Using TOP-C and AMPIC to Port Large Parallel Applications to the Computational Grid

Gene Cooperman* Henri Casanova†‡
Jim Hayes†‡ Thomas Witzel*

∗ College of Computer Science, Northeastern University.
{gene,twitzel}@ccs.neu.edu
† Computer Science and Engineering Department, University of California, San Diego.
{casanova,jhayes}@cs.ucsd.edu
‡ San Diego Supercomputer Center, University of California, San Diego.

Abstract

Porting large applications to distributed computing platforms is a challenging task from a software engineering perspective. The Computational Grid *has gained tremendous popularity as it aggregates unprecedented amounts of compute and storage resources by means of increasingly high performance network technology. The primary aim of this paper is to demonstrate how the development time to port very large applications to this environment can be significantly reduced. TOP-C and AMPIC are software packages that have each seen successful application in their respective domains of parallel computing and process creation/communication. We combine them to implement and deploy a master-worker model of parallel computing over the Computational Grid. To demonstrate the benefit of our approach, we ported the 1,000,000 line Geant4* sequential *code in three man-weeks by using our TOP-C/AMPIC integration. This paper evaluates the benefits of our approach from a software engineering perspective, and presents experimental results obtained with the new implementation of Geant4 on a Grid testbed.*

1 Introduction

The problem of porting large parallel applications to new distributed computing platforms poses many software engineering challenges. The common approach is to define generic programming models that can accommodate ranges of applications. Each programming model must be enabled on the target platform thanks to the specification of an Application Programming Interface (API) and the implementation of that API with available software technology. An example is the Message Passing Interface (MPI) [4], which provides a generic *message passing programming model* for distributed memory platforms. MPI comes with an extensive API and has been implemented on many platforms [5].

Even though the message-passing model is fundamental for many scientific applications, it is possible to provide higher levels of abstraction in order to describe the overall structure of classes of applications of distributed computing platforms. The advantage is that higher abstractions are more directly usable by a scientist as they hide underlying mechanisms on the computing platforms (such as messages). A popular abstraction is the *master-worker* paradigm which is applicable when a problem can be divided into a number of smaller independent *work units* that a master process distributes to many worker processes. Master-worker computing has been successfully applied to many application domains and has been the object of a number of research endeavors (e.g. see [24] for a comprehensive survey of applications and techniques).

The TOP-C project [12] provides a simple master-worker model for applications implemented in Single Program, Multiple Data (SPMD) fashion. Like MPI, it provides an API that can be used to write applications. Since the master-worker model is more abstract than a pure message passing model, TOP-C's API is simpler and provides fewer but richer primitives [8, 9, 12]. The

This work is supported by the National Science Foundation under grants ACI-9872114 (*) and ASC-9619020 (†‡).

power of the TOP-C abstractions was demonstrated in [7, 11, 14, 15, 33].

Among today's available distributed computing platforms, the *Computational Grid* (or *Grid*) [19, 22] has gained tremendous popularity. It aggregates large amounts of compute and storage resources over the wide area by means of increasingly high performance network technology. Therefore, it promises to achieve unprecedented levels of performance for many scientific applications. In order to enable Grid computing, it has been recognized that a number of basic middleware services must be provided for issues such as authentication/security, information, resource access, data management, etc. A precise set of such services has been identified and corresponding software was developed as part of the Globus project [23]. Those services are being generalized in the context of the Global Grid Forum [3].

Given those technological advances, it is necessary that adequate programming models be provided for developers to target applications to the Computational Grid. In that context, the master-worker model provided by TOP-C is particularly relevant, since one of its design goals is high latency tolerance. The TOP-C model also provides a natural load balancing. In order to enable TOP-C to make use of the emerging Grid infrastructure, it is necessary to re-engineer its implementation so that it can use Grid services. Rather than implementing TOP-C directly on top of Globus services, we advocate the use of an intermediate software layer: AMPIC.

AMPIC [1] provides a layer of abstraction on top of fundamental Grid services to implement the concepts of remote process creation and inter-process communication. In addition, AMPIC also provides those same abstractions for traditional parallel computing technology, such as MPI [4] or PVM [26], shared memory architectures, as well as basic mechanisms such as Secure Shell and sockets. Like TOP-C, AMPIC's focus is on simplicity and ease of use and has been successfully used in several applications [2, 24, 31]. The goal of this paper is to demonstrate that the integration of TOP-C and AMPIC makes it possible to port master-worker applications to the Grid in a way that: (i) requires minimal software engineering effort; and (ii) exploits existing Grid technology such as Globus services.

