PION: A PROBLEM SOLVING ENVIRONMENT FOR PARALLEL MULTIVARIATE INTEGRATION*

SHUJUN LI †, ELISE DE DONCKER †, AND KARLIS KAUGARS †

Abstract. ParInt is a package for parallel multivariate numerical integration. This paper describes the design and implementation of a problem solving environment based on ParInt and Web technology. We call it ParInt ONline (Pion). It facilitates both common end-users and experts to solve computationally intensive numerical integration in parallel. No parallel programming experience or any knowledge of Unix/Linux operating systems is needed for the users. When the user submits an integration problem to Pion via a Web browser, the problem solving environment will compile the ntegrand function and link it dynamically with the ParInt package so that the execution can be done in parallel on high performance computing servers. The system was designed to be a globally accessible integration platform that operates as a black box, taking user data and producing the results.

Key words. problem solving environment (PSE), parallel integration, scientific visualization

AMS subject classifications. 68T20, 65D30, 65Y05

1. Introduction.

A high performance computing system consists of the hardware, the runtime systems, the programming languages, the problems, and the users. The objective is to solve the problems. Too much is involved for most users to buy the hardware, set up the systems, learn a special programming language, and write programs to solve their problems. Usually, it is unnecessary for them to even know the details of the algorithms. The solution to bridge the gap between the user and the high performance computing system is the so-called Problem Solving Environment (PSE) [19, 15, 13, 5]. A PSE provides a complete environment for solving large computationally intensive problems. Attention is paid to the needs of both end-users and experts, who are in a position to steer the solution of the application, and require suitable problem solving techniques based on information gathered during the problem solving process.

The characteristics of a PSE are:

Problem domain. PSEs offer an integrated environment for solving specialized problems, including means for composing, compiling, and running applications, while handling security issues.

Parallel resources. PSEs provide easy access to distributed computing resources, and state of the art problem solving power to the end user.

Black box configuration. PSEs attempt to remove hardware and software complexity, in order to eliminate the need for the end user to be an expert in the solution method.

Grid distribution. PSEs can be implemented on top of grid services.

Visualization. PSEs should address visualization that enhances computational steering, analysis and interpretation of the resulting data.

The aim of this work is to incorporate the state of the art integration techniques provided by ParInt into a globally accessible integration service. This effort is best understood in the context of a complete problem solving environment [19, 15]. We will call ours Pion, which stands for Parallel Integration ONline.

*This work was supported in part by NSF grants CISE ACR-0000442, EIA-0130857, ACI-0203776, and Western Michigan University.

†Department of Computer Science, Western Michigan University, Kalamazoo, MI 49008 USA ({sli,elise,kkaugars}@cs.wmich.edu).

Many PSEs are designed to solve problems in specific domains. Ecce [14] is a PSE, consisting of a suite of applications or components for constructing molecules, assigning basis sets, selecting input options, browsing resource availability, launching jobs, visualizing results, and creating and managing projects and calculations.

PDELab [13] is a PSE for modeling physical objects described by partial differential equations. It provides a graphical interface to define a problem and select a solution method, then provides powerful computational resources to solve the problem and visualize the results.

VizCraft [12] is problem-solving environment used by aircraft designers during conceptual design. It integrates simulation code that evaluates a design with visualization for analyzing the design individually or in contrast with other designs.

WBCSim [12] is a prototype PSE that is intended to aid in the research of wood-based composite materials, by allowing access to legacy FORTRAN programs.

The following systems target more general applications. The paper [9] presents a model for a World Wide Web computational server, which uses Maple as its underlying engine, and communicates with its clients using OpenMath, MathML, and VRML. The authors mainly focus on the client/server communications.

Cactus [2] is an open source PSE intended to provide a unified modular and parallel computational framework for engineers and physicists.

NETCARE [6], powered by PUNCH [7, 6] and Condor [16], is a Web-accessible distributed infrastructure of tools and computing resources for the computer architecture community. PUNCH allows users to access and run command-line tools via standard World Wide Web browsers. Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management for batch jobs. With NETCARE, students, educators, and researchers can share, test, evaluate, and use these tools on problems in electronic design automation.

