Adaptive Computation over Dynamic and Heterogeneous Networks

Kaoutar El Maghraoui[1], Joseph E. Flaherty[1], Boleslaw K. Szymanski[1], James D. Teresco[2], and Carlos Varela[1]

[1] Rensselaer Polytechnic Institute, Troy, NY 12180, USA,
szymansk@cs.rpi.edu,
http://www.cs.rpi.edu/
[2] Williams College, Williamstown, MA 01267, USA

Abstract. Over the last two decades, efficient message passing libraries have been developed for parallel scientific computation. Concurrently, programming languages have been created supporting dynamically reconfigurable distributed systems over the heterogeneous Internet. In this paper, we introduce SALSA-MPI, an actor programming language approach to scientific computing that extends MPI with a checkpointing and migration API and a runtime system that manages both periodic checkpoints and process or application migration. The goal is to enable dynamic network reconfiguration and load balancing without sacrificing application performance or requiring extensive code modifications. As driving technology for this effort of unifying parallel and distributed computing, we plan to use adaptive solvers of partial differential equations. Fields as diverse as fluid dynamics, material science, biomechanics, and ecology make use of parallel adaptive computation, but target architectures have traditionally been supercomputers and tightly-coupled clusters. SALSA-MPI is intended to allow these computations to make efficient use of more distributed and dynamic computing resources.

1 Introduction

Large-scale scientific and engineering applications involving the solution of partial differential equations are among the most demanding computational problems, arising in fields including fluid dynamics, material science, biomechanics, and ecology. Adaptivity, where meshes and numerical methods are automatically adjusted to achieve specified levels of solution accuracy, and parallelism are essential tools to solve modern multi-dimensional transient problems. The usual approach to these problems is to distribute a discretization (mesh) of the domain across cooperating processors, then to compute a solution, appraising its accuracy using error estimates at each step. If the solution is accepted, the computation proceeds to the next step. Otherwise, the discretization is refined adaptively, and work is redistributed, if necessary, to correct for any load imbalance introduced by the adaptive step. The adaptive strategies automatically refine, coarsen, and/or relocate meshes and may also change the method with a goal of obtaining a solution to a prescribed level of accuracy as quickly as possible [1]. Adaptivity makes automatic (compiler-directed) parallelization difficult, so dynamic

partitioning and load balancing procedures become necessary since the locations where meshes must be refined or simpler numerical procedures replaced by more complex ones are not known *a priori* and are determined as part of the solution process.

The adaptive software with the described above features is complex and hard to develop. Hence, the existing software of this kind is very valuable and difficult to replace, motivating our research on combining middleware written in a new actor programming language SALSA with the existing C++ codes using MPI. We target the adaptive software base developed at Rensselaer's Scientific Computation Research Center that executes in serial and parallel computational environments [2, 3]. It has been used successfully by many software packages for classical finite element [4], finite volume [5], and discontinuous Galerkin (DGM) [6, 7] methods. *DG* [7] is a software package that implements a parallel adaptive DGM using the *Algorithm Oriented Mesh Database* (AOMD) [8] mesh structures and services. AOMD supports a variety of mesh representations, including hybrid meshes. It is written in C++ using the Standard Template Library [9] for computations and the Message Passing Interface (MPI) [10] for communication. DG is used to solve a wide range of problems including Rayleigh-Taylor flow instabilities [7]. Distributed AOMD meshes [3] use the *Rensselaer Partition Model* (RPM) [11] to aid in data distribution and migration.

The importance of the applications and, perhaps, the cost of access to supercomputers have led to proliferation of solution strategies on other architectures including PC clusters and, most recently, grids [12, 13]. Target architectures range from small clusters to the largest supercomputers with interprocessor communication ranging from shared memory to wide-area networks.

As discussed in [14] in this volume, open source SALSA actor programming language and IO middleware provide distribution *transparency* to scientific programmers and support *efficient* message passing. Yet, only software written in SALSA can fully benefit from these features. The main contribution of this paper is the SALSA-MPI middleware that supports dynamic partitioning and load balancing for existing software for parallel adaptive partial differential equation solvers. Our middleware improves also computation fault-tolerance via data and process migration and replication. When fully developed, SALSA-MPI will provide a fully integrated software framework linking the applications layer (programmer interface) with the middleware layer, so that adaptivity and transparency can be simultaneously and efficiently achieved.