In order to demonstrate and validate our approach we take as a case study the Geant4 particle physics application [25]. Geant4 is a 1,000,000 line library produced by a team of over 100 developers from around the world and coordinated by a core development team at CERN. This test case is realistic since Geant4 was originally conceived purely as a sequential application, with no thought for parallelization.

This work is related to numerous works targeting master-worker applications. Among recent such efforts in the context of the Computational Grid are the AMWAT project [2], and the MW project realized as part of Condor [28, 29]. Those projects have address scheduling and resource management issues that are directly applicable to the TOP-C framework and that we will examine in the next phase of this work. In terms of providing a framework for enabling master-worker applications, this work makes the following contributions. TOP-C provides a richer abstraction that contains notions of shared data, non-trivial parallelism, high latency tolerance, and application robustness and resilience [12]. This work demonstrate how the integration of TOP-C with AMPIC makes the TOP-C model directly applicable to Grid computing on existing resources running a variety of software services.

Even though the MW project [28, 29] was originally Condor-centric, it also partially aims at supporting a variety of underlying mechanisms (e.g. Globus). We claim the we have achieved that goal thanks to the use of AMPIC with only minimal software engineering efforts. From that standpoint, aspects of this work would be eminently applicable to the MW project. A related issue is that our approach could use resources managed by systems like Condor or Legion [27]. This would only require that AMPIC support remote process creation mechanisms of those systems. Such AMPIC developments are not currently under way but would be straightforward. Other avenues that could be explored are the reimplementation of the TOP-C programming model in Java, where it could take advantage of the MANTA [32] support for wide-area computing and direct support for objects.

This paper is organized as follows. In Sections 2 and 3 we present the TOP-C and AMPIC models and software implementations. In Section 4 we describe our software integration effort in the context of the GEANT application. Section 5 contains preliminary experimental results and Section 6 concludes the papers and highlights future directions.

2 The TOP-C model

TOP-C is free, open source software [13]. It is designed with the goals of

- following a simple, easy-to-use, general programmer's model,
- supporting easy parallelization of existing sequential code, and

- having high latency tolerance.

The TOP-C (Task Oriented Parallel C/C++) model was developed almost ten years ago. It has been ported to LISP [8], to C and C++ [9] and to GAP [10]. It provides a uniform interface which can execute both in a distributed and shared memory model [12]. Source code for a TOP-C application can be compiled and linked with the appropriate TOP-C library to run on a shared memory machine using POSIX threads, or to run on a cluster using either MPI or TCP/IP sockets.

TOP-C also provides a sequential version of the library. Typically, a TOP-C application is developed first as a sequential application. The traditional sequential debugging tools (symbolic debugger, etc.) are used to ensure that the application is running correctly. Then the application is re-linked with one of the TOP-C parallel libraries.

The TOP-C model is based on a master/worker model with a Single Program, Multiple Data (SPMD) implementation. Each process operates independently until a call to `TOPC_master_slave()` which serves as a point of synchronization. When all processes have reached this routine, the thread of control is passed to TOP-C, and TOP-C invokes application-defined callback routines as needed.

The TOP-C programming model is based on three concepts: the task, the shared data, and the action (See Figures 1, 2, and 3).

A TOP-C application depends on four application-defined callback functions:

1. `TOPC_BUF GenerateTaskInput()`;
2. `TOPC_BUF DoTask(void *taskInput)`;
3. `TOPC_ACTION CheckTaskResult(void *taskInput, void *taskOutput)`; and
4. `void UpdateSharedData(void *taskInput, void *taskOutput)`.

Pointers to these four callback functions are passed as arguments to `TOPC_master_slave()`. When `TOPC_master_slave()` executes, TOP-C receives the thread of control. `GenerateTaskInput()` is then invoked from the master process. Its output is a buffer, `taskInput`. `DoTask()` is invoked from a worker process. Its output is a buffer, `taskOutput`. TOP-C does not do any marshaling of the task input and output buffers. Marshalling is left to be implemented in one or more independent library, some of which may be application-specific.

