Towards an Integrated, Web-executable Parallel Programming Tool Environment*

Insung Park Nirav H. Kapadia Renato J. Figueiredo
Rudolf Eigenmann José A. B. Fortes
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907
{ipark,kapadia,figueire,eigenman,fortes}@ecn.purdue.edu

Abstract

We present a new parallel programming tool environment that is (1) accessible and executable "anytime, anywhere," through standard Web browsers and (2) integrated in that it provides tools that adhere to a common underlying methodology for parallel programming and performance tuning. The environment is based on a new network computing infrastructure, developed at Purdue University.

We evaluate our environment qualitatively by comparing our tool access method with conventional schemes of software download and installation. We also quantitatively evaluate the efficiency of interactive tool access in our environment. We do this by measuring the response times of various functions of the URSA MINOR tool and compare them with those of a Java Applet-based "anytime, anywhere" tool access method. We found that our environment offers significant advantages in terms of tool accessibility, integration, and efficiency.

1 Introduction

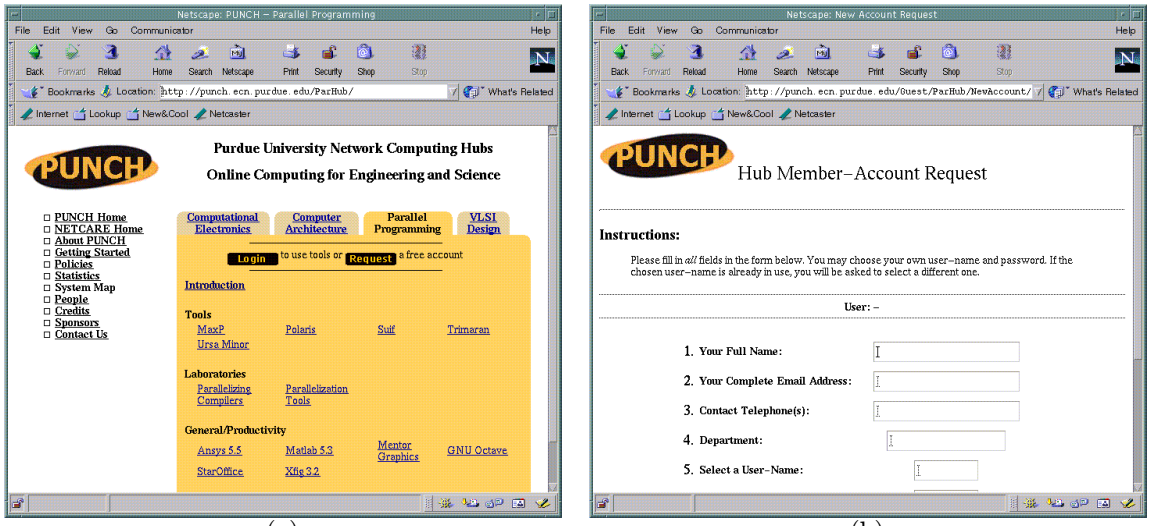
Programming tools are of paramount importance for efficient software development. However, despite several decades of tool research and development, there is a drastic contrast between the large number of existing tools and those actually used by ordinary programmers. In the present paper we provide solutions to two of the primary problems that cause this situation. The first problem is that a programmer, in order to benefit from a new tool, will typically have to go through one or several tedious efforts of searching, downloading, installing, and resolving platform incompatibilities, before new tools can even be learned and their use can be evaluated. The second problem is that, even if the value of a number of tools has been established, they often use different terminology, diverse user interfaces, and incompatible data exchange formats – hence they are not integrated.

We have created a new tool environment, referred to as the *Parallel Programming Hub* (or briefly, *ParHub*), that addresses these two issues. It contributes solutions in the following way. First, the ParHub makes available a growing number of tools "on the Web" where they are accessible and executable through standard Web browsers. A new tool can be installed as is, providing dynamically generated user interfaces and, if necessary, being served directly off of the home site of a proprietary provider. Nevertheless, the authorized user can access the tool via standard Web browsers. Our work is related to other projects that have provided end users with the means to transparently access and use distributed resources. Examples are CCS [1], MMM [2], NetSolve [3], Ninf [4], RCS [5], Rivendell [6], and Schooner [7].

We address the second issue by providing, as part of the ParHub, a family of tools that are integrated in that they support a common underlying programming methodology and they can exchange intermediate data. In this regard, our work is related to work done to support different phases of the parallel program development cycle. Examples of such tools are Fortran D editor/Pablo [8], the Annai Tool Project [9], SUIF Explorer [10], and KAP/Pro Toolset [11].

The Parallel Programming Hub is being built in a project called NETCARE (NETwork com-

*This work was supported in part by NSF grants #9703180-CCR, #9872516-EIA, and #9975275-EIA. This work is not necessarily representative of the positions or policies of the U. S. Government.



(a) (b)

Figure 1: Entering the Parallel Programming Hub: (a) the main interface and (b) the user account request interface.

puting for Computer Architecture Research and Education), which uses a well-developed and established online-computing infrastructure called PUNCH (Purdue University Network Computing Hubs). In Section 2 we give an overview of the tools in the Parallel Programming Hub and its underlying technology. Section 3 evaluates several aspects of the ParHub. More detailed descriptions of the above related projects are given in Section 4. Section 5 concludes the paper.

2 The Parallel Programming Hub

In this section, we describe the tools available in the Parallel Programming Hub and the underlying technology that enables this tool environment. On the surface, the Parallel Programming Hub looks like a web server, through which users can run various parallel programming tools. Underneath this interface is an elaborate network computing infrastructure, called the Purdue University Network Computing Hubs (PUNCH).

