Issues and Approaches in Parallel Multi-Component and Multi-Physics Simulations

Thomas J. Downar Rudolf Eigenmann José A. B. Fortes
Nirav H. Kapadia
Purdue University, West Lafayette, IN, U.S.A. *

Abstract *We discuss two large computational problems and identify the needs and opportunities to develop and run these applications across the world-wide infrastructure of computer systems and their interconnections. The applications include a nuclear reactor simulation and a seismic processing code. A new infrastructure being developed at Purdue University addresses the described needs. The infrastructure is an extension of the Purdue University Network Computing Hubs (PUNCH). It facilitates the development and execution of large, multi-component applications on globally distributed systems and supports remote collaboration.*

Keywords: multi-physics applications, distributed computing, Web operating system

1 Motivation

A vast parallel and distributed computational and communication infrastructure is currently emerging in the form of the Internet, the World-Wide Web, and its attached compute engines. This infrastructure is sometimes referred to as the Information Power Grid. We are only beginning to learn how to exploit its potential for solving large-scale computational engineering problems. On the application side, the demands for solving ever-larger problems are increasing at a similar speed. These demands come with expectations of running programs that consist of a coordinated set of geographically-distributed application components, and to perform simulations that encompass multiple problem areas. For example, a microscopic simulation of material behavior might be combined with the macroscopic simulation of a manufacturing process that uses such material; the two simulations may be developed and executed at different research sites and may exchange intermediate data via the Information Power Grid infrastructure.

This paper takes the viewpoint that we are in the initial stages of searching for methods to solve parallel and distributed multi-component and multi-physics problems. The paper presents issues from the viewpoint of two specific, large-scale applications in nuclear engineering and seismic processing, respectively. It then describes our approach for the development of an "Operating System for the Grid". This system is based on an operational Web-based infrastructure, called the Purdue University Network Computing Hubs (PUNCH). The paper presents an an overview of this system and outlines extensions that address the issues raised by the described applications.

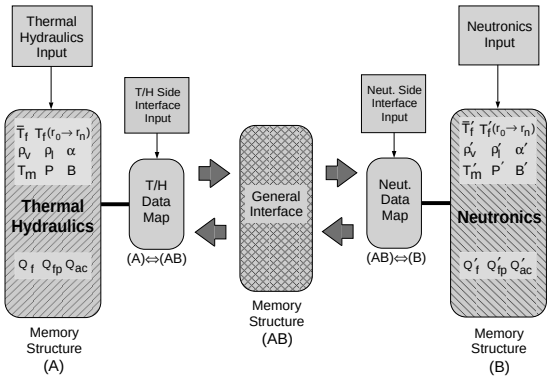

Figure 1: The General Interface in the framework of the coupled code

*This work was supported in part by NSF grants #9703180-CCR and #9872516-EIA. This work is not necessarily representative of the positions or policies of the U. S. Government.

2 RETRAN-3D: A Multi-Physics Nuclear Engineering Application

The RETRAN-3D [1] nuclear reactor simulation code has been developed over the last 30 years by the Electric Power Research Institute (EPRI) for the safety analysis of commercial Light Water Reactors. Today it is used to analyze the safe operation of many of the 400 commercial nuclear power plants in the world, which generate a significant fraction of the world's electricity. The user base for the RETRAN-3D code includes researchers and engineers from national laboratories, nuclear utilities, nuclear fuel vendors, and universities.

The code is written in Fortran and C and includes over 200,000 lines of source. Its development represents the investment of over a hundred man years effort. The code has recently been targeted for redevelopment to analyze the next generation of nuclear reactors. A primary motivation for this redevelopment is to update the physics of the code and to improve its computational efficiency.

RETRAN-3D is a multi-physics code in that it solves the coupled two fluid/temperature and neutron/nuclide field equations in a nuclear reactor. The primary safety analysis performed with the RETRAN-3D simulation is the possibility of loss of reactor core coolability and consequent fuel melting during a postulated reactor accident event. The solutions of the field equations required for practical problems typically leads to a mesh with thousands of nodes. The computational times for each transient scenario can exceed several hours using high end scientific workstations. The reduction of the computational burden for RETRAN-3D is a high priority, particularly with the recent desire to increase modeling fidelity (e.g. improved physics and 3-dimensional models). However, an equally important priority in the code redevelopment effort is to streamline maintenance of the code and to improve accessibility of the code by the user community. The following will briefly describe some of the efforts to improve computational performance and then describe efforts to redesign the code to improve its maintainability and accessibility.