However, it is impractical to use any of the current general purpose Web-enabling tools for our application. The idea of Web-based program execution is simple. Via the graphical user interface on the Web page, input parameters for an application are passed to the middleware which interacts with the Web server. The middleware starts the application and sends the results back to the user via the Web server. However, the PSE of ParInt must be designed and implemented specifically to handle the following situations.

The ParInt end user enters an integrand function which will be compiled into a library. The library is loaded dynamically by ParInt. This is a dedicated procedure that is not handled via general purpose Web-enabling mechanisms.

Secondly, some ParInt parameters are interdependent. For example, an n-dimensional problem needs $2n$ numbers for its hyper-rectangular lower and upper limits. When an argument indicates that the integration region is a simplex, $n(n+1)$ numbers are required to define the simplex region. In our approach, the JavaScript checks parameters and handles inputs accordingly.

Finally, ParInt uses a heap (optionally, a double ended heap or deap) to keep intermediate regions, which may consume a large amount of memory. Therefore, a customized resource management strategy is preferred.

To design a PSE for numerical integration, we have to keep the following factors in mind. The execution time of a job is often unpredictable, which means that both the system administrator and the user should be able to control the execution of the job. The resources may have to be restricted by the system administrator. For example, the system administrator should be able to limit the number of processes running on a node.

Based on current technologies, the performance, user interface, and accessibility lead to trade-offs. It may be easy to meet one or two of these, but building a system that meets all of them requires extra effort. For example, one can write a command-line parallel program for users who have access to a parallel system. One can even provide a decent graphical interface. But if the system is powerful and it needs to be made accessible to remote users, there are still a lot of considerations. On the other hand, if the performance issue can be neglected, so can the term PSE, because many available desktop applications are somewhat user-friendly. Most PSEs address the issue of performance. Some have a Unix/Linux desktop interface. Some are implemented in Java so that users can access the PSE using a Web browser. We will discuss the pros and cons of using Java Applets as the major GUI below.

We aim at building a problem solving environment for numerical integration, so that the users can focus on the problems themselves, instead of focusing on coding, computational techniques or the algorithms used to solve the problems. Unless the integrand functions are predefined in the program, solving an integration problem numerically involves entering the integrand function in some way and compiling it, so that it will work with other parts of the program.

Commercial products such as Maple, Mathematica and MatLab support symbolic and numerical integration. However, the users have to pay for the products and learn to use them. They may not be efficient enough to solve complicated problems. In particular, numerical integration is not well-supported. Non-commercial functions, packages or toolkits are also available. A common problem of these programs is that they are not user-friendly. If we want to provide a user-friendly solution over the Web, there are a number of possible ways to proceed.

Web browsers are becoming popular as interfaces to remote high performance computers. Some PSEs provide this kind of interface. Based on the current technology, there are three major graphical user interface technologies for accessing remote computing resources, namely Java Applets, Java Web Start, and through a Web browser without using Java.

Some PSEs use Java Applets in their graphical user interface to display images or data. We also have an Applet version of the interface to ParInt [10]. In [10], multi-thread Java server communicates with the Java Applet and controls the job execution. We found that it is inconvenient to manage the system and user data with our server. A Java Applet can be a good visualization component of the environment. However, if the Web-based PSE is designed to be globally accessible, it is much easier to use a middleware layer coded in PHP (a widely-used, Open Source, general-purpose scripting language that is especially suited for Web development) or JSP (JavaServer Pages), with database support to handle most interactions and data.

Java Web Start is an innovative technology for deploying applications based on the Java 2 platform, which enables the user to launch full-featured applications via a browser. It provides the ease of deployment and the use of HTML, as well as the power and flexibility of a full-fledged application. It frees the developer from concerns as to how the client is launched. It may be a good candidate for result visualization. However, if we use it to support the client deployment, we still have to implement our own server that takes care of everything, including connections.

Furthermore, it is increasingly difficult to implement a PSE when the application logic becomes complex. The users need to install Java plug-ins, and it is difficult for some users to install plug-ins successfully on some platforms. With these considerations in mind, we used a simple and extensible method based on JavaScript and DOM (Document Object Model) [18] to build the GUI.