In general, non-trivial parallelism is implemented through other possible actions, including `UPDATE` (invoke `UpdateSharedData()` on all processes) and `REDO`

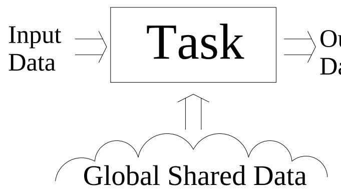

Figure 1. TOP-C Concept #1: The task abstraction

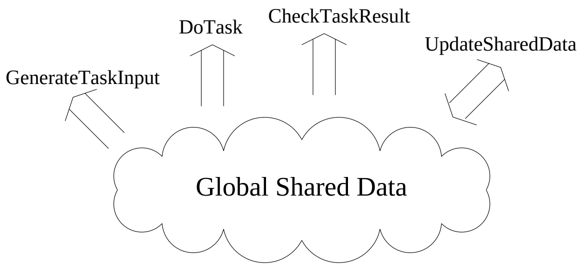

Figure 2. TOP-C Concept #2: The shared data abstraction

(invoke `DoTask()` again with the original `taskInput` on the original worker process, but using the most recent version of the shared data). Provision is made for more efficient execution of `REDO`, since the worker process can can store intermediate results from the previous `DoTask()` computation and re-use those results in the second computation. For a more detailed description, see the TOP-C home page [13].

3 The AMPIC middleware

The AppLeS Multi-Protocol Interprocess Communication (AMPIC) [1] package addresses the two following fundamental issues:

1. inter-process communication,
2. remote process management.

Many software projects have addressed those issues for the deployment of distributed computing applications. The goal of AMPIC is not to provide yet another set of mechanisms, but rather to provide a common and easy-to-use interface to existing mechanisms.

AMPIC was originally developed as part of the AppLeS Master-Worker Application Template (AMWAT) project [2, 24] which is related to TOP-C in the following way. Like TOP-C, AMWAT targets master-worker

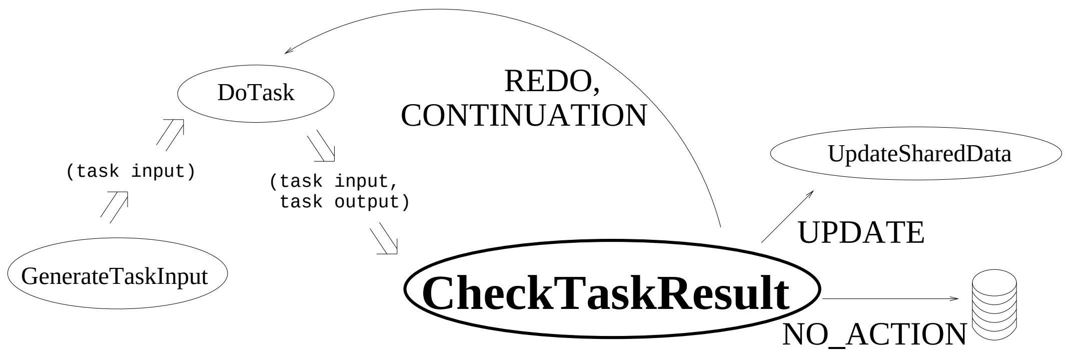

Figure 3. TOP-C Concept #3: The action abstraction

applications. However, AMWAT's focus in on scheduling whereas TOP-C's focus is on providing a flexible and powerful programming model. In fact, the scheduling techniques developed as part of AMWAT would be eminently applicable as part of the TOP-C software (see Section 6). Very early, AMWAT developers recognized that the communication and process management component of the software could be abstracted as a completely separate package, which became AMPIC.

AMPIC consists of 4,000 lines of C and is currently freely available at [1]. AMPIC provides fundamental functions as part of its API to identify communication peers, transfer data, and start remote processes: `AMPIC_MyId()`, `AMPIC_Send()`, `AMPIC_Recv()`, `AMPIC_Spawn()`. AMPIC also contains several other functions that provide finer levels of control. The AMPIC API was defined by inspecting many available technologies in the parallel computing community and converging towards a common denominator. In its current version, AMPIC allows applications to use the following underlying technology transparently: MPI [4], PVM [26], Globus [23], sockets, Secure Shell, and shared memory. In addition, AMPIC allows applications to use a mix of the above technologies simultaneously. AMPIC has been ported and tested on all major flavors of UNIX. In addition, AMPIC has been ported to the Windows operating systems, where it can use shared memory, sockets, and Secure Shell.

