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

Gene Cooperman [a,1] Henri Casanova [b,c] Jim Hayes [b]
Thomas Witzel [a]

[a] *College of Computer Science, Northeastern University*

[b] *Computer Science and Engineering Department, University of California, San Diego*

[c] *San Diego Supercomputer Center, University of California, San Diego*

Abstract

Porting large parallel applications to new and various distributed computing platforms is a challenging task from a software engineering perspective. The primary aim of this paper is to demonstrate how the development time to port very large applications to the *Computational Grid* can be significantly reduced. TOP-C and AMPIC are software packages that have each seen successful applications in their respective domains of parallel computing and process creation/communication over the Computational Grid. We combined the two packages in one man-week, thereby leveraging several man-years of previous independent software development. As a real world test case, the 1,000,000 line Geant4 *sequential* application was then deployed over the Computational Grid in three man-weeks by using TOP-C/AMPIC. The cluster parallelization of Geant4 using TOP-C is now included as part of the Geant4 4.1 distribution, and the integration of TOP-C/Ampic and the Globus protocols will additionally enable the use of the fundamental Grid middleware services in the future.

Key words: computational grid, master-worker parallelism, TOP-C, Ampic, Geant4

Email addresses: `gene@ccs.neu.edu` (Gene Cooperman), `casanova@cs.ucsd.edu` (Henri Casanova), `jhayes@cs.ucsd.edu` (Jim Hayes), `twitzel@ccs.neu.edu` (Thomas Witzel).

[1] This work is supported by the National Science Foundation under Grant No. ACI-9872114

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) [27], 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 [28].

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 (e.g. see [6,25,31]) and has been the object of a number of research endeavors (see [29] for a comprehensive survey).

The TOP-C project [11] 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 [7,8,11]. The power of the TOP-C abstractions was demonstrated in [3,10,12,13,32].

Among today's available distributed computing platforms, the *Computational Grid* [16,19] has gained tremendous popularity. It aggregates large amounts of compute and storage resources over a 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 [20]. Those services are being generalized in the context of the Global Grid Forum [23].

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 Computational Grid infrastructure, it is necessary to re-engineer its implementation so that it can use Grid services. Rather that implementing TOP-C directly on top of Globus services, we advocate the use of an intermediate software layer: AMPIC.

AMPIC [4] 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 [27] or PVM [22], 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. 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 Computational 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 our approach in the real world, we take as a case study the Geant4 particle physics application [1,21]. 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 work targeting master-worker applications. Among recent such efforts in the context of the Computational Grid are the AMWAT project [5], and the MW project realized as part of Condor [24,26]. 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 (see [11]). 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 [24,26] 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 (see Section 4) with only minimal software engineering efforts. From that standpoint, aspects of this work would be eminently applicable to the MW project. We plan to investigate possible integrations and collaborations with the Condor MW project.

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 [30]. 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 [7], to C and C++ [8] and to GAP [9]. It provides a uniform interface which can execute both in a distributed and shared memory model [11], and now over the Grid. 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 star topology (master/worker architecture). A TOP-C application is SPMD (Single Program, Multiple Data). Hence, the same application is run on master and worker processes. Each process operates independently until a call to `TOPC_master_slave()`. Entry into `TOPC_master_slave()` 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 programmer's 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:

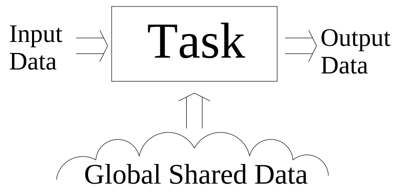

Fig. 1. TOP-C Concept #1: The task abstraction

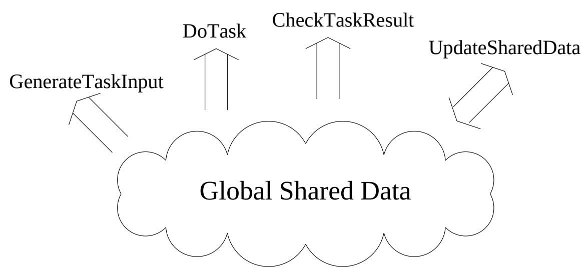

Fig. 2. TOP-C Concept #2: The shared data abstraction

(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 our application, `CheckTaskResult()` always returns `NO_ACTION`. In general, non-trivial parallelism is implemented through other possible actions, including `UPDATE` (invoke `UpdateSharedData()` on all processes) and `REDO` (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. [30]