2. ParInt Overview. ParInt is a software package that numerically solves integration problems in parallel. Multivariate functions are integrated over hyper-rectangular or simplex regions, using cubature, Monte-Carlo (MC) or Quasi-Monte Carlo (QMC) rules. General product regions can be handled using transformations [17]. For a function $f(\mathbf{x})$, the objective is to calculate an approximation Q to the integral

$$I = \int_{\mathcal{D}} f(\mathbf{x})\ \mathrm{d}\mathbf{x},$$

over a given n-dimensional domain $\mathcal{D}$, and an error estimate E_a, subject to

$$|I - Q| \le E_a \le \max\{\varepsilon_a, \varepsilon_r |Q|\},$$

where ε_a, ε_r are user-specified absolute and relative error tolerances, respectively. The integrand may be a vector function $\mathbf{f}(\mathbf{x})$, as long as the component functions are sufficiently similar (to allow their integral approximation using the same strategy).

ParInt will evaluate the integrand function a number of times to get the result. The number of function evaluations is used as a measure of the amount of effort spent in the computation. The user can set an upper limit on the number of integrand evaluations to ensure that the computation will stop. It is quite possible that the result of an integration may not be achieved within the given error tolerances, due to the nature of the integrand function, the accuracy requirement, the computation time, the effect of possible round-off error in the computation, or the limits on machine precision. If the function evaluation limit is reached, then the required accuracy has probably not been achieved. Various algorithm parameters can also be specified by the user, for optimization and speed-up of the computation.

3. The Architecture. The ParInt code can be invoked through function calls in an application. It can also be compiled into a ParInt console executable. The user can perform integration via a GUI, which runs locally or remotely. The overall architecture is shown in Figure 3.1.

ParVis [8, 4] is an interactive graphical tool for analyzing region subdivision trees and mesh structures for adaptive partitioning algorithms. It currently provides post-mortem event viewing. In future work, it will be integrated into the Web-based interface environment to enable computational steering.

This paper focuses mainly on the left half of Figure 3.1, with the remote interface to the parallel computing resources. The figure depicts a typical three-tier architecture with database support. However, this environment differs from the common Web applications in that it has to deal with the special issues arising from high performance computing.

The remote interface is a Web browser. JavaScript and DOM [18] are intended for the user's specification of the problem and its parameters, and the display of the results and error estimates. The middle layer provides user management, integrand function management, job management, and result management. In the bottom layer, the ParInt Plug-in Compiler (PPC) [20] (which uses gcc) compiles the code of an integrand function to an object file, which is then loaded dynamically by ParInt.

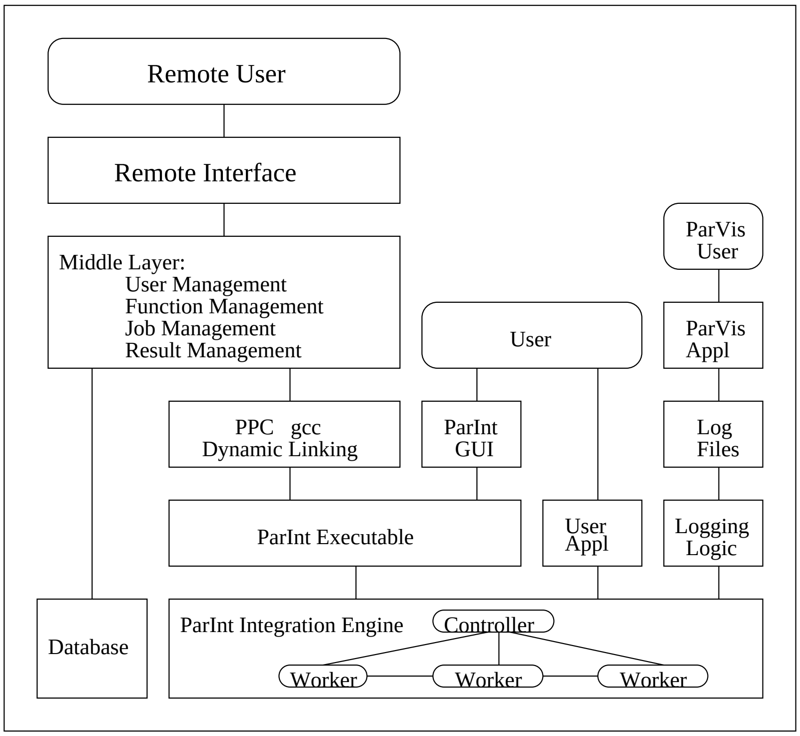

FIG. 3.1. *The Pion architecture.*

The back end contains the parallel PARINT integration engine and a database for user data.