Even though the initial goal for AMPIC was to be an in-house software tool used only for AMWAT, it quickly became apparent that the AMPIC model answers the needs of other application developers. This is demonstrated in the next section, where we describe our software integration effort.

4 Software Integration

The software generated as part of this work was developed using TOP-C version 2.4.0, Geant4 version 4.3.2, AMPIC version 1.2, and Globus version 1.1.3. The TOP-C and AMPIC packages are written in C, while Geant4 consists of 1,000,000 lines of C++ based on strict software engineering principles.

This work took three large software projects that had been independently developed, and integrated them into a single binary application capable of running both over the Globus toolkit and over a cluster using TCP/IP sockets. Using the TOP-C –seq switch, it can also run as a single process for ease of debugging during development. The total effort to combine these packages was approximately one man-month, of which the majority of the time was spent in understanding and parallelizing the Geant4 software design. In part, this work demonstrates the ability to rapidly integrate these disparate software packages. The natural software architecture for this work consists of the layers depicted in Figure 4.

Geant4 Application

Geant4

TOP-C

AMPIC

Globus Toolkit

Figure 4. Integration of Software Layers

4.1 Integration of Geant4 with TOP-C

Geant4 (GEometry ANd Tracking) [6, 25] is a simulation package for particle-matter interaction developed at CERN. Although initially developed to assist in collider design, its use has spread to many areas of physics, engineering and biomedicine. Including the associated toolkits, Geant4 contains approximately 1,000,000 lines of C++ code. The Geant4 software was developed by RD44, a world-wide collaboration of about 100 scientists participating in more than 10 collider experiments in Europe, Russia, Japan, Canada and the United States. Many specialized working groups are responsible for the various domains of the toolkit.

An early parallelization of Geant4 using TOP-C was reported on in [7]. That work was carried out independently of the Geant4 developers with the single goal of simulating cosmic ray showers over Geant4. It is notable as being the first successful parallelization of Geant4 sufficient to execute large simulations. Approximately four man-weeks of software development were required. After meetings and discussions with the Geant4 team in July, 2001, a second, independent parallelization of Geant4 using TOP-C was executed. This time, the goal was to leave the Geant4 kernel library unmodified. The total development time was approximately three weeks.

The changes were created by writing a distinct library that was linked with the Geant4 library. This involved writing approximately 200 new lines of code for a new derived class to shadow existing virtual functions in Geant4, and then writing approximately 250 lines of independent code needed for marshaling the Geant4 data structures. The new derived class replaced Geant4's own work queue with a call to TOP-C routines to manage the work queue.

Once the combination of Geant4 and TOP-C (ParGeant4) was working correctly in sequential mode, it was re-linked with the TOP-C distributed memory library over sockets. Initial experiments were carried out on the CERN cluster of 50 Pentium III CPUs running at 550 MHz. The measured speedup was approximately 10 times, and it ran in 29 seconds. There were 20,000 TOP-C tasks. Hence each task took approximately 72.5 ms ($50 \times 29/20,000$ ms). The measured CPU load on the master process was only 18%. It is an important confirmation of the TOP-C methodology, which dictates that all CPU-intensive operations should be executed on worker processes via the application callback routine, `DoTask()`.

The speedup could have been further improved by running multiple worker processes on each processor, in order to better overlap communication and computation. Another technique is to merge multiple messages for task input. However, these techniques were not used in this initial experiment measuring the raw communication overhead.

4.2 Integration of TOP-C with AMPIC

Approximately one man-week was devoted to integration of TOP-C with AMPIC. The primary coding for this phase of the project was carried out by the fourth author, an undergraduate student. The elapsed time from the beginning to the end of this phase was two weeks.

Initially TOP-C/AMPIC was tested using the AMPIC sockets mode. Once TOP-C/AMPIC was correctly running simple TOP-C applications over sockets, we then quickly ported ParGeant4 (Geant4/TOP-C) to run using AMPIC over sockets. There were no problems experienced in this phase.