2 Programming Abstractions and Technology for Dynamic Grids

The Java [15] platform – which includes the definition of a programming language, a virtual machine, and a set of libraries providing high-level application programming interfaces (API) – is a step forward in portable distributed software engineering. In particular, Java's support for concurrent and distributed programming includes multithreading and remote method invocation APIs. Although a common perception exists that Java's main drawback is its lack of performance caused by its bytecode interpretation overhead, recent advances in JIT (Just In Time) compilation and adaptive compilation make Java a very attractive platform for scientific applications [16].

SALSA [17] is an actor-oriented programming language with high-level constructs for remote messaging, universal naming, migration, and coordination. SALSA programs are compiled into Java code, allowing a heterogeneous network of physical machines to be viewed as a homogeneous network of Java virtual machines. The WWC (World-Wide Computer) run-time architecture consists of naming servers and virtual machines running as Java applications on different Internet nodes. The virtual machines, called *theaters*, provide an environment for execution of universal actors using local resources. High-level programming language abstractions enable actors to create remote communication links with peer actors running on other WWC theaters. Furthermore, actors can easily migrate with their full state to other WWC theaters as they become available, supporting dynamic load balancing and scalability. The naming servers keep track of universal actor locators, so that communication remains transparent to actor location and migration. To enable the existing programs to take advantage of the actor features, we developed a middleware, called SALSA-MPI, that enables MPI-based programs to be viewed as actors by the SALSA/WWC computational environment.

3 SALSA-MPI

3.1 SALSA-MPI Architecture

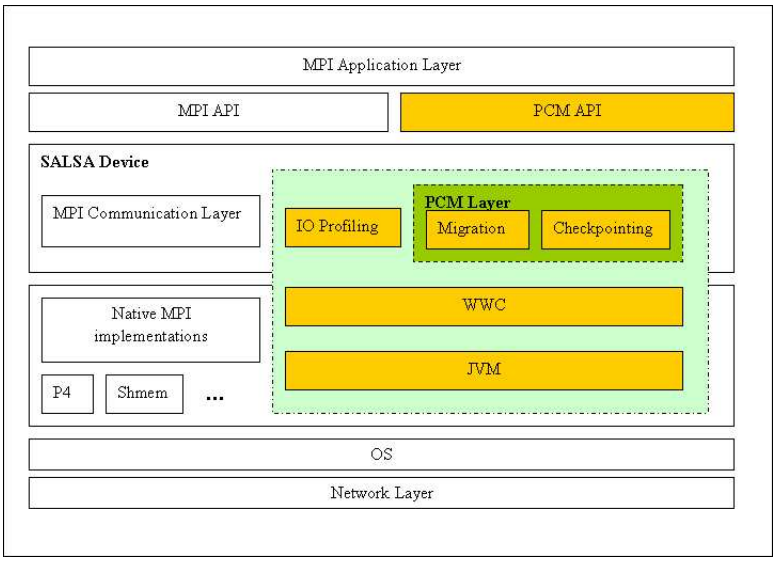

Fig. 1. SALSA-MPI Architecture.

The SALSA/IO architecture [18] consists of an actor-oriented programming language (SALSA), a distributed runtime environment (WWC), and a middleware infras-

tructure for autonomous load balancing (IO: Internet Operating System). This infrastructure is highly dynamic thanks to its support for 1) resource profiling, 2) migration of actors to optimal execution environments, and 3) dynamic addition and removal of nodes in the computation. SALSA-MPI provides an infrastructure to allow the MPI processes to run on dynamic grid environments. This is achieved through the use of the IO middleware that supports dynamic reconfiguration and load balancing. Figure 1 shows the proposed SALSA-MPI architecture. The SALSA-MPI communication driver runs on top of vendor supplied MPI implementations. It uses intra-machine vendor supplied MPI implementations and inter-machine TCP communication. The SALSA device provides also an interface to SALSA/IO proxy actors, which act as profiling actors in the IO network. Every SALSA-MPI proxy-actor has a profiling actor and a decision agent. The profiling actor monitors the communication of running MPI processes and the participating nodes' memory, CPU, and network bandwidth. This profiling information is fed periodically to the decision agent. As processes join and leave the computation, the decision agent tries to reconfigure the running MPI application by migrating some of the MPI processes under its control to improve the theater overall performance. This architecture supports intra-cluster process migration and inter-cluster application migration.