2.1 Parallel Computing for Improving Computational Performance

Recent efforts to reduce the computational burden in RETRAN-3D have focused on the functional decomposition of the field equation solution and the domain decomposition of the spatial mesh [2]. The functional decomposition was achieved by assigning the solution of the temperature/fluid and neutron/nuclide field equations to separate processors with data exchange performed using the PVM message passing model [3]. Domain decomposition was achieved by using natural boundaries in the nuclear reactor system and employing a threads based multiprocessing model. For example, during the solution of the hydrodynamics matrix equations, a separate thread is created for the portions of the matrix corresponding to the nuclear reactor core and each of the four loops used to transport fluid to the steam generators. This decomposition is reasonably efficient because of the weak coupling between each of these five components. One of the most challenging aspects of mixing message passing and threads parallel programming models in RETRAN-3D has been the development of algorithms to schedule processors and to optimize data access patterns. The target platforms for the RETRAN-3D code are high-end multiprocessor workstations.

The RETRAN-3D code integrates two simulation components, solving the fluid/ temperature and the neutron/nuclide field equations, respectively. There is a clear separation between the two components. They are based on independent codes that exchange messages via a well-defined interface [5] at each time step of the simulation. Figure 1 shows the component structure and the interface. The interface module is responsible for the conversion between units used in the two components, for the mapping between the different node structures, and for data packing into messages to be sent and unpacking of received messages.

2.2 The Motivation for Network and Distributed Computing

The modularization of RETRAN-3D and the use of a general interface module provides the potential of improved maintainability and accessibility of the code without compromising code security. Because the RETRAN-3D code was developed and is maintained by EPRI for the analysis of commercial light water reactors, it contains a large amount of proprietary data and security of the code is an important consideration. The maintenance of RETRAN-3D is a considerable task with various responsibilities distributed among national laboratories, universities, and private contractors. The current administration of the code involves the prior clearance of prospective users and the subsequent distribution of the code with an installation package on a CD ROM. The burden for compilation and execution of the code falls entirely on the user and any available local software support. Modifications to the RETRAN-3D can occur as frequently as each quarter, which often requires a redistribution of the entire code package.

There are several opportunities for the RETRAN-3D multi-physics application to exploit the global, distributed compute infrastructure.

1. Facilitating remote collaboration: New methods for access and administration of the RETRAN-3D code can be developed. Instead of the current distribution scheme, the code could be maintained separately at the institution with primary responsibility for the development of the respective component. Access to the other institutions would be provided through a shared network facility. This facility would provide remote collaboration tools that allow new users to ask questions, "meet" with the developers, exchange ideas, and work on common problems. Novice users and analysts could run experiments that can be monitored and assisted directly by the experts at the code's home site.

2. "Net-make": Given that the home sites of the different code components are at different organizations, facilities must be provided that support the building process of a new simulation. Code components must be fetched from their home site, and the availability of a new version of a code component must be made known. Basic mechanisms to create this facility would include posting mechanisms for new components and screening mechanisms that would test for compatibility of a new component version with the previous one. The latter may include a description of component updates and guides for the adaptation of the interface.

3. Distributed execution: Executing the simulation at different sites has two objectives. First, it is natural to execute a code component at its home organization. This will allow the local experts to use better monitoring facilities (e.g., use advanced instruments that don't have network support) and to strictly protect access to the code. The network infrastructure thus would facilitate the communication among the code's components between the respective home sites. It would identify the appropriate communication media for a particular simulation run and perform the mapping of the code's interface descriptions onto the interconnection system. Second, distributed execution can take advantage of available machine resources, hence improving the code's execution performance. In this scenario the network infrastructure will, upon an execution request, identify both the appropriate compute nodes and the communication media. It will then map the computation and communication links to this configuration and route input/output to the user's access points.