Note that integrating TOP-C with AMPIC provides the following advantage from the software engineering standpoint: AMPIC provides a cleanly separated component, which is in charge of the logistics of remote process management and inter-process communication. Note that the AMPIC's functionality is comparable to that of Globus' Nexus [21] and GRAM [17]. As a matter of fact, AMPIC can use GRAM as a mechanism and could potentially support Nexus communication. The main argument in favor of AMPIC for this work is its simplicity, which made the software integration straightforward. An added benefit of integrating TOP-C with AMPIC is that TOP-C now benefits from AMPIC developments "for free". This was key in achieving the use of Globus services and therefore in getting TOP-C to work in Computational Grid environments. TOP-C directly supports MPI [4], sockets using Secure Shell, and has a shared memory mode. AMPIC supports all of these mechanisms, and PVM [26], and also simultaneous combinations of the above mechanisms.

The combined software (Geant4/TOP-C/AMPIC) was tested with Globus on a Grid testbed as described in Section 5. This step required a half day of debugging as misconceptions of the fourth author about the proper invocation of AMPIC were quickly corrected by the third author (maintainer of AMPIC).

Finally, in tests over our Grid testbed, the Geant4/TOP-C/AMPIC application was tested for the first time in a situation in which there was no shared file system. An issue came up because the location of the application executable was given by different absolute path names on different administrative domains. TOP-

C and the Globus toolkit each have different mechanisms for specifying the directory of a remote process. For this experiment, we inserted code in the application to directly change directories. A future version will include some combination of the TOP-C and Globus mechanisms.

5 Experimental Results

The main goal of this paper is to demonstrate how the integration of TOP-C and AMPIC reduces the software engineering effort required in parallelizing and porting master-slave applications to the Computational Grid. This has been demonstrated in the previous section. In order to further validate our approach we performed two sets of experiments on which we report here.

5.1 Small Geant4 Runs

This set of experiments was designed to estimate some of the overhead due to using Grid middleware. We used a Grid testbed consisting of 8 hosts at the University of California, San Diego, and 8 hosts at the University of Tennessee, Knoxville. The testbed was used in non-dedicated mode and we performed back-to-back application runs in order to ensure some degree of reproducibility. Each host in the testbed was a single processor node running Linux and Globus version 1.1.3. All results presented in this section are averages over 10 application runs.

# procs	Average elapsed time (s)	Average busy time (s)	Average spawn time (s)
2	272.8	267.8	5.0
4	143.7	133.2	10.5
7	87.6	66.6	21.0
16	95.5	33.9	61.6

Table 1. Execution time breakdowns for small Geant4 runs on the Grid testbed.

We performed a set of runs of the Geant4 application with 2, 4 and 8 hosts on the UCSD system, and with 16 hosts using all UCSD and UTK resources. Obtained results are shown in Table 1. Experiments were run for a total of 16 Geant4 work units. In all those experiments, the spawning of the TOP-C processes is performed from a host on the UCSD system. That node also participates in the computation (spawning a TOP-C process on itself). For each number of processors the table shows the average elapsed time for the application, the actual "busy time", as well as the "spawn time", that is the time to start all remote processes on Grid resources. The elapsed time is the sum of the busy time and the spawn time. Note that the busy time includes the time spent in inter-process communication. In these experiments, standard deviations for all numbers in Table 1 were below 2 seconds.

The busy time decreases nearly linearly as the number of processors increases. This is due to the fact that in this Geant4 run, communication was not a bottleneck for the application. It is interesting that the spawn time for these results is significant. It is between 2.5 and 3.0 seconds for hosts on the UCSD system, and between 5.0 and 6.0 seconds for hosts on the UTK system.

This spawning cost negates the benefit of using 16 hosts for this Geant4 run as it leads to a performance decrease of 9% over using 8 hosts. Note that here that behavior is *solely* due to the overhead incurred when accessing and acquiring Grid resources (public-key authentication with the Globus Security Infrastructure (GSI) [20] and remote process creation with the Globus Resource Allocation Manager (GRAM) [17]). The application executable and required data files were prestaged onto the resources and thus caused no additional network overhead. Note also that in these experiments we do not access Grid information services (such as the MDS [16]) but instead cache that information as it rarely changes. Work performed as part of the GrADS project [30, 18] has quantified the overhead and has shown it to be significant with current implementations of Grid information services. Local caching is indeed the approach of choice. Our results further confirm that Grid middleware overhead can be extremely significant for relatively short-lived Grid application runs such as short Geant4 runs.