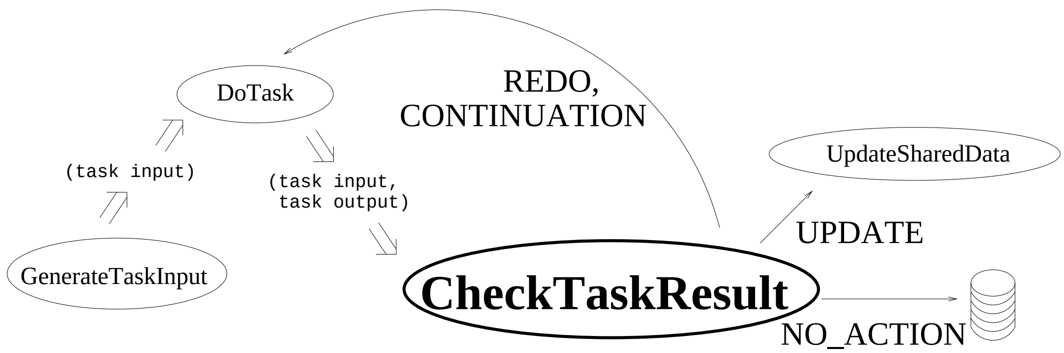

Fig. 3. TOP-C Concept #3: The action abstraction

3 The AMPIC middleware

The AppLeS Multi-Protocol Interprocess Communication (AMPIC) [4] 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 [5,29] which is related to TOP-C in the following way. Like TOP-C, AMWAT targets master-worker applications. However, AMWAT's focus in on scheduling whereas TOP-C's focus is on providing a flexible and powerful programming model (see Section 2). 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, the 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 [4]. AMPIC provides fundamental functions are 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 technology 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 [27], PVM [22], Globus [20], 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 and can use shared memory, sockets, and Secure Shell on that platform.

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 3.2 (a large simulation library with approximately 1,000,000 lines of C++ code), AMPIC version 1.2 (middle layer between the Globus toolkit and the target application), and the Globus toolkit. The TOP-C and AMPIC packages are written in C, while Geant4 is written in C++ using strict software engineering principles.

This work combined three large software projects, which 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. 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. The integration of the Geant4 and TOP-C layers, and the integration of the TOP-C and AMPIC layers, were carried out part-time over a period of three months.

4.1 Integration of Geant4 with TOP-C

Geant4 (GEometry ANd Tracking) [1,2,21] is a simulation package for particle showers developed at CERN. It was initially planned primarily to simulate the particle showers of a collider, although it has since seen many other uses. 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 col-

Geant4

TOP-C

AMPIC

Globus Toolkit

Fig. 4. Integration of Software Layers

lider experiments in Europe, Russia, Japan, Canada and the United States. Many specialized working groups are responsible for the various domains of the toolkit. The history of GEANT goes back to 1970 and can be followed through CERN.

An early parallelization of Geant4 using TOP-C was reported on in [3]. That work was carried out with the single goal of simulating cosmic ray showers over Geant4. It was carried out without the opportunity for face-to-face discussions with the Geant4 developers. 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 unmodified the Geant4 kernel library. The total development time was approximately three weeks. This code for integration of TOP-C using this second parallelization was then incorporated into the Geant4 distribution in June, 2002, and is included in Geant4 version 4.1.

The changes were created by writing a distinct library that depended on the Geant4 library and from which a new, larger Geant4 library was created. 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%. This is an important issue for TOP-C, since it operates in a star topology. 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, a student. Hence, the software development was carried out on weekends and some evenings. The elapsed time from the beginning to the end of this phase was two weeks.

