DARP: Java-based Data Analysis and Rapid Prototyping Environment for Distributed High Performance Computations

Erol Akarsu, Geoffrey Fox, Tomasz Haupt

Northeast Parallel Architecture Center at Syracuse University

Abstract:

The integration of a compiled and interpreted HPF gives us an opportunity to design a powerful application development environment targeted for high performance parallel and distributed systems. This Web based system follows a three-tier model. The Java front-end holds proxy objects which can be manipulated with an interpreted Web client (a Java applet) interacting dynamically with compiled code through a tier-2 server. Although targeted for HPF back-end, the system's architecture is independent of the back-end language, and can be extended to support other high performance languages

1. Introduction

The development of large applications is a complex process and the assistance of adequate tools is always very welcome. Not surprisingly, there are numerous commercially available tools for this purpose. Visual debuggers, profilers, data analysis and visualization packages are integral parts of the environments on the workstations of scientists and engineers.

The situation is different for high-performance, parallel, or distributed architectures. Performance tuning, debugging, and data analysis is more difficult, and yet tools for these are not widely available. We also face a problem of software integration, as different software components often follow different parallel programming paradigms. On the other hand, we witness a rapid progress of Web-based technologies that are inherently distributed, heterogeneous, and platform-independent. Of particular interest are the definition and standardization of interfaces that enable cross-platform software interoperability. In this report we describe a system that borrows from these Web technologies.

The integration of compiled and interpreted HPF gives us an opportunity to design a powerful application development environment targeted for high performance parallel and distributed systems. This DARP environment includes a source level debugger, data visualization and data analysis packages, and an HPF interpreter. A capability to alternate between compiled and interpreted modes provides the means for interaction with the code at real-time, while preserving an acceptable performance. Thanks to interpreted interfaces, typical of Web technologies, w can use our system as a software integration tool.

The fundamental feature of our system is that the user can interrupt execution of the compiled code at any point and get an interactive access to the data. For visualizations, the execution is resumed as soon as the data transfer is completed. For data analysis, the interrupted code pauses and waits for the user's commands. The set of available commands closely reproduces functionality of a typical debugger (setting breakpoints, displaying or modifying values of variables, etc.). However, our system has a unique feature that values of distributed arrays can be modified by issuing HPF commands. In this sense, our system can be thought of as an HPF interpreter. For more complex data transformations, the user can dynamically link precompiled functions written in HPF or other languages. This enables rapid prototyping. In particular, parallel libraries that do not necessarily follow the HPF computational model can this way be integrated dynamically with the HPF code, through the HPF extrinsic interface.

The functionality of our system is further increased by implementing proxy libraries in Java. This allows us to design and develop DARP system as a three-tier system rather than a traditional client-server one. Now we can treat components of the DARP system as distributed objects to be implemented as CORBA ORB-lets or JavaBeans. We use this mechanism for dynamical embedding of calls to a visualization system (such as SciViz[2]), or for coupling this system with the WebFlow[3].

The paper is organized as follows. In section 2 we discuss the overall architecture of the system in the context of the High Performance Commodity Computing paradigm. Sections 3-6 describe the three-tier design of the DARP system and its components: the tier-2 DARP server, the instrumentation server, DARP front-end and HPF interpreter, respectively. Section 7 demonstrates integrating the DARP system with the visualization package using a proxy library. Finally, in Section 8 we give our summary and conclusions.

2. High Performance Commodity Computing

In recent years we observed impressive and rapidly improving software artifacts, both raw technologies and open interfaces which enable development of distributed modular software. These interfaces are at both low and high levels and the latter generate a very powerful software envi-