5.2 Larger Geant4 Runs

Using the same testbed as in the previous section, we executed larger Geant4 runs with between 4 and 16 hosts. We performed 2 sets of such runs on Feb. 22nd 2002, and 3 sets on Feb. 25th 2002, for a total 10 hours of runtime. Each observed execution time is shown as a single data point on Figure 5. The average execution time for a given number of processors over the 5 sets is shown, and solid lines show linear fits for the UCSD runs (under 8 hosts) and the UCSD+UTK runs (over 9 hosts). The use of remote hosts at UTK leads to some improvement, but is limited by the available Internet bandwidth over the wide-area. Overall, running the

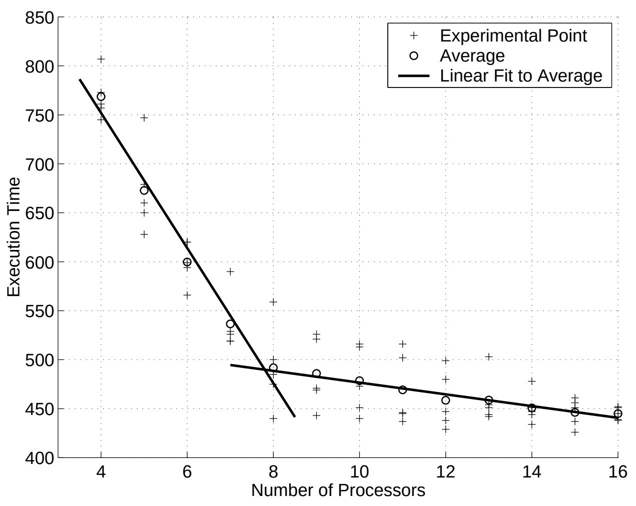

Figure 5. Execution times for larger Geant4 runs on the Grid testbed.

application on 16 hosts at UCSD and UTK leads to a relative average improvement of 11% over running the application only on 8 hosts at UCSD. Note that we purposely used a Grid testbed with relatively poor connectivity (we observed around 1.5 Mbps over the wide-area during those experiments). This indicates that Geant4 would efficiently run on a more tightly connected Grid, e.g. on a campus.

The results in this section show that our work enables realistic application runs over a wide-area testbed using Globus. Most importantly, this is accomplished completely transparently to the user thanks to the TOP-C/AMPIC integration. Note that in this paper we only addressed the software engineering aspect of Grid application development and that we did not address the issues of application performance tuning and scheduling. We highlight such research directions in the next section.

6 Conclusion and Future Work

The main focus of this paper was on the integration of TOP-C, AMPIC, and Geant4, and how that integration provides an elegant and easy way to parallelize and port classes of applications to the Computational Grid using Globus resources. The TOP-C/AMPIC integration appears to hold the potential for rapidly porting classes of large sequential applications to the Grid. In our experiments we primarily used the Globus toolkit for spawning remote processes, after which TCP/IP sockets were employed for communication.

Note that this initial implementation did not use the more advanced features of TOP-C, which support non-trivial parallelism, high latency tolerance, checkpointing, natural load balancing, soft aborts, dynamic process creation, and robustness as network connections die. These features can be combined with scheduling and resource management strategies in order to promote application performance. Our next focus for this work will be on application performance tuning. We will evaluate adaptive scheduling techniques developed in previous work [2, 24, 28, 29], demonstrate how TOP-C provides a good framework for implementing those techniques, and perform extensive sets of experiments with a number of applications on various Grid testbeds.

Acknowledgements

The authors wish to thank the Innovative Computing Laboratory at the University of Tennessee, Knoxville for providing computing resources for running experiments.

References

[1] Ampic webpage. `http://grail.sdsc.edu/projects/ampic/`.

[2] AMWAT webpage. `http://grail.sdsc.edu/projects/amwat`.

[3] Global Grid Forum. http://www.gridforum.org.

[4] MPI Forum webpage. `http://www.mpi-forum.org`.

[5] MPICH webpage. `http://www.mpi-forum.org`.