Upon entering the main page, as shown in Figure 1.a, users see the list of tools available for use. The links to individual tools and the relevant documents are found here. A new user can request an account via the interface shown in Figure 1.b. Upon registering, a user is given an account and disk space that is accessible as long as the user is on PUNCH.

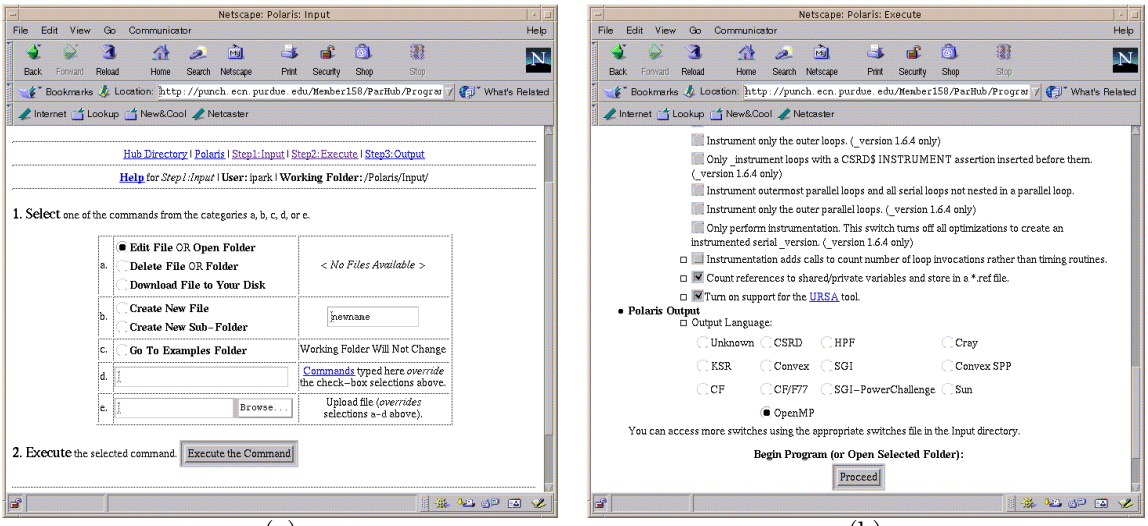
2.1 Tools in the Parallel Programming Hub

2.1.1 An Integrated OpenMP Environment: Polaris, Ursa Minor, MaxP

We have described our goals of providing tools that support a common programming and performance tuning methodology. A practical approach to parallel program optimization is to apply various parallelization techniques, execute and profile the resulting program, identify performance bottlenecks, and apply possible remedies. This process repeats until there are diminishing returns. Our methodology is based on this approach. It is supported by an environment that includes a parallelizing compiler, a parallelism analysis tool, and a performance evaluation and visualization tool. In the following we give an overview of each tool.

The Polaris parallelizing compiler [12] is an advanced source-to-source restructurer. Polaris can automatically find parallelism and express it in OpenMP form [13]. In our parallel programming projects, Polaris provides a good starting point for parallelizing and performance tuning scientific and engineering applications. On the Parallel Programming Hub, Polaris can be run through a form-based interface as shown in Figure 2.b. Figure 2.a shows the form to specify a Polaris input file.

The dynamic evaluation of a program can provide insights into behavior that remains undetected by static analysis methods. MAX/P is a Polaris-based tool that evaluates the inherent parallelism



(a) (b)

Figure 2: Polaris User Interface on the Parallel Programming Hub: (a) the input file interface and (b) the execution interface.

of a program at runtime. The tool can find the minimum execution time of a program assuming the availability of an unlimited number of processors. It can also find the parallelism that remains undetected by Polaris. The URSA MINOR performance evaluation tool described below, can incorporate this information into its displays and decision making process.

The programmers' intervention is often necessary to improve the performance of a compiler-parallelized program. To aid this process, the ParHub provides a performance evaluation tool, URSA MINOR [14], which combines performance analysis with the compiler's static program analysis information. With its support for OpenMP and the ability to understand the compiler analysis data, it complements the Polaris compiler. It collects and combines information from various sources, and its graphical interface provides selective views and combinations of the gathered data. URSA MINOR consists of a database utility, a visualization system for both performance data and program structure, a searching and viewing tool, and a file management module. In addition, it provides an active performance guidance system, which can analyze performance problems and suggest remedies. Figure 3 shows URSA MINOR in use on the Parallel Programming Hub.

The users who would like to improve the performance of their applications can follow the scenario given below. First, they upload their program into their ParHub account. Then they can run Polaris to instrument and parallelize the program. After the execution of the parallel program, the performance profiles generated from these runs and the outputs of the Polaris compiler can be loaded into URSA MINOR for evaluation. Users may add manual changes to the program based on their findings or run MAX/P to obtain detailed information on the parallel nature of the target program. Throughout this optimization and evaluation process, URSA MINOR provides means to visualize and reason about the gathered data.

2.1.2 A Growing Number of ParHub Tools

An increasing number of tools are being made available through the ParHub. Currently, the Trimaran environment for instruction level parallelism [15] and the SUIF parallelizing compiler [16] are accessible. In addition, authorized users can access a number of common support tools such as Matlab, Mentor Graphics, GNU Octave, and StarOffice (Only Purdue users are authorized to use commercial tools). In the present environment, users cannot run the generated parallel programs on the ParHub machines. Users' executable programs currently need to be run outside the ParHub. In the near future, users will be able to both develop and experiment on the ParHub, and have access to powerful high-performance computer systems. "Experiment modes," such as single-user runs, will be provided to support an adequate computer systems research environment.