Spreading MPI processes across clusters is expensive because they are usually very tightly coupled. Therefore, it is important to co-locate all the running MPI processes in a given application within one cluster. Inter-cluster application migration on the other hand, can significantly improve the performance if the current cluster experiences failures or increased load.

To support migration, we propose an application-level checkpointing API called PCM (Process Checkpointing and Migration) and a runtime system called PCMD (Process Checkpointing and Migration Daemon). Few PCM calls need to be inserted in MPI programs. They specify the data that need to be checkpointed. They also restore the process to its current state after the migration. This library is semi-transparent because the user does not have to worry about when or how checkpointing and restoration is done. The underlying PCMD infrastructure takes care of all the checkpointing and migration details.

3.2 The PCM API

The PCM API consists of a set of function calls that allow MPI programs to be dynamically reconfigurable. The PCM can be used with any iterative MPI application. The PCM library consists of set of routines with the following functionalities:

- Periodic checkpoints of MPI processes or MPI application data.
- Storage of the checkpointed data in a PCMD daemon (either locally or remotely).
- Restoration of a previously checkpointed data.
- Suspension, restart, or migration of an MPI process or an MPI application.
- Periodic probing of the status of an MPI application or an MPI process

Vadhiyar et al. have devised a similar approach through their SRS library [19]. Our approach differs from Vadhiyar's in two important features. First, our architecture allows for both process and application migration. Second, we use the SALSA-IO middleware to trigger reconfigurability and load balancing when necessary.

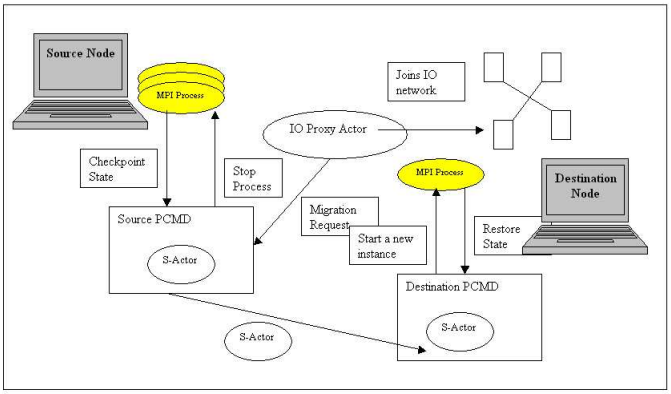

Fig. 2. Interactions among the components of the SALSA-MPI framework.

3.3 PCM Daemons

PCMD daemon needs to be launched in every computational node that joins the MPI parallel computation. The PCMD has a port number on which it listens to incoming requests. It interacts with the running MPI processes, with the IO proxy actor, and with remote PCMD daemons. When the IO decision agent chooses a process for migration, it sends a migration request to the PCMD, which creates a shadow actor (S-Actor) for the migrating process, redirects all messages sent to the migrating process (MP). The MP initiates checkpoints at the PCMD daemon. The S-Actor serializes the state of the MP and migrates to the new destination. On the remote PCMD, the S-Actor starts a new instance of the MP, stores the states of the MP, notifies the source node to stop message redirections and terminates. Once the MP is restarted, it contacts the local PCMD to restore its state. Figure 2 illustrates the interactions between the PCM Daemons and the other components of the SALSA-MPI framework.