There are a number of issues arising in such scenarios. A remote collaboration environment will need to provide secure, shared file spaces for collaborating groups at different geographical locations. It will be natural for such an environment to take advantage of existing facilities, such as the *Netmeeting* utility. Hence the integration of existing tools is an issue. "Netmake" facilities will need to support local and global scenarios. A *local scenario* would build the application component at a particular site, accessing the already existing components at the remote organization via a well-defined interface description. A *global make* would build all components for an entire sim-

ulation. This will require appropriate bookkeeping and notification messages going to the respective developer sites, with network directories keeping track of the code versions and their coherency. Support for distributed application execution will require basic facilities for message passing across distributed compute nodes. The actual properties of these nodes and their interconnection is not known at development time. It needs to be determined and the code needs to be adapted prior to the execution or – in an advanced scenario – dynamically at runtime. Such adaptation can include the switch from message-passing communication to shared-memory data exchange, the recompilation of a code for a new target architecture, and the dynamic exchange of new algorithms that are more appropriate for the selected machine organization.

3 Seismic: A Multi-Program Application Suite for Oil and Gas Exploration

The *Seismic* application is used in the seismic processing industry for the search of oil and gas [4, 5]. The code represents an actual suite of four applications, each of which is called a *component*. They perform the seismic processes named *data generation, data stacking, time migration*, and *depth migration*, respectively. The entire code contains 20,000 lines of Fortran and C, which can execute in parallel on a shared- or distributed-memory system. A Seismic execution involves intensive communication as well as intensive disk I/O.

Seismic integrates multiple capabilities into a single code, as is typical for large computational applications. Program input data selects the specific functionality that is to be performed in a particular run. A typical seismic processing run executes all four components. Hence, it activates the Seismic code four times. There is a partial order among the components, i.e., some of them have to run sequentially while others can run concurrently. The components have distinct characteristics, which will become important for exploiting potentially heterogeneous, distributed compute resources. For example, *data generation* is a highly parallel seismic process, while *depth migration* is very communication-intensive.

The communication between the component executions is via disk files. Each component reads the input data from a file, processes it, and writes its results to disk again. This is a practical way of coupling several program executions in a traditional, sequential environment, because it does not require any special support for communication between different programs. However, it puts high demands on the performance of the disk IO subsystem, and it requires each application phases to complete before the next one can start.

There are three areas where the Seismic processing suite can take advantage of a parallel and distributed information infrastructure.

1. Parallel execution of Seismic components: In the current configuration, the Phases *data generation* and *data stacking* must be run sequentially, while *depth migration* and *time migration* can be performed concurrently. Exploiting such *component parallelism* can lead to significant performance improvements.

2. Exploiting heterogeneous machines: the different communication characteristics of the individual components call for different target machines. *Data generation* could be run on a highly-parallel distributed-memory architecture, while the intense communication of *depth migration* necessitates a tightly-coupled machine. These characteristics can be taken advantage of in a system that manages a global information infrastructure with different types of compute nodes.

3. Component communication: currently, the four components of Seismic communicate through disk IO, as described above. More advanced forms of inter-program communication could lead to significant performance improvements. Conceptually, the phases communicate in a form similar to UNIX pipes, where the intermediate data can take the form of a disk file, a memory buffer, or a message transmitted from producer to consumer component. The concrete form of this medium could be defined dynamically by the global runtime system.

A number of issues arise in these scenarios. A language needs to be available that

allows the developer to define the parallelism structure, the component attributes, and interconnections. For example, in Seismic one would specify that Component 1 is followed by Component 2, which is followed by a possibly concurrent execution of Component 3 and 4. The attributes of the phases will be given (e.g., weak communication in Component 1, strong communication in Component 2) and the component interconnection is specified (e.g., communication through files or "network pipes"). The attributes have to be understood by an operating system that is aware of the global information infrastructure. This operating system will map the application components to the appropriate compute nodes and establish the necessary communication links. The attributes can be generated by hand or via automatic tools. In both cases, performance prediction capabilities will be important, which can determine the characteristics of the applications and how they impact the runtime behavior.

4 A Web-based Operating System for Parallel and Distributed Multi-Program Applications

We have begun the development of a software infrastructure for the execution of concurrent metaprograms and network applications. This system is an extension of the Purdue University Network-Computing Hubs (PUNCH) [6, 7, 8], which has been operational for over four years, and has been widely used for running individual (sequential) applications. PUNCH is a Perl-based infrastructure (with about 13,000 lines of code) that can be viewed as an operating-system for the World-Wide Web. It provides: (1) transparent and universal access to remote programs and resources, (2) access-control (privacy and security) and job-control (run, abort, and program-status) functionality in a multi-user, multi-process environment, and (3) support for logical organization and management of resources. PUNCH allows users to: (a) upload and manipulate input-files, (b) run programs, and (c) view and download output — all via standard WWW browsers.