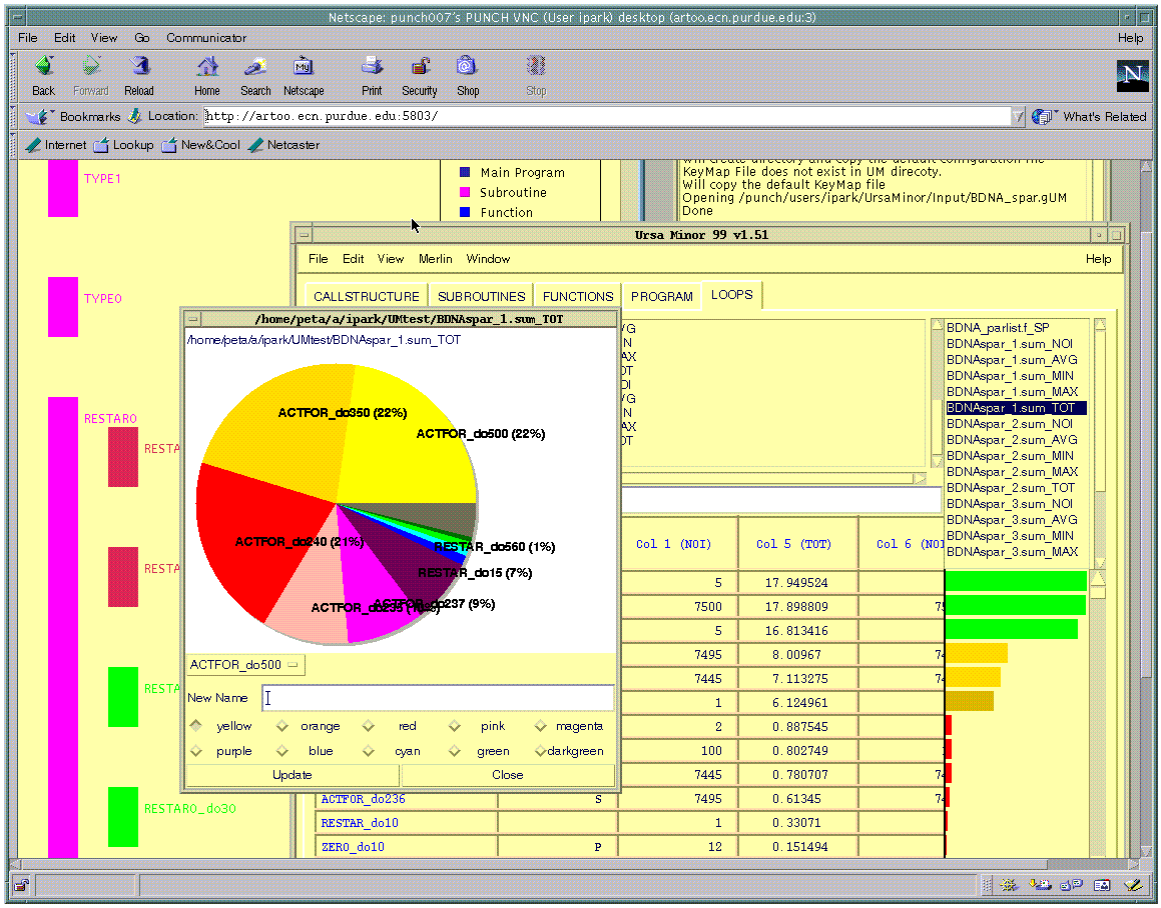


Figure 3: URSA MINOR usage on the Parallel Programming Hub.

2.1.3 ParHub User Interfaces

Users can easily find proper documentation for the tools in the Parallel Programming Hub. All these tools come with brief descriptions for casual browsing, manuals for detailed usage information, and new user's guides or demo scenarios. The PUNCH system allows applications to be installed with either a configurable form-based or a graphics-based user interface. For batch-oriented tools, users set relevant parameters to control the tool runs through a form-based interface as shown in Figure 2.b. For interactive tools, users are presented with a flexible window-based interface similar to Figure 3, through the VNC display management utility [17], which allows access to the video buffer of a remote server (VNC can also be used via its helper application). In either case, users can edit, delete, create or upload files in their account directories before or after running the application. A typical file interface is shown in Figure 2.a. The Parallel Programming Hub is available at http://www.ece.purdue.edu/punch/ParHub/.

2.2 The Underlying Technology: PUNCH

From a user's perspective, PUNCH is a Web portal that allows users to access and run software tools via standard Web browsers. PUNCH allows tools from different geographical locations to be indexed and cross-referenced, and makes them available to users world-wide. The infrastructure hides all details associated with the remote invocation of tools from its users. Existing tools can be installed without modification. Currently, PUNCH provides access to tools for parallel programming, computer architecture, semiconductor technology, and VLSI design; fifty tools developed by six vendors and thirteen universities are available.