As often happens in these situations, the integration was also a useful exercise to uncover a bug in AMPIC. The bug involved AMPIC's use of the Globus GRAM [15] to spawn processes on remote resources. It had not previously appeared because the TOP-C integration stressed AMPIC in a way which would not typically be seen in ordinary applications.

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 [18] and GRAM [15]. 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. But it also allows TOP-C to run using PVM [22], using Secure Shell to start remote processes, using shared memory transparently for inter-process communication, or any simultaneous combination of AMPIC's supported mechanisms.

The combined software (Geant4/TOP-C/AMPIC) was tested with the Globus toolkit using a Globus testbed (see 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 on the two grids was given by different absolute path names. 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 for Validation

The objective of this paper is to show that 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. We report on two experiments in order to further validate our approach. We used a two site Computational 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 but was unloaded, 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 PIII running Linux and Globus version 1.1.3. All results presented hereafter are averages over 10 application runs.

5.1 Geant4 runs

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. Experiments were run for a total of 16 Geant4 work units. Thus, there was a static, even partitioning of the Geant4 work units was our choice for work allocation. Note that the testbed is homogeneous, meaning that all compute resources have equivalent computing speeds. Using a heterogeneous testbed

# 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
8	87.6	66.6	21.0
16	95.5	33.9	61.6

Table 1
Sample Geant4 runs on the Grid testbed

would require careful work allocation and load-balancing, which is not in the scope of this paper. However, note that it would be simple to integrate any standard load-balancing algorithm in our integrated software, due to the high-level of abstractions provided by TOP-C.

The results of the Geant4 runs are shown in Table 1. 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 the numbers in Table 1 were below 2 seconds.

One observation is that the busy time decreases nearly linearly as the number of processors increases. Such linear speed-up is due to the fact that in this Geant4 run communication was not a bottleneck for the application. A more interesting observation is that the spawn time dramatically increases, becoming close to a minute when using all 16 hosts. This cost negates the benefit of using 16 hosts for this Geant4 run as it leads to an performance decrease of 9% over using 8 hosts. In traditional parallel applications such degradation in performance is generally expected given the communication to computation ratios. Note that the behavior in this case is *solely* due to the overhead incurred when accessing and acquiring Grid resources.

The overhead is due to accessing resources via the Globus Security Infrastructure (GSI) [17] and the GRAM [15] over the network. We do not access Grid information services (such as the MDS [14]), but instead cache that information, since it rarely changes. Querying Grid information services would further increase the observed overhead. These results indicate that the Grid overhead should be accounted for when evaluating possible schedules for "short-lived" Grid applications.

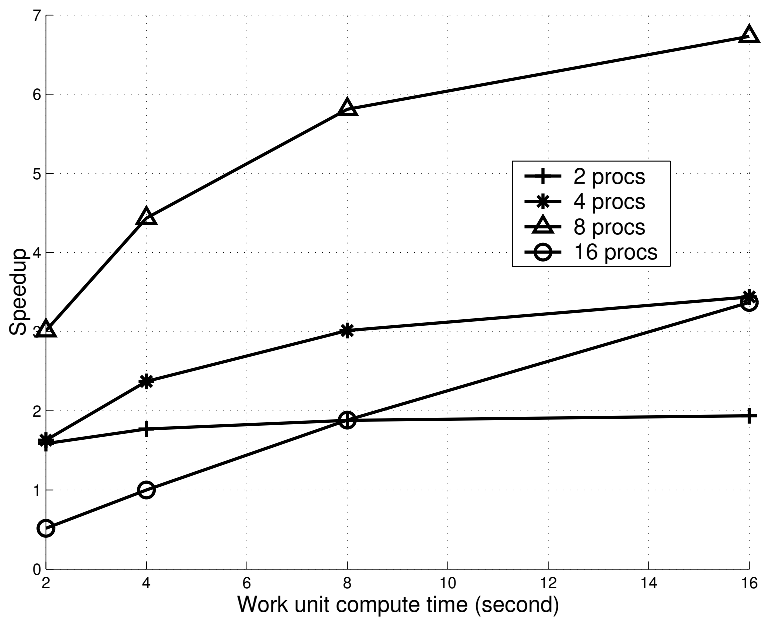

Fig. 5. Speedup of the master-worker program for the 2, 4, 8, and 16 hosts testbeds versus computation time for a work unit.