[6] J. Allison, J. Apostolakis, G. Cosmo, P. Nieminen, M. Pia, and the Geant4 Collaboration. Geant4 Status and Results. In *Proc. of CHEP 2000*, pages 81–84, Padua, 2000.

[7] G. Alverson, L. Anchordoqui, G. Cooperman, V. Grinberg, T. McCauley, S. Reucroft, and J. Swain. Using TOP-C for Commodity Parallel Computing in Cosmic Ray Physics Simulations. *Nuclear Physics B (Proc. Suppl.)*, 97:193–195, 2001.

[8] G. Cooperman. STAR/MPI: Binding a Parallel Library to Interactive Symbolic Algebra Systems. In *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*, volume 249 of *Lecture Notes in Control and Information Sciences*, pages 126–132. ACM Press, 1995. `http://www.ccs.neu.edu/home/gene/software.html#starmpi`.

[9] G. Cooperman. TOP-C: A Task-Oriented Parallel C interface. In *5th International Symposium on High Performance Distributed Computing (HPDC-5)*, pages 141–150. IEEE Press, 1996.

[10] G. Cooperman. GAP/MPI: Facilitating Parallelism. In *Proc. of DIMACS Workshop on Groups and Computation II*, volume 28 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 69–84. AMS, 1997.

[11] G. Cooperman. Practical Task-Oriented Parallelism for Gaussian Elimination in Distributed Memory. *Linear Algebra and its Applications*, 275–276:107–120, 1998.

[12] G. Cooperman. TOP-C: Task-Oriented Parallel C for distributed and shared memory. In *Workshop on Wide Area Networks and High Performance Computing*, volume 249 of *Lecture Notes in Control and Information Sciences*, pages 109–118. Springer Verlag, 1999. `http://www.ccs.neu.edu/home/gene/topc.html`.

[13] G. Cooperman. TOP-C: Task Oriented Parallel C/C++. `http://www.ccs.neu.edu/home/gene/topc.html`, 2001. includes 40-page manual.

[14] G. Cooperman and V. Grinberg. Scalable Parallel Coset Enumeration Using Bulk Definition. In *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '01)*, pages 77–84. ACM Press, 2001.

[15] G. Cooperman and M. Tselman. New Sequential and Parallel Algorithms for Generating High Dimension Hecke Algebras using the Condensation Technique. In *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '96)*, pages 155–160. ACM Press, 1996.

[16] C. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing*, 2001. to appear.

[17] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[18] H. Dail. A modular framework for adaptive scheduling in grid application development environments. Master's thesis, University of California at San Diego, March 2002. Available as UCSD Tech. Report CS2002-0698.

[19] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[20] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[21] I. Foster, C. Kesselman, and S. Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[22] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 2001. to appear.

[23] I. Foster and K. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[24] G. Shao. *Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources*. PhD thesis, University of California, San Diego, May 2001.

[25] Geant4 Team. Geant4. `http://wwwinfo.cern.ch/asd/geant4/geant4.html`, 2001.

[26] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM : Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press Cambridge, Massachusetts, 1994.

[27] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Wide-Area Computing: Resource Sharing on a Large Scale. *IEEE Computer*, 32(5):29–37, May 1999.

[28] E. Heymann, M. Senar, E. Luque, and M. Livny. Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In *Proceedings of the First IEEE/ACM Internation Workhop on Grid Computing (GRID 2000), Bangalore, India*, December 2000.

[29] J. Linderoth, S. Kulkarni, J.-P. Goux, and M. Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania*, pages 43–50, August 2000.

[30] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerial Libraries and the Grid: The GrADS Experiment with ScaLAPACK. In *Proceedings of SC'01, Denver*, November 2001.

[31] G. Shao and R. Berman, F.and Wolski. Master/Slave Computing on the Grid. In *9th Heterogeneous Computing Workshop*, May 2000.

[32] R. van Nieuwpoort, J. Maassen, H. Bal, T. Kielmann, and R. Veldema. Wide-area Parallel Programming Using the Remote Method Invocation Model. *Concurrency Practice & Experience*, 12(8):643–666, July 2000.

[33] M. Weller. Construction of Large Permutation Representations for Matrix Groups II. *Applicable Algebra in Engineering, Communication and Computing*, 11:463–488, 2001.