From an infrastructure standpoint, PUNCH consists of two parts, as shown in Figure 4. The network desktop allows users to interact with the network-computing infrastructure (SCION) via

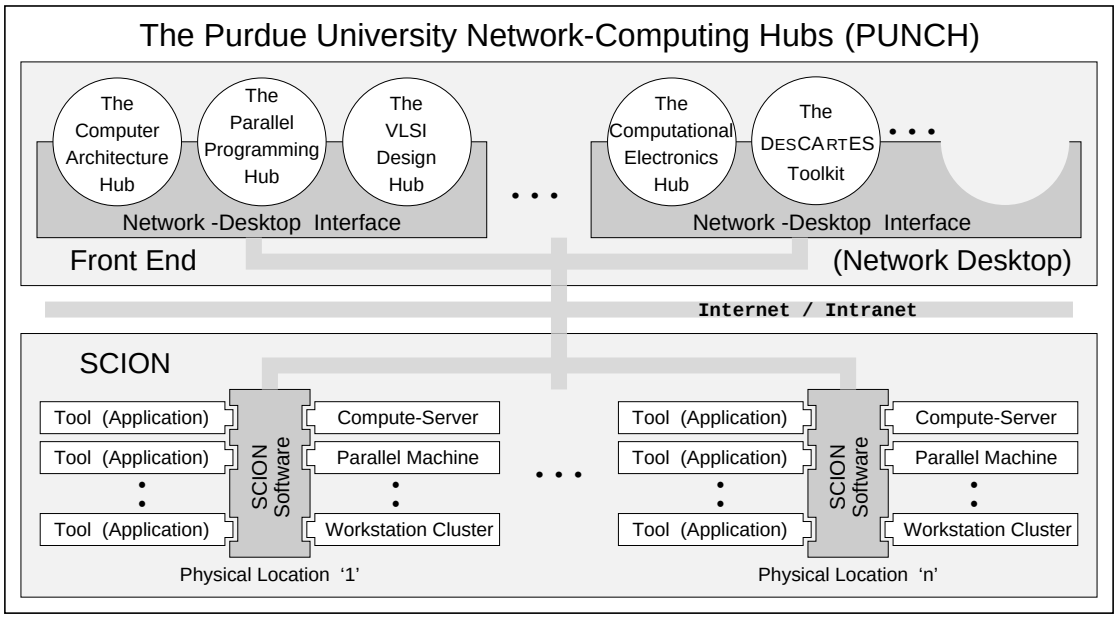


Figure 4: The main components of the PUNCH infrastructure: the network desktop manages data and interfaces, and SCION manages resources.

standard Web browsers, and generates customized views of available resources for each class of users. SCION serves as PUNCH's user-transparent middleware. It consists of hierarchically distributed servers that cooperate to provide on-demand network-computing. SCION manages the run-time environment, controls resource access and visibility, and manages available resources.

A network-computing system such as PUNCH can be characterized in terms of four parameters: 1) the interface to the external world, 2) the internal design, 3) the supported class of software and hardware resources, and 4) the resource management capabilities.

External system interface. PUNCH provides a universally-accessible, Web-based interface to a network-based computing environment. The system supports logical organization of resources (i.e., hierarchical indexing and cross-referencing) across multiple administrative domains, and can present users with a dynamically customized (individualized) view of resources [18]. PUNCH users have their private, logical, network-accessible user-accounts; the system manages its own physical accounts on remote resources and dynamically recycles them among users as necessary. Data files distributed across the network are retrieved in an application-transparent manner via a virtual file system based on the UFO prototype [19].

Internal system design. PUNCH employs a hierarchically distributed architecture in order to process requests in an efficient and a scalable manner [20]. The number of levels in the hierarchy can be increased as the system grows, which further contributes to the scalability of the core architecture. The infrastructure supports multiple, concurrent users and processes (jobs), and uses resource replication to address single-point failures. The PUNCH infrastructure can be deployed on heterogeneous platforms across multiple administrative domains. This is accomplished by partitioning the system into cells, each of which is independently-managed [20]. PUNCH users do not need physical accounts on the remote resources available to the system.

Support for software resources. The PUNCH infrastructure is designed to allow administrators to integrate (install) unmodified tools with little effort; access to source or object code is not required. Stand-alone tools with batch, text-based interactive, and graphical user interfaces are supported [21]. Tools are installed using a specifically-designed high-level language that describes the behavior of the tool to the PUNCH infrastructure [21]. This description is automatically compiled and integrated into PUNCH's knowledge management system. The Web-based user interfaces of the tools are dynamically generated from information specified in *HTML templates* [21]. Tools with simple interfaces can be installed in as

little as thirty minutes.

Resource management capabilities. PUNCH employs an application management framework that allows it to focus on achieving the best possible cost and performance tradeoff at runtime, based on specifications of acceptable cost and desired performance. Cost figures are provided by administrators. Performance is estimated by a prediction system [20] that utilizes machine learning techniques in order to estimate run-specific resource requirements before a scheduling decision is made. PUNCH utilizes a non-preemptive, decentralized, adaptive, sender-initiated resource management framework; the actual scheduling algorithms can be selected dynamically on the basis of predicted run-specific resource-usage information.

3 Evaluation

In this section we compare the Parallel Programming Hub with alternative tool environments and access methods. We provide qualitative and quantitative measures for its accessibility, integration, and efficiency.

3.1 Qualitative Evaluation: Accessibility and Integration

The immediate advantage of accessing tools in a network-based tool laboratory is the large savings in user efforts and computer resources. Typically, in order to make use of a tool without such an environment, a user would first locate (e.g., using a Web search engine) a downloadable tool that appears to fit the need. To install the tool the user will need to obtain the right machine resources (E.g., correct computer type, sufficient memory) and may need to perform software adaptations and configurations. Local user guides may need to be written after tool modifications. In our experience, this process can take from a few hours to several weeks. The unpredictability of this time is a critical disadvantage. In contrast, the Parallel Programming Hub is a network-based integrated tool laboratory that benefits users by providing access to already working tools and their documentation regardless of any client machine types. A typical tool access time for first time users of the ParHub is a minute, including authentication and navigating to the right tool.

