A Distributed Object Model for Solving Irregularly Structured Problems on Cluster

Yudong Sun and Cho-Li Wang
Department of Computer Science and Information Systems
The University of Hong Kong
{ydsun, clwang}@csis.hku.hk

Abstract

This paper presents a distributed object model MOIDE for solving irregularly structured problems on cluster. The primary appeal of MOIDE is its flexible system structure that is adaptive to heterogeneous architecture of a cluster. MOIDE integrates the object-oriented and multithreaded methodologies to set up a unified computing environment. Both the shared-data access and remote messaging are incorporated in a two-layer communication mechanism for efficient inter-object communication with the common communication interface. MOIDE supports dynamic load balancing by its autonomous load scheduling technique. A runtime support system implements the MOIDE model as a platform-independent infrastructure for developing and executing irregularly structured applications. N-body, ray tracing, and conjugate gradient applications are implemented to illustrate the advantages of MOIDE model.

1. Introduction

An Irregularly Structured Problem (ISP) is the application whose computation and communication patterns are input-dependent, unstructured, and evolving in the computation procedure [1]. A lot of applications in different fields can be classified as irregularly structured problems. Examples can be found in astrophysics, fluid dynamics, sparse matrix computations, system modeling and simulations, computer graphics and etc.

The irregularly structured problems are usually computation-intensive applications. From the data structure aspect, the irregular and dynamic nature of data distribution in ISP usually needs more complicated data structures to flexibly represent its runtime feature. The computations based on these data structures exhibit strong data dependency and limited parallelism while solving ISP in distributed environment. As a result, the data structures that describe the input data and intermediate status of ISP should be flexible enough to suit the task decomposition and could be reconstructed in a flexible way to reflect the data evolution during the computation.

It is also observed that the workload of ISP depends on the input data and its dynamic evolution in the computation, which cannot be accurately measured. Thus, it is usually difficult to evenly distribute the workload of an ISP onto multiprocessors by a static task mapping. The high data dependency among the decomposed subtasks further complicates the task scheduling. In addition, due to the high data dependency, ISP also presents irregular inter-process communication that severely affects the overall performance. To achieve high performance computation of ISP, we need an adaptive computing infrastructure and various efficient mechanisms for solving the above problems.

This paper presents a distributed object model MOIDE (Multithreaded Object-oriented Infrastructure on Distributed Environment) for solving irregularly structured problems on cluster. Based on this model, we build an adaptive and architecture-independent computing infrastructure for developing and executing irregularly structured applications. MOIDE system supports the following features:

(1) *Hierarchical Collaborative System* (HiCS): HiCS is the core of MOIDE. It is a runtime distributed object system built on the cluster nodes. HiCS helps to adaptively map the subtasks onto the underlying heterogeneous cluster nodes at runtime and seamlessly access the cluster resources.

(2) *Autonomous Task Scheduling* and *Asynchronous Computation*: Non-predetermined and dynamically evolving load distribution in ISP is the main obstacle to load balancing. Autonomous task scheduling can guarantee the dynamic load balance during the execution. To further improve the asynchrony in computation, remote messaging is implemented in the form of *one-sided communication* in which the communication operation is started on either the sender or the receiver only without the explicit participation of the communication partner.

(3) *Multithreading* and *Two-level Communication*: Multithreading support and two-level communication

mechanism are implemented in MOIDE to enhance the performance. The ability to create and control multiple threads is especially important in developing irregularly structured applications, since the computations are typically more asynchronous and dynamic. The integration of object-oriented and multithreaded methods facilitates the efficient inter-thread communication on heterogeneous cluster. A two-layer communication mechanism is implemented, which can dynamically link the communication interface to either local (object sharing) or remote (message passing) communication.

Three irregularly structured applications, including *N*-body problem, ray tracing and CG (conjugate gradient), are implemented to demonstrate the advantages of MOIDE model.