3.4 Preliminary Results

The experimental testbed consisted of two clusters at Rensselaer Polytechnic Institute: cluster A (the Sun cluster) consisting of 20 SUN Ultra 10 machines with 256MB of memory and cluster B (the IBM Netfinity cluster) consisting of 40 900Mhz processors with 128MB of memory. Hence, the computation was distributed across two clusters where one has Intel processors and the other has Sparc processors. Our MPI application computed a numerical solution of the classical heat equation in parallel [20]. This is a typical example of an iterative parallel application that requires large volume of communication between the boundaries of the MPI processes. The original MPI code was instrumented by inserting the PCM API calls to allow application reconfiguration and checkpointing by the SALSA-MPI framework. The goal of the first experiment was to determine the overhead incurred by the PCM API. Figure 3 shows the performance

of the original and instrumented MPI application running on cluster A with different numbers of nodes. The overhead introduced by the PCM library is reasonable when the number of nodes is small but it increases as the number of nodes increase. This is the result of using a single PCM Daemon in the current prototype implementation that centralizes its services. In the second experiment, the heat distribution program was first

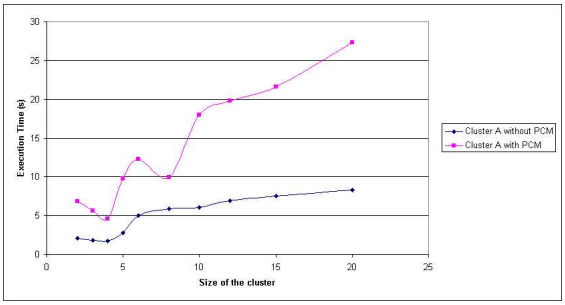

Fig. 3. The overhead that results from instrumenting the heat distribution program with the PCM API calls on cluster A.

run on cluster A. We then substantially increased the load of all the nodes in this cluster by launching several long-running, computationally intensive applications. Figure 4 is based on results from repeating the same experiment, but the MPI application was migrated to cluster B once the load in cluster A increased. As expected, the performance of the MPI application improved when it migrated to a lightly loaded cluster. The performance improves only when there are fewer than 10 nodes. Again this is caused by the limitations of the current prototype implementation. These preliminary results show that reconfigurability and load balancing of parallel MPI applications can improve the performance of the application. The overhead incurred by the PCM library is offset by the application's overall improved performance. The results also show that having a decentralized architecture is necessary to achieve scalability.

4 Discussion and Future Work

We have investigated programming methodologies that promote a separation of the concerns in the implementation of large scientific computations on a large network of computers. High-level programming abstractions provide a natural interface to scientists so that they can concentrate on their domain of expertise. Programming tools map these high-level abstractions into executable units that support efficient communication, dynamic partitioning and load balancing. Run-time middleware infrastructure supports adaptability of executing systems to an evolving underlying network. The presented programming paradigm, languages, and tools are a first step towards the unification of parallel and distributed computing by enabling systems to adapt to different and evolving execution environments.

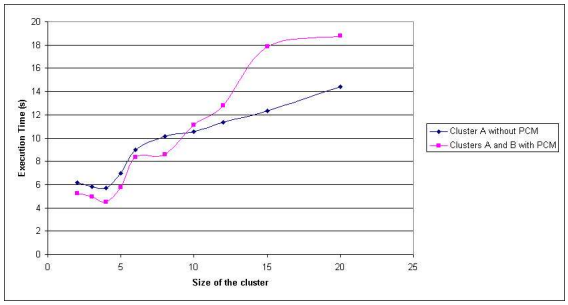

Fig. 4. MPI runs of the Heat Distribution program with and without PCM.

Our initial target applications for SALSA-MPI are parallel adaptive scientific computations. We cannot expect scientists to rewrite or even make significant modifications to extensive libraries of C and C++ software that currently use MPI. The SALSA-MPI architecture allows application programs to run using native C/C++ code and continue to use MPI for interprocess communication. Applications that provide the minimal required checkpointing functionality can immediately take advantage of SALSA-MPI functionality for dynamic resource allocation, process migration, and fault tolerance. Large-scale computations often already provide checkpointing functionality, so in many cases, this will be a minimal burden. Scientists may choose to provide more fine-grained checkpointing to allow their applications to improve the efficiency of the dynamic migration capabilities, particularly if the computing environments being used are very dynamic.