4. User Management. Many Web applications need to keep user data for various reasons. The simplest way to handle this is by making a directory for each user. Creating individual directories has the advantages of less access time and easy deployment. This method is suited in small applications. MetaΨ [3] uses LDAP (Lightweight Directory Access Protocol) and a Web server as the middleware to locate computing resources and to keep user profiles. For Pion, we found that using a database is a better choice. In addition to the normal fields such as the user name and the email address, the user table currently has two additional fields, maxRunTime and maxJobs. These two values determine the user's computing resource access levels. Depending on the workload trends of the system, more fields may be added to classify the users. In addition to managing user data, a Web-based computational system must have a way to control the computing resources. This issue will be discussed in the job control section below.

5. Function Management. There are two stages in setting up a numerical integration problem, namely defining the problem and specifying the numerical algorithm's parameters. On the "New Function" page of the Pion interface (cf. Figure 6.1), the user enters the dimension of the integrand function, the absolute and relative error tolerances, the type of integration region, the region itself and the integrand function. The user has the option to upload the function definition from the local disk. When the function is submitted to the server, all the data is saved to a function table in the database. The function is then compiled to generate a .ppl file, which is linked dynamically with the ParInt executable at runtime. If there are compilation errors, another page will show them with highlighted line numbers. The user must then correct the errors and submit the function source code again.

For a multivariate integral of dimension n, $2n$ and $n(n+1)$ text boxes are required for specifying hyper-rectangular and simplex integration regions, respectively. At times, the user may want to change the dimensions or switch to another region type. To solve this problem, we use JavaScript to manipulate the DOM [18] object of the Web page. When the change of dimensions or region type is confirmed by another event, the JavaScript code will compute the new numbers of rows and columns, and refresh the text box table.

DOM [18] is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. It can be accessed via any language available in the browser, including Java, JavaScript, and VBScript (for Internet Explorer only). DOM is supported in Internet Explorer 5 and Netscape 6. The dynamic characteristics of DOM are useful to handle interdependent parameters. To support the earlier browsers we also have pages that do not use DOM. In this case, the user must specify the number of dimensions and the integration region type on one page, and enter the other parameters on another page.

All integrals owned by a user are listed in a table. The user can select an entry to edit, clone or delete. The table also indicates which rules/techniques are available to solve the integration problem at hand. For example, Pion provides adaptive, Quasi-Monte Carlo, and Monte Carlo algorithms for hyper-rectangular integration regions. An adaptive cubature is implemented for simplex regions.

For low dimensional problems, the adaptive algorithm is preferred, unless the integrand behavior is erratic. For erratic integrands and/or high dimensions, Monte Carlo integration is advised. Quasi-Monte Carlo integration is justified for some high dimensional problems, under certain smoothness conditions on the integrands.

The user also specifies a function evaluation limit, a timeout limit, and a number of processes for the parallel execution. Advanced algorithm parameters are intended for users who have some idea of the algorithm implementation. These parameters can be specified by clicking the “Advanced Options” button.

6. PPC Compiler. Before an integrand function written in C is compiled, it is combined with the other integration parameters to generate an intermediate file. The intermediate file is compiled by a pre-processor called PPC (ParInt Plug-in Compiler) to generate a temporary C source file [20]. This temporary file is then passed to the C compiler to be compiled into a .ppl file. The C compiler reports errors in terms of the intermediate file, not the function entered by the user in the “Function Body” box. The middleware of Pion adjusts the error line numbers to reflect problems with the original file and highlights them for the end user as shown in Figure 6.1. In the event of no errors, the user will be informed with a short message of a successful compilation.

