Design Strategies for Irregularly Adapting Parallel Applications

Leonid Oliker[*] *Rupak Biswas*[†]
Hongzhang Shan[‡] *Jaswinder Pal Singh*[§]

Achieving scalable performance for dynamic irregular applications is eminently challenging. Traditional message-passing approaches have been making steady progress towards this goal; however, they suffer from complex implementation requirements. The use of a global address space greatly simplifies the programming task, but can degrade the performance of dynamically adapting computations. In this work, we examine two major classes of adaptive applications, under five competing programming methodologies and four leading parallel architectures. Results indicate that it is possible to achieve message-passing performance using shared-memory programming techniques by carefully following the same high level strategies.

Adaptive applications have computational workloads and communication patterns which change unpredictably at runtime, requiring dynamic load balancing to achieve scalable performance on parallel machines. Efficient parallel implementations of such adaptive applications are therefore a challenging task. This work examines the implementation of two typical adaptive applications, Dynamic Remeshing and N-Body, across various programming paradigms and architectural platforms. We compare several critical factors of the parallel code development, including performance, programmability, scalability, algorithmic development, and portability.

[*]One Cyclotron Rd, MS:50B-2239, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (loliker@lbl.gov).

[†]Mail Stop T27A-1, NASA Ames Research Center, Moffett Field, CA 94035 (rbiswas@nas.nasa.gov).

[‡]Department of Computer Science, 35 Olden Street, Princeton University, Princeton, NJ 08544 shz@cs.princeton.edu.

[§]Department of Computer Science, 35 Olden Street, Princeton University, Princeton, NJ 08544 jps@cs.princeton.edu.

Dynamic Remeshing and N-Body are irregular dynamic applications where the processor workloads and interprocessor communication can change dramatically with time; thus, dynamic load balancing is a required component. Our Dynamic Remeshing application simulates flow over an airfoil, by refining localized regions of the underlying unstructured mesh. It consists of several distinct modules, including mesh adaptation, load balancing, and numerical solution. The N-Body problem is a classical one, and arises in many areas of science and engineering. Having specified the initial positions and velocities of the N interacting bodies, the problem is to find their positions after a certain amount of time. Our experiments simulate two neighboring Plummer model galaxies that are about to undergo a merger.

We compare the most popular implementation strategy, message passing with MPI, against a number of alternate approaches. In the message-passing model, each process has only a private address space, and must communicate explicitly with other processes to access their (private) data. This model is perhaps the most difficult to program for irregular applications; however, the benefits lie in enhanced performance for coarse-grained communication and implicit synchronization through blocking communication. We also evaluate the effectiveness of using the SHMEM communication library, which uses symmetric address spaces for individual processes. This allows one-sided communication, unlike explicit message passing which requires send-receive pairs. Message-passing results are present on several parallel systems, including the distributed-memory Cray T3E located at NERSC.

Next, we examine an implementation using the OMP programming strategy, which uses shared-memory algorithms and OpenMP-style compiler directives on systems supporting global addressing. OMP can greatly reduce programming overhead compared to message passing, but may suffer from poor spatial locality of physically distributed shared data. The cache-coherent shared address space (CC-SAS) programming methodology is then examined for our adaptive applications. Here, remote data are accessed just like locally-allocated data using loads and stores, much like OMP. However, the CC-SAS approach focuses on spatial locality through methods such as data remapping and replication, which are traditionally not considered a part of the shared-memory programming paradigm. Both OMP and CC-SAS results are presented on the SGI Origin2000, a scalable, hardware-supported cache-coherent non-uniform memory access (CC-NUMA) system, with an aggressive communication architecture.

Experimental results using the hybrid programming paradigm are then reported. This mixed programming method is well suited for the latest teraflop-scale parallel architectures, which are built as clusters of shared-memory multi-processors (SMPs). Here, two layers of parallelism are combined by implementing OpenMP codes within the SMPs, while using message passing between the SMP clusters. For our test cases, we weigh the performance gains versus the increased programming complexity and reduced portability, as compared with pure MPI codes. Hybrid experiments are performed on the latest generation of IBM POWER3 SMP cluster located at SDSC. Each node of this system consists of eight processors connected via a crossbar to the shared memory. The nodes are connected to each other through an omega-type topology switch interconnect.

Finally, we present multithreaded results on the radically different architecture of the Cray MTA. The MTA is a true shared-memory system in which data placement is not required due to a unique memory hashing scheme. This makes programmability much easier than on standard cache-based multiprocessor systems. Rather than using data caches to hide latency, the MTA processors use multithreading to tolerate latency. Performance thus depends on having a large number of concurrent computational threads. Experiments were performed on the eight-processor MTA located at SDSC. Results show that multithreaded systems offer tremendous potential for efficiently solving some of the most challenging problems on parallel computers. However, they are not well suited for all classes of computations.