport calculations. Significant performance improvements are observed on a variety of simple test problems.

1 Introduction

The computational effort needed to solve flow and transport problems is dominated by the number of discretization unknowns per grid cell. Reactive transport applications may have tens or even hundreds of

unknowns per cell, and current techniques for geologic characterization can resolve the spatial domain into millions of cells. Parallel processing is an important tool for solving such challenging problems.

Simply dividing a large computation among N processors will not reduce the wall-clock time by a factor of N, however. For example, passing information between processors imposes a penalty on performance. Problems with irregular meshes require nontrivial decompositions of the spatial domain among processors to ensure that the computational load is appropriately balanced.

While much research has focused on developing computer science strategies for efficient parallel processing, a less widely recognized source of computational overhead is the physical problem itself. Some applications are intrinsically "smooth," in the computational sense, meaning that the effort required to solve the governing equations scales with the number of points or elements at which a numerical solution is sought. Any domain decomposition that assigns each processor about the same number of grid cells will distribute the work of solving the problem fairly equally.

In computationally "rough" applications, the local computational workload can vary significantly with position in the domain and with time. Reactive transport problems are a good example. Because of the hyperbolic nature of the transport equation, changes in chemical composition will propagate through the porous medium in a coherent wave-like fashion. In general these waves of chemical change propagate at different velocities, and regions of constant (or nearly constant) composition arise to separate the waves. At a given point in the spatial domain and at a given time, the work required to update the local chemical composition will be small or even zero if that point lies within a region of near constant composition. At the same time, a point through which a chemical wave is passing will require significant computational work to update the local composition.

2 Examples of Computational Roughness

To illustrate computational roughness, we consider the natural attenuation of a mixed-waste contaminant of cobalt and nitrilotriacetate (NTA) in an aquifer containing microbes, idealized as a finite pulse of contaminant entering a one-dimensional porous medium. The cobalt and NTA form various complexes in solution, some of which adsorb

onto the porous medium. One of the NTA complexes undergoes biodegradation. This chemically complex system involves a total of 26 species and 8 chemical components. The biological and adsorption reactions are rate-limited, proceeding according to Monod and mass-action kinetics, while the speciation reactions are at thermodynamic equilibrium (see Valocchi[1]).

This problem was solved on a uniform grid using Parssim, the Parallel Subsurface Simulator (see Wheeler[2]). The code employs operator splitting to solve the transport and reaction subproblems sequentially. The reaction step may be subdivided into several substeps if the kinetics are sufficiently fast. The chemical composition is equilibrated at the end of each kinetic substep. An interior point scheme for the minimization of total Gibbs free energy is used for the equilibrium computation (see Saaf[3]). A good measure of the chemical workload is therefore the number of iterations taken in the computation of thermodynamic equilibrium, plus the number of kinetic substeps.

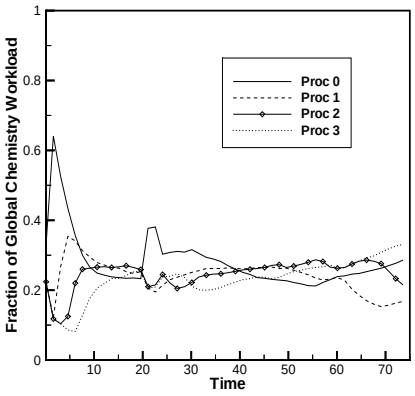

Figure 1: Chemistry workload for 1D bioremediation with slow sorption kinetics.

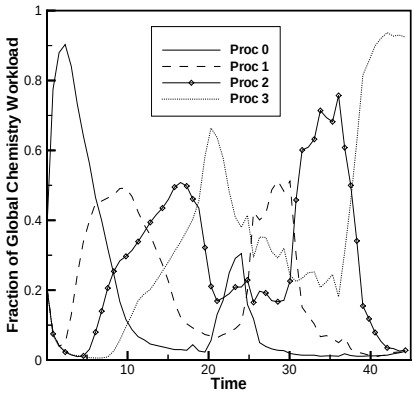

Figure 2: Chemistry workload for 1D bioremediation with fast sorption kinetics.

Figure 1 shows the distribution of the work involved in solving the chemistry subproblem on each of four processors when processor 0 takes the first quarter of the domain, processor 1 the next quarter, and so on. For this simulation, all the rate-limited reactions were taken to be relatively slow. As the pulse of contaminant enters the medium, the chemistry work falls principally upon the first processor

(Proc 0). The advance of the pulse's leading edge is apparent in the subsequent increases in chemistry work on the other processors. Similarly, the trailing edge of the pulse introduces another wave of higher chemistry work when it enters the domain at time 20. These two fronts introduce computational roughness; however, once the pulse has fully entered the porous medium (at about time 25), the problem becomes fairly smooth computationally.