2. MOIDE: A Distributed Object Model

2.1 System Architecture

HiCS is constructed with a group of objects. Figure 1 plots the structure of a hierarchical collaborative system built on a cluster of *P* hosts. The object on *Host 0* is called *compute coordinator*, which is the initiator of the system. It creates the remote objects on other hosts and allocates computing tasks to them. It also coordinates the whole computing procedure and conducts the system-wide synchronization. Other objects are called *compute engines*, which accept and execute the computing tasks. The *registration mechanism* registers the references of the computer coordinator and compute engines for the interaction between them. The *interaction mechanism* is the communication interface that supports the *two-layer communication*. The *host selection mechanism* is used to detect and select the hosts in the underlying cluster.

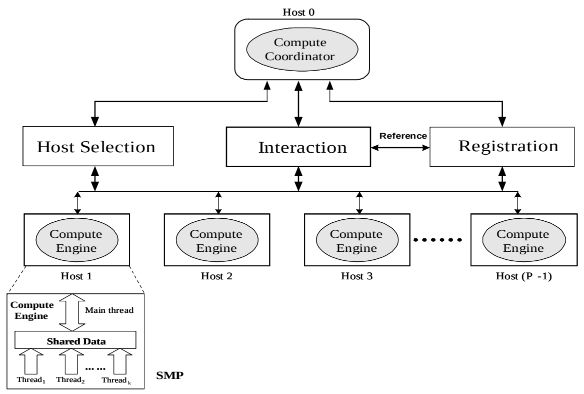

Figure 1. Hierarchical collaborative system

HiCS has a two-level structure. The compute engines form the upper level that is directly managed by the compute coordinator. The compute coordinator or compute engines can generate multiple computing threads within themselves. For example in Figure 1, multiple threads are generated on *Host 1*, which is an SMP node.

A runtime support system, *MOIDE-runtime*, is developed to support MOIDE-based computation. The major components of MOIDE-runtime are shown in Figure 2. It includes the class of compute coordinator and the class of compute engine. MOIDE-runtime supports two types of system reconfiguration: *system expansion* that adds new compute engine into HiCS and *host replacement* that replaces the compute engine on an overloaded host by the new compute engine on another host. MOIDE-runtime is implemented in Java and RMI so that it is executable on heterogeneous platforms.

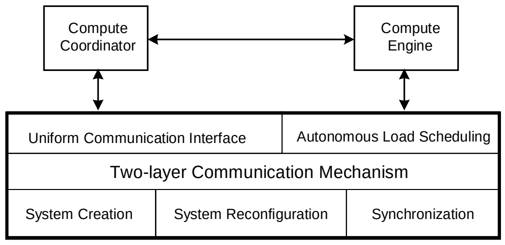
Compute Coordinator
Compute Engine
Uniform Communication Interface
Autonomous Load Scheduling
Two-layer Communication Mechanism
System Creation
System Reconfiguration
Synchronization

Figure 2. MOIDE runtime support system

2.2 Multiple Work Modes

When building HiCS, the creation of the multiple threads are related to the architecture of the underlying hosts. With the location information maintained in the registration mechanism, the threads can be flexibly organized into one of the execution modes to run an application based on the computation requirement, (1) c*ooperative mode*: the group of threads work together to cooperatively process a computing task; (2) *independent mode*: each thread works as an independent compute engine and processes individual computing task.

2.3 Two-layer Communication

There are two communication paths in HiCS. Local communication within a group of threads can be fulfilled by rapid local data-object access. Only the communication between the distributed objects calls for remote messaging. These two paths are integrated to create a *two-layer communication mechanism*. The shared-data access can reduce the irregular communication cost. Therefore the two-layer communication can reduce the heavy and unpredictable communication overhead in irregularly structured problems.

The two-layer communication mechanism is transparent to the applications. MOIDE runtime support system provides a uniform communication interface at application level. Application calls the same interface and the runtime support system will decide the proper communication path depending on the locations of the communication partners.

2.4 Autonomous Load Scheduling

It is difficult to evenly decompose an irregular computation before execution. MOIDE supports *autonomous load scheduling* as a dynamic load balancing scheme. Differing from the master/slave load scheduling such as in [2], the autonomous load scheduling needs no dedicated load scheduler. It allows a compute engine or thread to directly fetch a computing task from the global task pool on demand. As a result, the computing tasks can be progressively distributed to the compute engines or threads. Therefore dynamic load balance can be automatically reached.