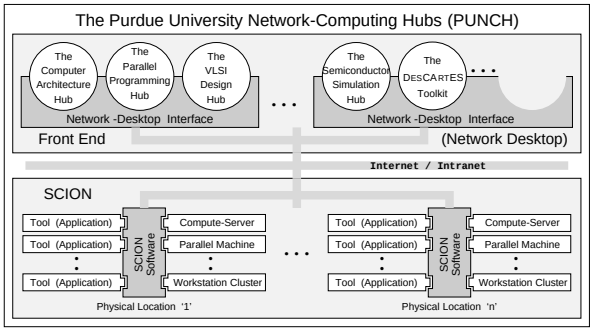

Figure 2: The PUNCH infrastructure is made up of two parts: the front-end (network desktop) and SCION. Content and resources can be "plugged" into the infrastructure, as shown.

Figure 2 shows a simplified view of PUNCH's organization. PUNCH consists of the front-end, called the network desktop, and the back-end, called SCION for Scalable Infrastructure for On-demand Network computing. The network desktop consists of one or more logical groupings of programs. Each grouping is called a "hub". SCION manages the access and control of distributed resources and tools (i.e., programs) to serve requests from the front-end. A resource can be an arbitrary platform, and can be added incrementally by specifying its architecture (make, model, OS, etc.) and starting a server on it. A new program can be added by providing to PUNCH information about the locations where the program resides, input/output behavior, machines where it can run, and its logical classification in the hub. PUNCH can be "experienced" at http://www.ecn.purdue.edu/labs/punch.

The issues described earlier can be addressed by building a Web-based operating system on top of systems that provide low-level distributed computing services for heterogeneous computing environments (e.g., Globus [9, 10], Legion [11, 12]). For example, Globus consists of a collection of layered services that can be utilized by vertically integrated solutions such as PUNCH to address key technical issues that arise in a network computing environment. In the proposed Web-based operating system, an application will be defined as a set of logi-

cally grouped programs. These applications are specified using *scripts*. The operating system will manage and execute each component of the application independently using macro-dataflow semantics. These components can potentially reside and execute on different systems across geographical distances, while appearing as a single entity to the user and to higher-level software.

The script will be processed by an interpreter. Each component in the application is essentially a program that accepts some input and generates some output; some components process data, while others simply serve as translators for incompatible data-formats. The work of the interpreter will be performed in three phases, as described below.

In the first phase, the interpreter will analyze the data-dependences between the application components, and will generate the *component-graph*, an annotated macro-dataflow task-graph. The annotations will contain meta-information that will be used for run-specific resource analyses that allow the network-computing infrastructure to best match the requirements of each task with the underlying network-accessible resources. For example, an annotation corresponding to CPU usage prediction could contain: (1) an analytical (complexity) expression that would be exploited to compute a CPU-time estimate on the basis of specified parameters (e.g., matrix sizes, in the case of matrix manipulation codes), or (2) a list of parameters whose values would be used by a machine learning system to observe and *learn* the behavior of the component. The annotations will also include information about the relationship between the different components in the task-graph. For example, they could indicate that a collection of components within the component graph make up a concurrent program of communicating sequential processes (e.g., a collection of MPI-based programs). They will also include attributes of the communication between components (e.g. synchronous, asynchronous or pipelined) This process makes up the *resource-resolution* phase of the on-demand network-computing technology.

In the second phase, the network computing infrastructure will invoke the *resource-allocation* phase. This phase will involve the run-time selection of physical resources. The resource-allocation and management module will maintain dynamic, parameter-keyed pools of managed resources which are periodically updated by an on-line resource-monitoring sub-system. For scalability reasons, these pools are used as *resource-caches* that consist of a relatively small number of resources (the current *working-set*); new resources (from the list of available resources) are asynchronously added to the pools as the working-set is depleted.