Once a working set of tools is available for a given task, the degree of integration of these tools becomes important. Integrated tools provide a uniform user interface. One challenge for the tool developer is to do this without compromising individuality of necessary tool features. Furthermore, data formats need to be compatible, such that tools can produce, consume, and comprehend each others information. Perhaps most important, tools must appear to the user as speaking the same language and supporting a common methodology. While it is not possible to attribute a precise cost to the lack of integration of the current generation of programming tools, in our experience this cost is high. The large number of tools needed for developing and performance tuning parallel programs aggravate this problem further. The ParHub provides solutions in two ways. First, it provides a set of tools that are directly based on a common parallel programming methodology [22]. The user benefits by seeing the same terminology used across the tools and by being able to include in a tool the data files produced by the other tools. Second, the ParHub provides a common user interface that presents documentation, help, tutorials, and data input/output dialogs in a uniform way. At the same time, it allows unmodified tools to be installed, providing for necessary uniqueness of tool features. In our experience, this level of integration has led to a significant increase in work efficiency for tool users.

3.2 Quantitative Evaluation: Efficiency

Batch-oriented tools run as efficiently on the ParHub as on local platforms. In fact, thanks to the PUNCH system's powerful underlying machine resources, most users' tools have faster response times on the ParHub. Interactive tools need closer inspection. In order to provide a quantitative evaluation of interactive ParHub tools, we have selected the Ursa Minor tool described above. For comparison, we have implemented an alternative network access method via Java Applets. We call the two tool versions UM/ParHub and UM/Applet[1], respectively.

A typical tool interaction with Ursa Minor causes the tool to fetch from a repository a program database that represents a specific parallel programming case study. It then performs various operations on this databases and displays the results using Ursa Minor's visualization utilities. Table 1 shows how server, client, and file operations are invoked by various tasks in the tool usage.

[1] In [14] we refer to this tool as Ursa Major

tasks	UM/ParHub	UM/Applet
application execution	server	client Applet
database load	local disk IO + server	network transfer + client Applet
display	network transfer + client Applet (VNC)	client Applet

Table 1: Workload distribution on resources for the URSA MINOR operations selected for the experiment.

In a typical interactive tool session, a user loads input files, runs computing utilities on the data, and adds more files for further manipulation. From this scenario, we chose three tool operations. We have measured the time taken to load a database, perform a simple spreadsheet-like operation on the data, and search and display a portion of the source codes. The database load is an example of loading input data, while spreadsheet command evaluation is representative of computing on the data. Source search operation requires a simple string search through a source code. Interestingly, these three operations exhibit different patterns in resource usage. For UM/Applet, the database load operation requires downloading the database, parsing it, and updating the display appropriately. Hence, it exercises both networking and computing capabilities. The second operation, evaluation of a spreadsheet command, performs a mathematical operation on the data that the Applet already has downloaded, so it only involves computing on a client machine. The search operation mainly relies on networking. A source file is not part of the database, hence it has to be downloaded separately. In UM/ParHub, data transfer over the network is replaced by file IO. However, the response to a user action has to be updated on the display of the remote client machine.

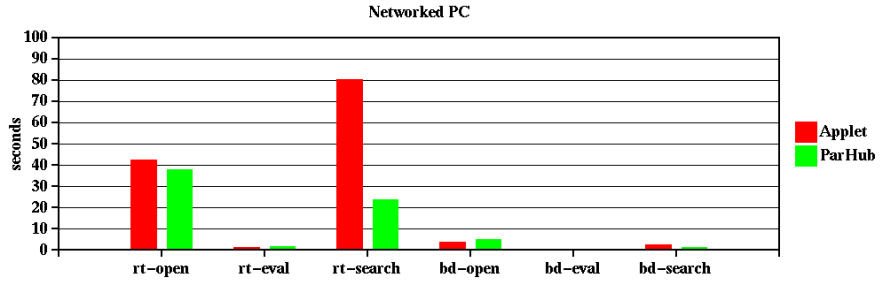
We chose two different databases in this experiment, representing a small and a large application study, respectively. The first database contains tuning information of the program BDNA from the Perfect Benchmarks [23]. The database size is about 72 Kbytes, and the accompanying source file is about 283 Kbytes. We consider this to be a small database. The second database contains information about the parallelization of the RETRAN code [24], which represents a large, power plant simulation application. The database we used is 463 Kbytes in size, and the size of the source is about 5.4 Mbytes.

Finally, we chose three machines on which we measured the tool response times. "Networked PC" is a PC with 300MHz Pentium II and 64 Mbytes of memory. Its operating system is WindowsNT. It is connected to the Internet through a 10 Mbps ethernet card. "Dialup PC" is a home PC with 233MHz Pentium II processor and 64 Mbytes of memory. Its operating system is Windows95, and its connection to the Internet is through 33.8K modem and via a local ISP. The third machine, "Networked Workstation", is an UltraSPARC workstation with 167 MHz processor and 160 Mbytes of memory. Its operating system is SunOS v5.6. The network bandwidth is 10 Mbps.