If the sorption rates of this problem are instead relatively fast, the system of ordinary differential equations that governs the kinetics becomes stiff, and the computational work associated with the pulse fronts increases greatly. This exacerbates the imbalance between the processors associated with the entrance of the leading and trailing edges of the contaminant pulse, as shown in Figure 2. Even after the pulse has fully entered the medium, the problem remains very rough computationally. As a result, the domain decomposition that balanced the work well for slow kinetics (after the pulse had fully entered the medium) is very unbalanced when some reactions are fast.

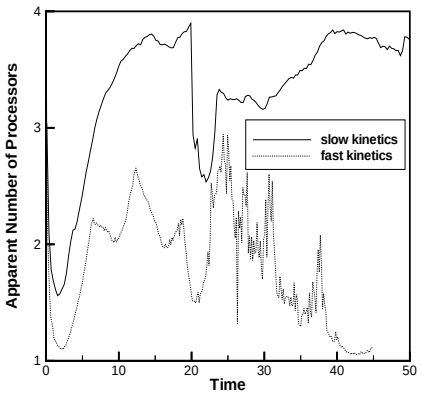

Figure 3: Efficiency of uniform domain decomposition across 4 processors for the 1D bioremediation examples.

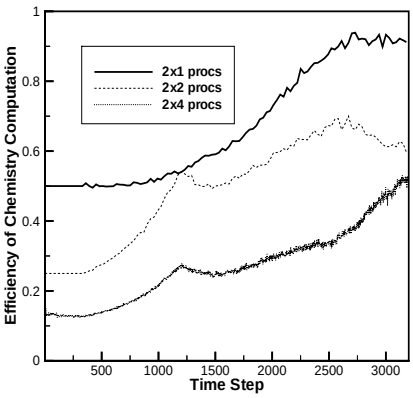

Figure 4: Utilization of parallel capacity for 3 domain decompositions in the 2D precipitation-dissolution problem.

In Figure 3 we show a measure of this load imbalance by plotting the apparent number of processors working on the chemistry calculations at any given time of the simulation. This measure is idealized in that overheads for message passing are ignored. During the

early stages of the simulation (time < 5), when the pulse is entering the medium, the processors assigned to the domain far from the entrance spend most of their time waiting for the processors handling the near-entrance domain to finish their chemistry calculations. At later stages, performance approaches optimal for the slow kinetics case, except for a brief imbalance associated with the trailing edge of the pulse. For the fast kinetics case, performance is far from optimal, especially toward the end of the simulation.

Next consider an example involving 2D flow from an injection well. The injected solution dissolves one mineral from the porous medium. The species liberated by dissolution react with other species in the injected solution, inducing precipitation of other minerals. These precipitates are soluble in the injected fluid, however, and thus they in turn are also dissolved. All reactions are near thermodynamic equilibrium, resulting in sharp precipitation/dissolution waves propagating through the medium. Essentially all of the chemistry computation is associated with these fronts and the leading edge of the injected solution, making this a computationally very rough problem.

Consequently, a partition of the domain among processors by assigning equal numbers of cells to each processor is not optimal. The efficiency (ratio of apparent number of processors to actual number) of 2×1, 2×2 and 2×4 decompositions is shown in Figure 4. Since in the early stages of the simulation all the chemistry computation is near the well, the single processor assigned to the domain containing the well limits the overall speed, regardless of how many other processors are involved. As the reaction fronts propagate farther into the medium, the work associated with them is distributed across more processors and the efficiency increases. But because the fronts are sharp, inevitably a substantial fraction of the processors always have much less chemistry work to do. Moreover, this fraction tends to increase as the number of processors increases.

These examples show that reactive transport problems are often computationally rough, and that this roughness is dynamic. Imbalance of the chemistry work can result in large reductions in parallel efficiency, due solely to the physical problem itself. Very poor scaling can arise even if great care is taken with regard to the purely computer science issues of message passing overhead and other details of parallel implementation. Consequently, some form of load balancing is essential for successful large scale reactive transport simulation.

Clearly any such scheme must be adaptive, since the spatial location of regions of high chemistry workload varies with time.

3 An Algorithm for Dynamic Load Balancing

Our algorithm attempts to dynamically balance the computational work associated with solving any computationally rough problem.