Finally, in the third phase, the network-computing infrastructure will dispatch each component of the application to the allocated resource; any existing parallelism between tasks will be automatically exploited by the component-graph. As tasks complete, the associated dispatcher module will retrieve the generated output and dispatch any tasks that are dependent on that output. For the purposes of the network-computing infrastructure, each allocatable resource consists of a network-accessible daemon that accepts requests for program execution. Depending on the local system configuration, a daemon may independently execute the request on the same machine, or it may translate and submit the request to an independent resource-management system (e.g., Condor [13]) for execution. This ability to support independent, recursive replication of each sub-system translates to a highly scalable and partitionable network-computing infrastructure.

The specific tasks needed to accomplish the described goals include designing a coordination language, the associated translator (to convert the specification into an efficient internal format), and the interpreter (to process the meta-application at run time). In addition, middleware and methods will need to be developed to co-schedule (e.g., gang-scheduling) modules of a module-graph, interface with resource schedulers (e.g., the internal scheduler of a multiprocessor machine), aggregate resources (e.g., for an MPI program module), and incorporate virtual resources (e.g., a PVM-based cluster of geographically dis-

tributed workstations). The underlying on-demand computing infrastructure provided by PUNCH remains largely unaffected, except for the addition of a meta-application interpreter; because each component of a meta-application is itself an application (or a meta-application), the added functionality can be elegantly supported by a simple recursive use of the basic on-demand computing infrastructure.

5 Conclusions

An "Information Power Grid" is currently emerging in the form of a new world-wide infrastructure that can be used to collaborate remotely, develop complex computer applications, and execute these applications in a globally distributed organization. Today, we are in the concept stages of exploring the potential offered by this vast infrastructure. The present paper has contributed to this process by identifying opportunities of two large computational applications to exploit this potential, and by presenting a new "operating system for the Web" that can exploit these opportunities. It builds on an operational system, called the Purdue University Network Computing Hubs (PUNCH). Extensions that provide more support for developing and running multi-component applications across "the Grid" and facilitate remote collaboration are planned.

References

[1] J. McFadden, et al.. RETRAN-03: A program for transient thermal-hydraulic analysis of complex fluid flow systems. EPRI NP-7450, 1992.

[2] T. Downar, J. Y. Wu, J. Steill, and R. Janardhan. Parallel and serial applications of the RETRAN-03 power plant simulation code using domain decomposition and Krylov subspace methods. Nuclear Technology, Vol. 117, February 1997.

[3] A. Geist et al.. PVM: Parallel Virtual Machine MIT Press, Massachusetts, 1994.

[4] C. C. Mosher and S. Hassanzadeh. ARCO seismic processing performance evaluation suite, user's guide. Technical report, ARCO, Plano, TX., 1993.

[5] Rudolf Eigenmann and Siamak Hassanzadeh. Benchmarking with real industrial applications: The SPEC High-Performance Group. *IEEE Computational Science & Engineering*, 3(1):18–23, Spring 1996.

[6] Nirav H. Kapadia, Mark S. Lundstrom, and José A. B. Fortes. A network-based simulation laboratory for collaborative research and technology transfer. In *Proceedings of the 1996 Semiconductor Research Corporation's Technical Conference (TECHCON'96)*, Phoenix, Arizona, September 1996.

[7] Nirav H. Kapadia, José A. B. Fortes, and Mark S. Lundstrom. The Semiconductor Simulation Hub: A network-based microelectronics simulation laboratory. In *Proceedings of the 12th Biennial University Government Industry Microelectronics Symposium*, pages 72–77, July 1997.

[8] Nirav H. Kapadia and José A. B. Fortes. On the design of a demand-based network-computing system: The Purdue University Network-Computing Hubs. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98)*, pages 71–80, Chicago, Illinois, July 1998.

[9] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2), 1997.

[10] Ian Foster and Carl Kesselman. The Globus project: A status report. In *Proceedings of the 1998 Heterogeneous Computing Workshop (HCW'98)*, pages 4–18, 1998.

[11] Andrew S. Grimshaw, Anh Nguyen-Tuong, and William A. Wulf. Campus-wide computing: Early results using Legion at the university of virginia. Technical Report CS-95-19, Department of Computer Science, University of Virginia, March 1995.

[12] Andrew S. Grimshaw, William A. Wulf, et al. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[13] M. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.