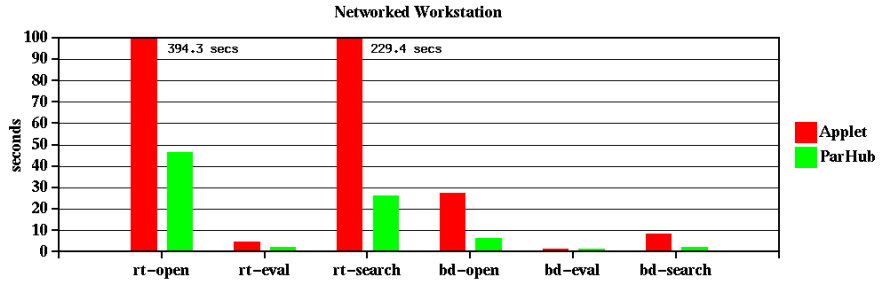
We have measured the response time of the three operations at 8-hour intervals over several days using a Netscape browser v4.7. We have inserted timing functions for UM/Applet and used an external wallclock for UM/ParHub. We made 9 measurements for each case. Figure 5 shows the average response time in seconds on the three machines. The figure shows the three measured tool operations. The average standard deviation for the measurements is 15.9 %. The maximum standard deviation is 54 %, which is the case with the evaluation operations on the dialup PC, which take 5 seconds or less.

Overall, the networked PC exhibits the shortest response time for all operations. On this machine, the response times of the Hub version and the Applet version are comparable. However, downloading of a large program source significantly increases the response time of the search operation, despite the fast network connection. In the case of UM/ParHub, files are read through file IO within the server, thus the network is not a dominating factor. The dialup PC displays adequate response time except for the search operation with UM/Applet. The network bottleneck is even more pronounced in this case. The performance of the networked workstation does not degrade substantially due to the network connection. However, its slow processor and the overhead of the Java Virtual Machine (JVM) make it the worst performing platform among the three.

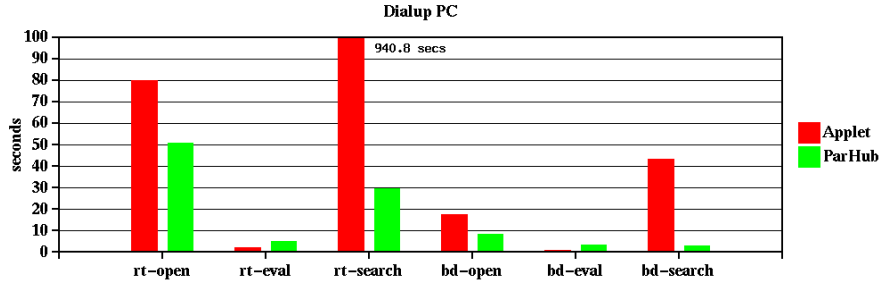
The response time on three different machines for each operation, as shown in Figure 6, offers a different perspective. We only present the data regarding the operations on the RETRAN database because those on the BDNA database show similar trends and the characteristics are more pronounced in the RETRAN case. The response time of UM/ParHub does not show noticeable variations



(a)



(b)



(c)

Figure 5: The response time of UM/Applet and UM/ParHub: on (a) a networked PC, (b) a networked workstation, and (c) a dialup PC. "rt-load" refers to the response time to load the RETRAN database. "rt-eval" and "rt-search" refer to the time to perform spreadsheet command evaluation and source search, respectively. The data tags with prefix "bd" refer to the same operations on the BDNA database.

on all three machines except on the dialup PC. The spreadsheet command evaluation takes more than twice as long on the dialup PC compared to the others. This operation executes in the order of seconds, so a screen update becomes a factor with the slow modem connection. For UM/Applet, the platform becomes a deciding factor. If the network is slow, the performance of the search operation degrades. For compute-intensive operations, the machine speed and the JVM overhead determine the response time. In all, the ParHub-based tool performs better than the Applet-based version.

Our experiments show that the Parallel Programming Hub offers users a fast and stable solution to interactive network computing. The network transmits only users' action (pressing buttons and clicking a mouse) and screen updates to and from the server, so the network or processor speed has little impact on the tool usage in our experiment. By contrast, Applet-based tools rely on the client machine for computation and on the network for data transfer. Thus, if the amount of data is large or the client machine is slow, the resulting operations take considerably longer. The two networked machines we used are located within the Purdue network. We expect these performance characteristics to be even more pronounced on geographically distributed machines.