The current prototype implementation is still a work in progress and we are working towards making the SALSA-MPI framework a fully distributed system where MPI process or application reconfiguration are triggered by the IO middleware. The IO middleware should be able to dynamically trigger fine-grain or process migration when the computation to communication ratio is high and coarse-grain or whole application migration when this ratio is low. This will allow the SALSA-MPI framework to accommodate a wide range of scientific and engineering parallel applications.

References

1. Clark, K., Flaherty, J.E., Shephard, M.S. Appl. Numer. Math., special ed. on Adaptive Methods for Partial Differential Equations **14** (1994)
2. Remacle, J.F., Karamete, B., Shephard, M.: Algorithm oriented mesh database. Proc. 9th Meshing Roundtable, New Orleans (2000)
3. Remacle, J.F., Klaas, O., Flahery, J.E., Shephard, M.S.: Parallel algorithm oriented mesh database. Eng. Comput. **18** (2002) 274–284
4. Bottasso, C.L., Flaherty, J.E., Özturan, C., Shephard, M.S., Szymanski, B.K., Teresco, J.D., Ziantz, L.H.: The quality of partitions produced by an iterative load balancer. In Szymanski, B.K., Sinharoy, B., eds.: Proc. Third Workshop on Languages, Compilers, and Runtime Systems, Troy (1996) 265–277

5. Flaherty, J.E., Loy, R.M., Shephard, M.S., Szymanski, B.K., Teresco, J.D., Ziantz, L.H.: Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws. J. Parallel Distrib. Comput. **47** (1997) 139–152
6. Flaherty, J.E., Loy, R.M., Shephard, M.S., Teresco, J.D.: Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In Cockburn, B., Karniadakis, G., Shu, S.W., eds.: Discontinous Galerkin Methods Theory, Computation and Applications. Volume 11 of Lecture Notes in Compuational Science and Engineering., Berlin, Springer (2000) 113–124
7. Remacle, J.F., Flaherty, J., Shephard, M.: An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. SIAM Review **45** (2003) 53–72
8. Remacle, J.F., Shephard, M.S.: An algorithm oriented mesh database. Int. J. Numer. Meth. Engng. **58** (2003) 349–374
9. Musser, D.R., Saini, A., Stepanov, A.: STL Tutorial and Reference Guide: C++ Programming With the Standard Template Library. Addison-Wesley (1996)
10. Gropp, W., Lusk, E., Skjellum, A.: Using MPI. M. I. T. Press (1994)
11. Teresco, J.D., Beall, M.W., Flaherty, J.E., Shephard, M.S.: A hierarchical partition model for adaptive finite element computation. Comput. Methods Appl. Mech. Engrg. **184** (2000) 269–285
12. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Lecture Notes in Computer Science **2150** (2001) 1–25
13. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
14. Szymanski, B., Varela, C., Cummings, J., Napolitano, J.: Dynamically reconfigurable scientific computing on large-scale heterogeneous grids. In R. Wyrzykowski, *et. al.*, ed.: Proc. Parallel Processing and Applied Mathematics, Czestochowa, Poland, Springer-Verlag (2003)
15. Gosling, J., Joy, B., Steele, G.: The Java Language Specification. Addison Wesley (1996)
16. Bull, J.M., Smith, L.A., Pottage, L., Freeman, R.: Benchmarking java against c and fortran for scientific applications. In: Proceedings of ACM Java Grande/ISCOPE Conference. (2001) 97–105
17. Varela, C., Agha, G.: Programming dynamically reconfigurable open systems with SALSA. ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings **36** (2001) 20–34 http://www.cs.rpi.edu/˜cvarela/oopsla2001.pdf.
18. Desell, T., ElMaghraoui, K., Varela, C.: Load balancing of autonomous actors over dynamic networks. In: To appear in Proceedings of the Hawaii International Conference On System Sciences (HICSS-37). (2004)
19. Vadhiyar, S.S., Dongarra, J.J.: Srs - a framework for developing malleable and migratable parallel applications for distributed systems (2002)
20. Wilkinson, B., Allen, M.: Parallel Programming. Prentice Hall (1998)