It is clear that allowing the user to write a function and run it on the system is potentially a security hole. Therefore some system calls are not allowed in the .ppl file. They are: accept(), bind(), fopen(), getmsg(), msgget(), open(), pause(), poll(), putmsg(), select(), semop(), wait(), alarm(), brk(), chdir(), dlclose(), dlerror(), dlopen(), dlsym(), exec(), fork(), popen(), pthread_create(), sbrk(), sethostent(), setgid(), setuid(), signal(), system(), thr_create(), and umask(). The restriction with

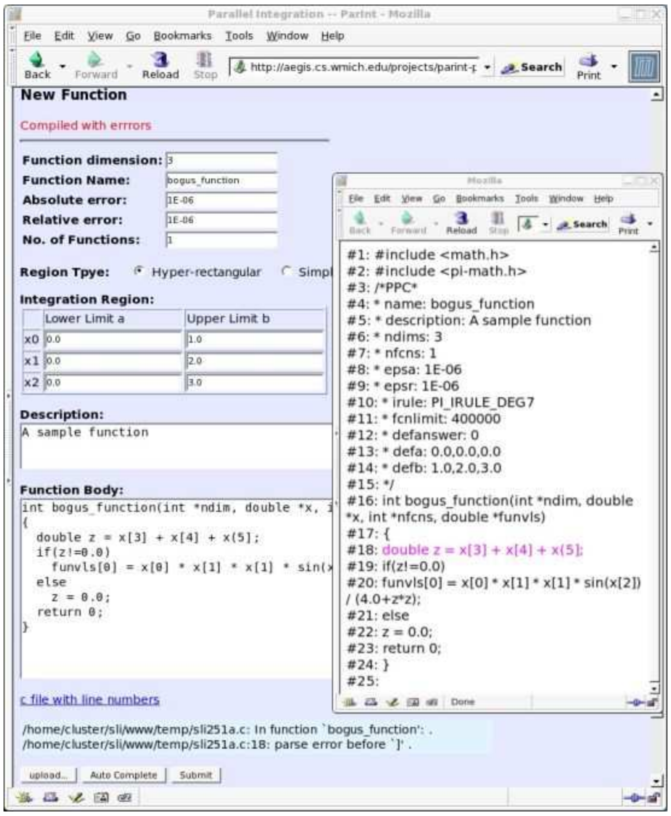

FIG. 6.1. *Integrand function compilation. Error messages are shown when a submitted integrand function has errors.*

respect to certain function calls is important in an environment where the user's real identity may be unknown.

7. Job Control. Load balancing is always an issue in distributed computing. When a user submits a job to Pion, the middleware will call the MPI [11] mpirun command which typically uses the first p (a user specified number) processors defined in a machine file to execute the job in parallel. In a multi-user environment, it likely that the first several processors are overloaded, while the last ones are idle. Pion has a module to check the workloads of all machines and select suitable processors for the mpirun command. The "System Info" pages show the status of the system in detail for both users and administrators.

A globally accessible computing system should have a mechanism to optimize system utilization. Unlike an online shopping application where the transaction time range is well-known, the time for a computational job is in general unpredictable. It can be a millisecond or a month, depending on the size of the problem. A user might submit a job that runs forever and never returns any result. As mentioned before, there is a field in the user table to specify a maximum execution time (maxRunTime) for the user's jobs. The value can be increased by the system administrator upon request. Another value is the maximum number of jobs (maxJobs) that a user can start concurrently in the system. This value is 1 by default, but can also be increased upon request. Our policy is to allocate just enough resources for a user. The user interface is designed to encourage the users to manage their jobs interactively. The same logic is applied to the input box for the number of processors. Its default value is always one, while most other boxes display the last values entered.

Killing a job is not an easy task in a Web-based computing system. Questions must be answered such as: who can kill the job (the user, the system, or both); how the job is killed; when the job should be killed; how the user is informed if the system has killed a job.

A user can refer to a status table on the Web page, which shows the numbers of running jobs, done jobs, and stopped jobs (killed by the user or the system). Let us first cover how the user executes a job, then discuss the implementation in Pion. After entering the integrand function and its problem parameters, the user: 1) specifies execution parameters including the number of processors and a timeout limit (should be less than maxRunTime); 2) gets the result back if the execution time is less than about one second, or 3) waits for the result while the screen is being refreshed; 4) increases the timeout limit (but not exceeding maxRunTime) if necessary; 5) aborts the job execution, or 6) closes the browser and comes back later to check the job status; 7) saves or discards the result. Figure 7.1 shows the page for the user to manipulate a job.