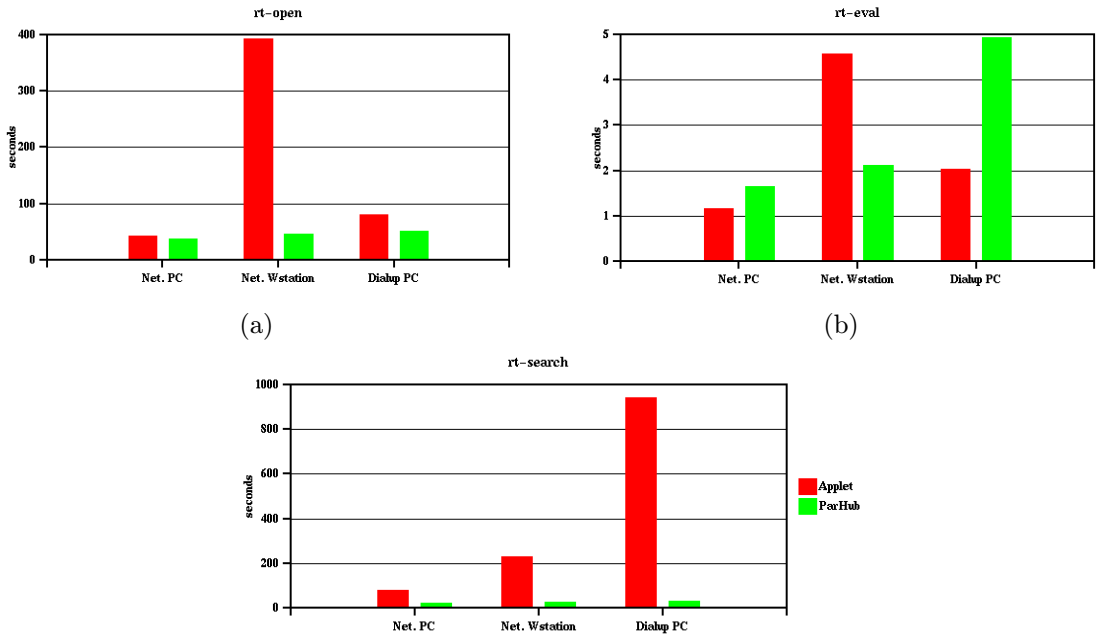


Figure 6: The response time of the three operations on RETRAN database: (a) loading, (b) spreadsheet command evaluation, and (c) source searching.

4 Related Work

In this section we compare the Parallel Programming Hub with related efforts. We do this from the angle of its major two contributions: advances in network computing technology and in parallel programming environments.

4.1 Network Computing

Our work on PUNCH was motivated by the fact that many of the systems and technologies that currently allow computing on the Web support a single or a relatively small set of tools. For example, a large number of systems (such as the Exploratorium [25], and JSPICE [26]) are based on scripts that need to be modified in order to add any new application to the system. Other designs are more flexible. The MOL [27] prototype, for example, employs static Web interfaces that can be adapted for individual tools. The NetSolve [3], Ninf [4], and RCS [5] systems are based on structured designs that target numerical software libraries. However, static interfaces are not adequate for all tools, and structured approaches cannot be easily applied to general-purpose applications. Another problem is that the majority of these designs assume the availability of the source and/or object code for the applications — which effectively precludes the installation of most commercial tools.

Other systems provide components for network computing environments. For example, VNC [28] and WinFrame [29] provide very flexible mechanisms for exporting graphical displays to remote consoles in a platform-independent manner. Globus [30] and Legion [31] provide low-level distributed computing services for heterogeneous computing environments.

By contrast, PUNCH provides a production wide-area computing framework and system, which includes all layers up to the user interface. The computing infrastructure is not tied to the characteristics of individual applications. Functionally, this is equivalent to designing a multi-user *operating system* for networked resources that provides user-transparent file, process, and resource management functions, handles security and access control across multiple administrative domains, and manages state information (session management). The need for such a system and the associated benefits in the context of the World Wide Web have also been recognized elsewhere [32, 33].

4.2 Parallel Programming Environments

The important role of tools to aid the process of parallel program development and performance tuning is widely acknowledged. Among the supporting tools are those that perform automatic parallelization, performance visualization, instrumentation, and debugging. Many of the current tools are summarized in [34, 35].

We have argued that state-of-the-art tools are not well integrated. Related projects have attempted to resolve this issue by broadening the support for different stages of the parallel program development process and by creating more complete parallel programming environments. These efforts include Faust [36], Fortran D editor/Pablo [8], the Annai Tool Project [9], SUIF Explorer [10], and the KAP/Pro Toolset [11]. Faust encompasses code optimization and performance evaluation with the emphasis on project management. Fortran D editor/Pablo and SUIF Explorer aim to integrate parallelization and performance evaluation. The Annai Tool Project focuses on the aspects of parallelization, debugging, and performance monitoring. The main features of the KAP/Pro Toolset include parallelization, performance evaluation, and parallel program debugging. In contrast to these tools, the ParHub presents an environment in which its tools are globally accessible and directly executable. The tools are integrated in that they support a common methodology and provide uniform user interfaces for many functions. Furthermore the ParHub is scalable in that the addition of a large number of new tools is possible and well facilitated.

5 Conclusions

We have described the Purdue Parallel Programming Hub (ParHub), a Web-executable tool environment that can be accessed via standard browsers. The ParHub provides a growing number of parallel programming tools. Among them, we have described an integrated set of tools that support a common programming methodology. Our main contributions are (1) to provide a new infrastructure for "anytime, anywhere" access of both new and existing tools, and (2) to provide an integrated parallel programming tool environment. We believe that both of these tool aspects are of decisive importance for successful programming tools. Such tools, in turn, represent the backbone of the future generation of software development environments.

We have quantitatively compared an interactive tool of the ParHub with an equivalent, Java Applet-based tool. We have found significant performance advantages of the Hub-based tool. They primarily stem from the fact that interactive operations on tool databases need only communicate the initial command and result from and to the user. In contrast, Applet-based tools would transfer the database to the user's site and perform all operations locally.

A novel aspect of the ParHub's underlying technology is that it represents not only an actual "information grid", but also includes the necessary portals for its end users. One vision is that future users can access software tools via any local platform – from a palmtop to a powerful workstation. Compute power and file space is provided "on the Web". Mobility is provided in that these resources are accessible transparently from any access point. The described infrastructure represents a significant step towards this vision.

References

[1] F. Ramme. Building a virtual machine-room-a focal point in metacomputing. *Future Generations Computer Systems*, 11(4–5):477–489, August 1995.

[2] O. Gunther, R. Muller, P. Schmidt, H. K. Bhargava, and R. Krishnan. MMM: a web-based system for sharing statistical computing modules. *IEEE Internet Computing*, 1(3):59–68, May–June 1997.