We assume that each grid cell is "owned" by a single processor, as dictated by the computational load balance needed for the flow and transport computations. For the chemistry computational work, we develop an algorithm that migrates grid cells from processors with more than the average amount of work to processors with less than the average amount of work, performs the needed chemistry work on those grid cells, and then returns the results to the owning processors. There are three major steps to the algorithm.

Step 1. Estimate the amount of work each processor will do to complete the chemistry subproblem on each grid cell. We assume simply that the work of the current time level is similar to the work of the previous time level.

Step 2. Calculate a new distribution of work that would (nearly) equalize the work of the processors. This requires 3 substeps.

(a) Compute the total (chemistry) work to be done, the average work per processor, and the amount by which each processor is overworked or under-worked relative to this average.
(b) Calculate an (approximately) balanced distribution of work. That is, determine the fraction of work that needs to be shifted out of or into each processor.
(c) Determine the movement of grid cells needed to achieve the balanced work distribution.

Step 3. Execute parallel calculations over the balanced distribution of grid cells as follows. For each processor:

(a) If under-worked, post a non-blocking receive command to catch the incoming work. Otherwise, pack message buffers with data from the excess grid cells and send them to the appropriate processors.
(b) Perform computations on the grid cells that do not require the passing of data in parallel.
(c) If under-worked, perform proxy computations on the incoming grid cells, and return the results. Otherwise, wait for these

proxy results to be returned.

(d) Accumulate statistics on the work required for each grid cell for this time level.

Steps 2(b) and 2(c) are performed using integer arithmetic to add work to under-worked processors until they have approximately the average load. Parallel communication is done with non-blocking, point-to-point MPI calls, allowing the maximum overlap of computation with communication. The cost of this algorithm is bounded above by $4 \log_2(N)\, \alpha$ in parallel communication time, where N is the number of processors and α is the time required to initiate a point-to-point communication on the given machine. We expect this algorithm to scale linearly as the number of processors is increased. Our current implementation of this algorithm into Parssim uses the count of equilibrium iterations and kinetic substeps for ODE integration as a measure of the work.

4 Results

For computationally smooth problems, such as the slow kinetics bioremediation example (cf., Figure 1), and for computationally rough problems in which the chemistry workload is small compared to the flow and transport workload, the algorithm yields marginal improvements in performance, from 0% to 20% speedup relative to the unbalanced code. This indicates that the algorithm adds very little computational overhead, so that it can be invoked routinely to improve performance for all parallel reactive transport simulations, regardless of whether they are computationally rough or smooth.

The principal application of the algorithm is to computationally rough problems which are dominated by the chemistry workload. In the fast kinetics bioremediation example (cf., Figure 2) we obtained a speedup of 2.0 relative to a uniform domain decomposition with no balancing. For reference, the maximum possible speedup for balancing the chemistry work in this example is 2.13 (assuming negligible work for flow and transport). For the 2D precipitation/dissolution problem with fast kinetics rather than equilibrium reactions (which substantially increases the amount of chemistry work), we obtained a speedup of 2.04. These examples clearly illustrate the benefit of load balancing. The implications for scalability are even greater: the maximum possible speedup will be proportional to the number of

processors in many cases. This algorithm is capable of approaching maximum speedup.

5 Conclusions

Dynamic load balancing is critical to parallel performance for problems dominated by the time required to solve rough (in either time or space) problems. Such computational roughness frequently arises in reactive transport in porous media, because the chemistry computation is often physically localized to reaction fronts, wells and other heterogeneities. In many problems the chemistry computation is a significant or even dominant fraction of the total computational workload. The advantages of parallel processing can be greatly reduced or even eliminated by failing to balance such problems. By performing dynamic load balancing with a low-overhead algorithm, we have been able to achieve near optimal parallel performance in our flow and reactive transport simulator Parssim.

6 Acknowledgments

This work was supported in part by the U.S. Department of Energy and the National Science Foundation.

References

[1] Valocchi, A., *Parameter sensitivity in multicomponent reactive transport models*, presented at the Workshop on Subsurface Reactive Transport Modeling, Pacific Northwest National Laboratory, 29 Oct.–1 Nov., 1997.

[2] Wheeler, M.F., T. Arbogast, S. Bryant, C. Dawson, F. Saaf & C. Wang, *Computational methods for multiphase flow and reactive transport problems arising in subsurface contaminant remediation*, J. Comp. Appl. Math., 74, pp. 19–32, 1996.

[3] Saaf, F., R. Tapia, S. Bryant & M. Wheeler, *Computing general chemical equilibria with an interior-point method,* in Computational Methods in Water Resources XI: Computational Methods in Subsurface Flow and Transport Problems, pp. 201–208, 1996.