3. Irregularly Structured Applications

Three irregularly structured applications are developed on MOIDE model. These applications have distinct irregular features that demand specific techniques adopted in the design of high-performance solutions.

3.1 Distributed *N*-body Method

N-body problem simulates the evolution of a physical system that contains numerous particles (bodies). It is a computation- and communication-intensive problem. The *N*-body method on MOIDE model is derived from the Barnes-Hut method [3]. We designed a distributed tree structure as the distributed variation of the Barnes-Hut tree [4]. The distributed tree is created by space decomposition. Each compute engine builds a subtree. The threads in each compute engine work in cooperative mode to share the subtree and compute the force exerted on the bodies in the subset. The distributed tree structure also includes a partial subtree scheme to satisfy the data-sharing requirement under low communication cost. A partial subtree contains the top-half levels of the subtree, which is broadcasted to all compute engines. If a body needs more information beyond a partial subtree in force computation, the body will be sent to the remote compute engine to access the full subtree there.

The distributed *N*-body method is tested on a cluster of four quad-processor SMP nodes linked with Fast Ethernet switch. The *N*-body method runs on the cluster with the problem sizes *N* from 10,240 to 102,400 bodies. Figure 3 depicts the execution time breakdowns. The computation dominates the execution time. There is not obvious uprising of the communication cost when increasing the number of processors and the problem size. The proportion of the communication time decreases in the total execution time on large problem size.

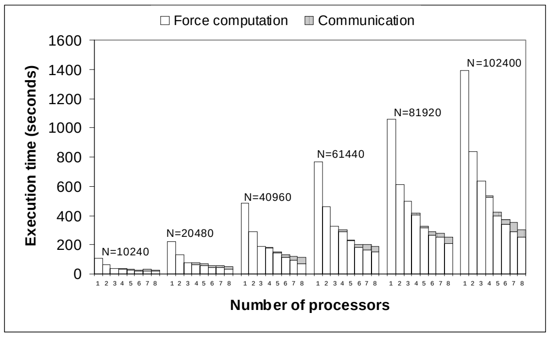

Figure 3. Execution time breakdowns of the distributed *N*-body method

3.2 Ray Tracing

Ray tracing is a graphic rendering algorithm. It generates an image on a view plane from the mathematical description of the objects in the scene. In parallel ray tracing, the view plane is partitioned into blocks that can be rendered in parallel. We demonstrate the efficiency of the proposed autonomous load scheduling mechanism by a parallel ray tracing application, as the rendering of a block is independent from any other block.

During the execution, all computing threads are working in independent mode and all threads can individually perform the rendering operations. Having finished the rendering of a block, a thread can directly fetch next block from the global task pool. Thus, the computation and communication procedures on all threads are fully asynchronous. The autonomous load scheduling can exploit the computing power of all threads and achieve the highest parallelism in the ray tracing.

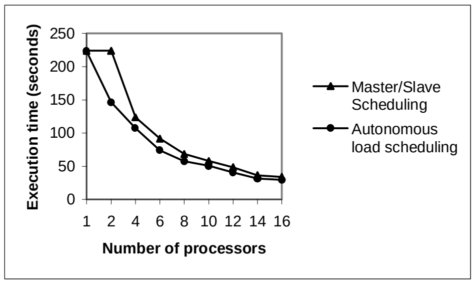

Figure 4. Execution times of the ray tracing

The ray tracing application is tested on the cluster of four quad-processor SMP nodes by comparing the performance with the master/slave scheduling. As the execution times in Figure 4 show, the autonomous load scheduling is superior to the master/slave scheduling due to the higher asynchrony in the autonomous load scheduling.

3.3 Conjugate Gradient

The conjugate gradient (CG) is an iterative method for solving large sparse linear systems. It obtains the approximated solution by iteration. Parallel CG algorithm is designed on the multiprocessors in mesh [5]. In each iteration, it performs multiple reduction and transpose communication operations for large vectors.

The implementation of CG is for the demonstration of the efficiency of the proposed two-layer communication and to verify the adaptability of MOIDE model. All computing threads are performed in independent mode and can be mapped into a mesh structure regardless of their physical locations.