The database has a transaction table with fields including: startTime, timeout, endTime, and isChecked. When the user submits a job, the timeout value and a timestamp are stored to the timeout field and startTime field, respectively. The system checks the transaction table periodically, and kills all jobs that are timed out. After a job has stopped running, a timestamp is written to the endTime field. After the user has viewed the result, the isChecked field is changed to true, indicating that this transactional record will not be manipulated again unless for statistics purposes. The data in the transaction table can be used to generate a histogram indicating the activities of the system. For example we can show the workload of the day or of the past week. The implementation of job management depends greatly on the scripting language of the middleware.

8. Result Data Management. Result management is simple but critical to problem solving environments on high performance computer systems. Simple Web applications rarely save the results for the user, because it is easy to repeat the computations. Pion, however, keeps the results even though the user closes the browser and ends the session. After the output has been generated, it is saved to a file. Although it is trivial to support e-mail notification, we decided not to implement this feature, because the user will come back and check the run status regularly if the output or the execution time is important.

Unlike other PSEs, Pion does not allow the users to access the file systems of the server directly, nor does it provide a virtual hierarchical file system. From the user's point of view, file manipulation is unnecessary. The results of an integration problem are associated with the problem and can be accessed via the links in a table.

9. Performance Evaluation. Pion was designed and implemented to be a globally accessible Web-based problem solving environment for high performance computation. Much attention has been paid to the security and load balancing of the system. It can be accessed at the following URL, http://aegis.cs.wmich.edu/projects/parint-php/. Currently our Beowulf Linux cluster consists of 64 nodes, including 32 AMD Thunderbird nodes at 1.2 GHz and 32 AMD Athlon nodes at 800MHz. They are connected by both Fast Ethernet and Myrinet. For users who want to run PARINT

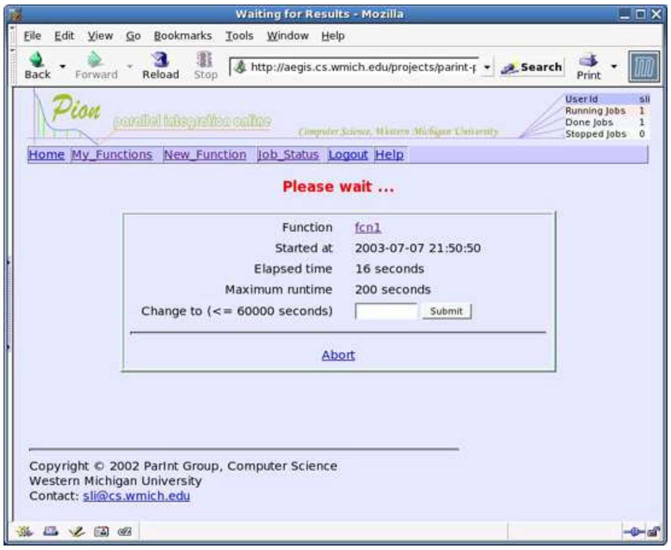

FIG. 7.1. *This page will display the results; it also gives the option to increase the execution time limit, or to abort the execution.*

on their own hardware resources, the PARINT package is available for download [1]. It can be compiled to parallel executables for a cluster, or sequential executables for a standalone Unix/Linux machine.

The administrative overhead of Pion is insignificant. The compilation time of an integrand function and the functions it calls depends on the size of the source code. The overhead of the Pion middleware for a job execution is about 0.1 second plus the extra waiting time described below. When a job is submitted, the middleware will wait 0.3 seconds for the result. If the result is produced within 0.3 seconds, it is returned to the user. If the execution time is longer, a window as shown in Figure 7.1 confirms that the job is being executed. The page is refreshed to pull the result in an interval of two seconds at first. The interval is incremented by 2 seconds for subsequent refreshes until it reaches 10 seconds.

Myrinet is the default communication device. The system administrator can switch to Ethernet if needed. If there are not enough working Myrinet nodes, Ethernet is selected automatically for a job that requires a larger number of processors.

10. Conclusions and Future Work. Pion provides a state-of-the-art globally accessible problem solving environment, so that users with little computer background can also access high performance computing resources to solve their numerical integration problems.