[3] H. Casanova and J. Dongarra. NetSolve: a network enabled server for solving computational science problems. *International Journal of Supercomputer Applications*, 11(3):212–223, Fall 1997.

[4] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: a network-based information library for global world-wide computing infrastructure. In *Proc. of High-Performance Computing and Networking, International Conference and Exhibition*, pages 491–502, April 1997.

[5] P. Arbenz, W. Gander, and M. Oettli. The Remote Computation System. *Parallel Computing*, 23(10):1421–1428, October 1997.

[6] C. E. Kaiser, S. E. Dossick, J. Wenyu, J. J. Yang, and X. Y. Sonny. WWW-based collabo-

ration environments with distributed tool services. *World Wide Web*, 1(1):3–25, October 1998.

[7] P. T. Homer and R. D. Schlichting. Configuring scientific applications in a heterogeneous distributed system. *Distributed Systems Engineering*, 3(3):173–184, September 1996.

[8] V. S. Adve, J. Mellor-Crummey, M. Anderson, K. Kennedy, J. C. Wang, and D. A. Reed. An integrated compilation and performance analysis environment for data parallel programs. In *Proc. of Supercomputing Conference*, pages 1370–1404, 1995.

[9] B. J. N. Wylie and A. Endo. Annai/PMA multi-level hierarchical parallel program performance engineering. In *Proc. of International Workshop on High-Level Programming Models and Supportive Environments*, pages 58–67, 1996.

[10] W. Liao, A. Diwan, R. P. Bosch Jr., A. Ghuloum, and M. S. Lam. SUIF explorer: An interactive and interprocedural parallelizer. In *Proc. of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 37–48, August 1999.

[11] Kuck and Associates Inc. *KAP/Pro Toolset*, 2000. http://www.kai.com.

[12] William Blume, Ramon Doallo, Rudolf Eigenmann, John Grout, Jay Hoeflinger, Thomas Lawrence, Jaejin Lee, David Padua, Yunheung Paek, Bill Pottenger, Lawrence Rauchwerger, and Peng Tu. Parallel programming with polaris. *IEEE Computer*, 29(12):78–82, December 1996.

[13] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *Computing in Science and Engineering*, 5(1):23–30, January 1999.

[14] Insung Park, Michael J. Voss, Brian Armstrong, and Rudolf Eigenmann. Parallel programming and performance evaluation with the URSA tool family. *International Journal of Parallel Programming*, 26(5):541–561, November 1998.

[15] Trimaran Homepage. *Trimaran Manual*, 2000. http://www.trimaran.org/docs.html.

[16] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S-W. Liao, E. Bugnion, and M. S. Lam. Maximizing multiprocessor performance with the SUIF compiler. *IEEE Computer*, 29(12):84–89, December 1996.

[17] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, jan–feb 1998.

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

[19] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. UFO: A personal global file system based on user-level extensions to the operating system. *ACM Transactions on Computer Systems*, 16(3):207–233, August 1998.

[20] Nirav H. Kapadia and José A.B. Fortes. PUNCH: An architecture for web-enabled wide-area network-computing. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 2(2):153–164, September 1999. In special issue on High Performance Distributed Computing.

[21] Nirav H. Kapadia, Renato J. Figueiredo, and José A. B. Fortes. PUNCH: Web portal for running tools. *IEEE Micro*, pages 38–47, May-June 2000.

[22] Paramount Research Group, Purdue University. *Program Parallelization and Tuning Methodology*, 2000. http://min.ecn.purdue.edu/~ipark/UMinor/meth_index.html.

[23] Lynn Pointer. Perfect: Performance evaluation for cost-effective tranformations report 2. Technical Report 964, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, March 1990.

[24] T. J. Downar, Jen-Ying Wu, J. Steill, and R. Janardhan. Parallel and serial applications of the RETRAN-03 power plant simulation code using domain decomposition and Krylov subspace methods. *Nuclear Technology*, 117(2):133–150, February 1997.

[25] C. Adasiewicz. Exploratorium: User friendly science and engineering. *NCSA Access*, 9(2):10–11, 1995.

[26] D. Souder, M. Herrington, R. P. Garg, and D. DeRyke. JSPICE: A component-based distributed Java front-end for SPICE. *Concurrency Practice and Experience*, 10(11–13):1131–1141, September–November 1998.

[27] A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza, F. Ramme, T. Romke, and J. Simon. The MOL project: An open, extensible metacomputer. In *Proc. of the 1997 IEEE Heterogeneous Computing Workshop*, pages 17–31, 1997.

[28] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January–February 1998.

[29] Citrix. *ICA technical paper*, 1996. http://www.citrix.com/products/ica.asp.

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

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

[32] S. B. Lamine, J. Plaice, and P. Kropf. Problems of computing on the web. In *Proc. of the 1997 High Performance Computing Symposium*, pages 296–301, 1997.

[33] F. D. Reynolds. Evolving an operating system for the web. *Computer*, 29(9):90–92, September 1996.

[34] J. Brown, A. Geist, C. Pancake, and D. Rover. Software tools for developing parallel applications. 1. code development and debugging. In *Proc. of Eighth SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.

[35] J. Brown, A. Geist, C. Pancake, and D. Rover. Software tools for developing parallel applications. 2. interactive control and performance tuning. In *Proc. of Eighth SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.

[36] Vincent Guarna Jr., Dennis Gannon, David Jablonowski, Allen Malony, and Yogesh Gaur. Faust: An integrated environment for the development of parallel programs. *IEEE Software*, 6(4):20–27, July 1989.