5.2 Speedup for Varying Computation-Communication Ratios

Since the Geant4 application had relatively light communication requirements, we added a second experiment concerned with varying rations of communication to computation. We implemented a simple and customizable "toy master-worker program" using TOP-C and Ampic. This allowed us to vary message sizes and work unit computation times for experimental purposes (e.g. to simulate applications other than Geant4). We successfully ran this program on the same Grid testbed using a wide range of values for number of tasks, message sizes, and computing costs.

We present a set of results in Figure 5. The figure plots the speedup achieved by the master-worker program versus the computation time for one task or work unit. We show a curve for the same testbed configurations as described in Section 5.1. We employ a message size of 1 MB, in order to createn significant communication overhead. We see that when the testbed spans only one institution, i.e. for 2, 4, and 8 processors, the speedup increases logarithmically with work unit sizes. When the computation time of the work unit reaches 16 seconds, each test has nearly reached its maximum speedup.

For the 16 host testbed, which is spanning resources at UCSD and UTK, we see that the speedup curve grows almost linearly. This is due to the penalty of the large initial overhead incurred to start tasks (as also experienced in our Geant4 runs), and also to the higher network overhead. On the 16 processor testbed, work unit compute times must be much higher so achieve speedup values close to 16. In fact, there is a slowdown for smaller work units.

From the perspective of of parallel computing, the results of for Sections 5.1 and 5.2 are to be expected, given the nature of the testbed and of the application. Nevertheless, this demonstration in varied configurations is important as a validation that our software integration correctly implements a simple, yet powerful, programming model.

6 Conclusion and Future Work

In this paper we have reported on our software integration efforts of the TOP-C and AMPIC packages for parallelizing and enabling the Geant4 application to run on a variety of Computational Grid resources, and in particular 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. Also, 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 creation, and robustness as network connections die. These features can be combined with scheduling and resource management strategies in order to promote application performance. 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 Grid. Our next focus for this work will be on application performance. We will evaluate adaptive scheduling techniques developed in previous work [5,24,26,29], 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] A. Dell' Acqua et al. Geant4: a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A*, 2003. to appear, also available as tech. report: CERN-IT-20020003, KEK Preprint 2002-85, or SLAC-PUB-9350.

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

[3] 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.

[4] Ampic webpage. `http://gridlab.ucsd.edu/ampic/`.

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

[6] J. Basney, R. Raman, and M. Livny. High throughput Monte Carlo. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing (San Antonio, Texas)*, 1999.

[7] 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`.

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

[9] 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.

[10] G. Cooperman. Practical task-oriented parallelism for Gaussian elimination in distributed memory. *Linear Algebra and its Applications*, 275–276:107–120, 1998.

[11] 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`.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

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

[21] Geant4 webpage. `http://cern.ch/geant4/`.

[22] 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.

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

[24] 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.

[25] V. Kumar, K. Ramesh, and V. Rao. Parallel best-first search of state-space graphs: A summary of results. In *Proceedings of the 1998 National Conference on Artificial Intelligence*, pages 122–127, March 1998.

[26] 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.

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

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

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

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

[31] D. Weber, M. Specialetti, and H. Barada. Vidnet: Distributed processing environment for computer generated animation. *Software - Practice and Experience*, 26(2):237–250, 1996.

[32] M. Weller. Construction of large permutation representations for matrix groups II. *Applicable Algebra in Engineering, Communication and Computing*, 11:463–488, 2001.