In PUNCH [6], different applications are supported by using templates to generate HTML code for the user interface. There are actually six different executables on the Pion servers to support different integration algorithms. They handle adaptive, QMC and MC methods using double or long double precision. PHP classes are used to encapsulate the parameters for each method. Similar classes can be implemented to support applications other than PARINT.

The visualization tool PARVIS [8] is used on Linux platforms to analyze the PARINT results. Future work includes integrating a visualization tool with the browser so that the end-users can analyze their results online, and to allow for computational steering. It is further necessary to have ParInt available on multiple clusters. XML technology can be used to exchange integration data between clusters. XML format input and output is an extension currently in progress.

Acknowledgments. The authors thank the members of the ParInt project, including Laurentiu Cucos and Paul Castone, for helpful discussions.

REFERENCES

[1] http://www.cs.wmich.edu/parint, ParInt web site.
[2] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf, *The Cactus Code: A problem solving environment for the grid*, in Proc. High Performance Distributed Computing 2000, 2000, pp. 253–260.
[3] R. Baraglia, R. Ferrini, and D. Laforenza, *MetaΨ: A web-based metacomputing environment to build a computational chemistry problem solving environment*, in 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002.
[4] E. de Doncker, R. Zanny, and K. Kaugars, *Integrand and performance analysis with* ParInt *and* ParVis. To be published in Concurrency & Computation: Practice & Experience.
[5] E. Gallopoulos, E. Houstis, and J. R. Rice, *Computer as thinker/doer: Problem-solving environments for computational science*, IEEE Computational Science & Engineering, 1 (1994), pp. 11–23.
[6] N. H. Kapadia, R. J. Figueiredo, and J. A. B. Fortes, *Punch — Web portal for running tools*, IEEE Micro, 20 (2000), pp. 38–47.
[7] N. H. Kapadia and J. A. B. Fortes, *PUNCH: An architecture for Web-enabled wide-area network-computing*, Cluster Computing, 2 (1999), pp. 153–164.
[8] K. Kaugars, E. de Doncker, and R. Zanny, ParVis*: Visualizing distributed dynamic partitioning algorithms*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00), 2000, pp. 1215–1221.
[9] H. Le and C. Howlett, *Client-server communication standards for mathematical computation*, in International Symposium on Symbolic and Algebraic Computation, 1999, pp. 299–306.
[10] J. Li and E. de Doncker, *Dynamic visualization of computations on the internet*, in Lecture Notes in Computer Science, vol. 1593, Springer-Verlag, 1999, pp. 360–369.
[11] Message Passing Interface Forum. http://www.mpi-forum.org/.
[12] N. Ramakrishnan, L. T. Watson, D. G. Kafura, C. J. Ribbens, and C. A. Shaffer, *Programming environments for multidisciplinary grid communities*, 2001.
[13] J. R. Rice and R. F. Boisvert, *From scientific software libraries to problem-solving environments*, IEEE Computational Science & Engineering, 3 (1996), pp. 44–53.
[14] K. Schuchardt, J. Myers, and E. Stephan, *Open data management solutions for problem solving environments: application of distributed authoring and versioning to the extensible computational chemistry environment*, in 10th IEEE International Symposium on High Performance Distributed Computing, 2001, pp. 228–238.
[15] M. S. Shields, O. Rana, D. W. Walker, M. Li, and D. Golby, *A Java/CORBA-based visual program composition environment for PSEs*, Concurrency - Practice and Experience, 12 (2000), pp. 687–704.
[16] D. Thain and M. Livny, *Building reliable clients and servers*, in The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman, eds., Morgan Kaufmann, 2003.
[17] J. Van Voorst, A. Raju, E. de Doncker, and K. Kaugars, *Transformation interface - ParInt*, in Parallel and Distributed Computing and Systems, 2003, pp. 702–706.
[18] W3C, *Document Object Model*. `http://www.w3.org/DOM/`.
[19] D. W. Walker, M. Li, O. F. Rana, M. S. Shields, and Y. Huang, *The software architecture of a distributed problem-solving environment*, Concurrency: Practice and Experience, 12 (2000), pp. 1455–1480.
[20] R. Zanny, E. de Doncker, K. Kaugars, and L. Cucos, ParInt*1.2 User's Manual*, 2003. `http://www.cs.wmich.edu/parint/`.