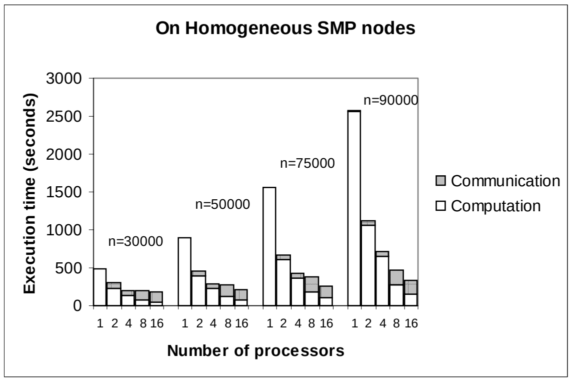

Figure 5. Execution time breakdowns of the CG method on homogeneous SMP cluster

Figure 5 shows the execution time breakdowns of the CG method on the cluster of four quad-processor SMP nodes (the size of sparse matrix A is $n \times n$). With the two-layer communication, the communication overhead does not increase linearly with the problem size. Higher efficiency can be achieved on large problem size.

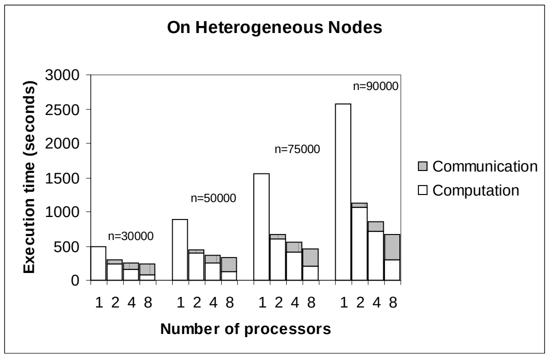

Figure 6. Execution time breakdowns of the CG method on heterogeneous hosts

The CG method is also tested on a cluster of three heterogeneous hosts: one quad-processor SMP and two dual-processor nodes. All processors logically form a 2×4 mesh. The execution time breakdowns are shown in Figure 6. The performance on the heterogeneous cluster is close to that on the homogeneous SMP nodes but the communication cost is a bit higher.

4. Conclusions

From the implementations of the irregularly structured applications, we can find that MOIDE is a flexible model to support the adaptive mapping of irregularly structured problems onto heterogeneous systems. The two-layer communication mechanism can reduce the communication overhead and enhance the overall performance of communication-intensive irregular applications. The autonomous load scheduling is an effective approach to produce a runtime workload balance and exploit the high parallelism during the execution of the applications. The distributed tree structure in the *N*-body method is the communication-efficient data structure to handle the high data-dependency involved in the irregular computation.

The future work will emphasize on the system scalability. For wide-area network computing as on Grid, the current two-level hierarchical collaborative system should be expanded to a multilevel structure to integrate large number of computer nodes on different levels in a distributed system. All compute engines in the HiCS should have more autonomy rather than the single compute coordinator. The system coordination, load scheduling, and communication strategies should be improved to suit the multilevel system structure.

References

[1] T. Gautier, J. Roch and G. Villard, “Regular versus Irregular Problems and Algorithms”, Proceedings of Second International Workshop on Parallel Algorithms for Irregularly Structured Problems, IRREGULAR’95, Lyon France, September 1995, pp.1-25.

[2] A. Fava, E. Fava and M. Bertozzi, “MPIPOV: a parallel implementation of POV-Ray based on MPI”, Proceedings of Euro PVM/MPI, Barcellona, Spain, LNCS 1697, September 1999, pp. 426-433.

[3] J.P. Singh, C. Holt and et al., "Load Balancing and Data Locality in Adaptive Hierarchical *N*-body Methods: Barnes-Hut, Fast Multipole, and Radiosity", Journal of Parallel and Distributed Computing, vol.27, No.2, 1995, pp.118-141.

[4] Y. Sun, Z. Liang and C.L. Wang, “Distributed Particle Simulation Method on Adaptive Collaborative System”, Future Generation Computer Systems, vol. 18, issue 1, September 2001.

[5] V. Kumar, A. Grama and et al., “Introduction to Parallel Computing: Design and Analysis of Algorithms — 11.2.3 The Conjugate Gradient Method”, Benjamin/Cummings Publishing Co., Redwood City, Calif., 1994, pp. 433-435.