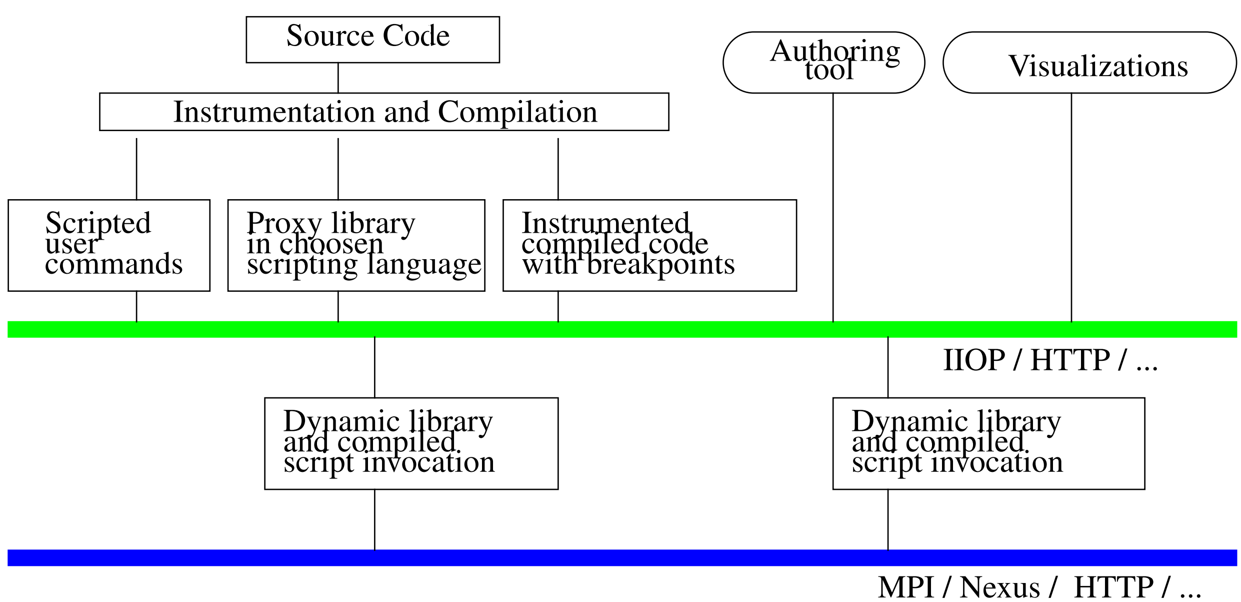


Figure 1: DARP implementation within HPcc framework

ronment in which large pre-existing components can be quickly integrated into new applications. We believe that there are significant incentives to build HPCC environments in a way that naturally inherits all the commodity capabilities so that HPCC applications can also benefit from the impressive productivity of commodity systems.

The growing power and capability of commodity computing and communication technologies is largely driven by commercial distributed information systems[1]. CORBA, Microsoft's DCOM, JavaBeans, and less sophisticated web and network approaches build such systems. These can all be abstracted to a three-tier model with largely independent clients connected to a distributed network of severs. These host various services including object and relational databases and of course parallel and sequential computing. High performance can be obtained by combining concurrency at the middle server tier with optimized parallel back end services. The resultant system combines the needed performance for large-scale HPCC applications with the rich functionality of commodity systems.

The design of the DARP system follows this idea of High Performance commodity computing (HPcc). Conceptually, the architecture of this three-tier system can be described as follows (c.f. Figure 1): The DARP system uses an interpreted Web client interacting dynamically with a compiled code. At this time, the system uses HPF back-end but the architecture is independent of the back-end language. The Java or JavaScript front-end holds proxy objects produced by an HPF front-end operating on the back-end code. These proxy objects can be manipulated with interpreted Java commands to request additional processing, visualization and other interactive computational steering and analysis.

3. DARP server: Interactive control over an application

As shown in Figure 2, the heart of the DARP system is the DARP server, which controls the execution of the application. The server accepts commands from a client implemented as a Java applet. The control over the execution of an application and an interactive access to the data is achieved by a source level instrumentation of the code.

Since HPF follows a simple global name space, data parallel paradigm, the DARP server can be implemented just as an extrinsic HPF LOCAL procedure. In this case, the server is a part of the application, and it comes into existence only after the application is launched. In this scenario, the application code is instrumented in such a way that the initialization of the DARP server is the first executable statement of the application. Once initialized, the server blocks the application waiting for the client to connect. From that point on, the execution is controlled by the client. Optionally, the initialization of the server may include processing a script that sets action/breakpoints and force resuming the execution without waiting for the user's commands.

In a general SPMD paradigm this simplistic implementation of the DARP server is not sufficient: the client looses the control over the application when the code on a single node dies. Therefore we extended the server architecture. Now, as shown in Figure 3, the manager which is independent of the application accepts requests from the client and multicasts them to all nodes participating in the computations. The application is packaged as an Java object with two Java threads: the instrumented application itself and the control thread that shares the context of the application.

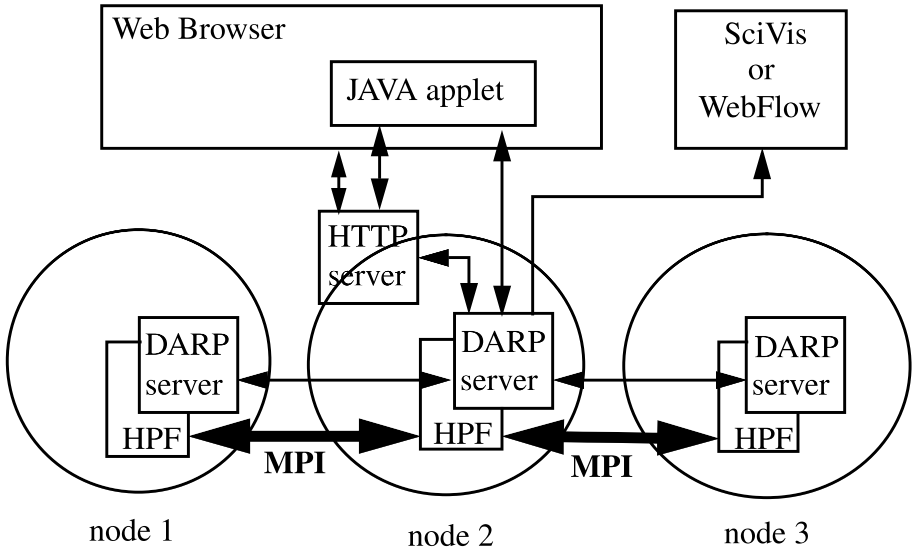
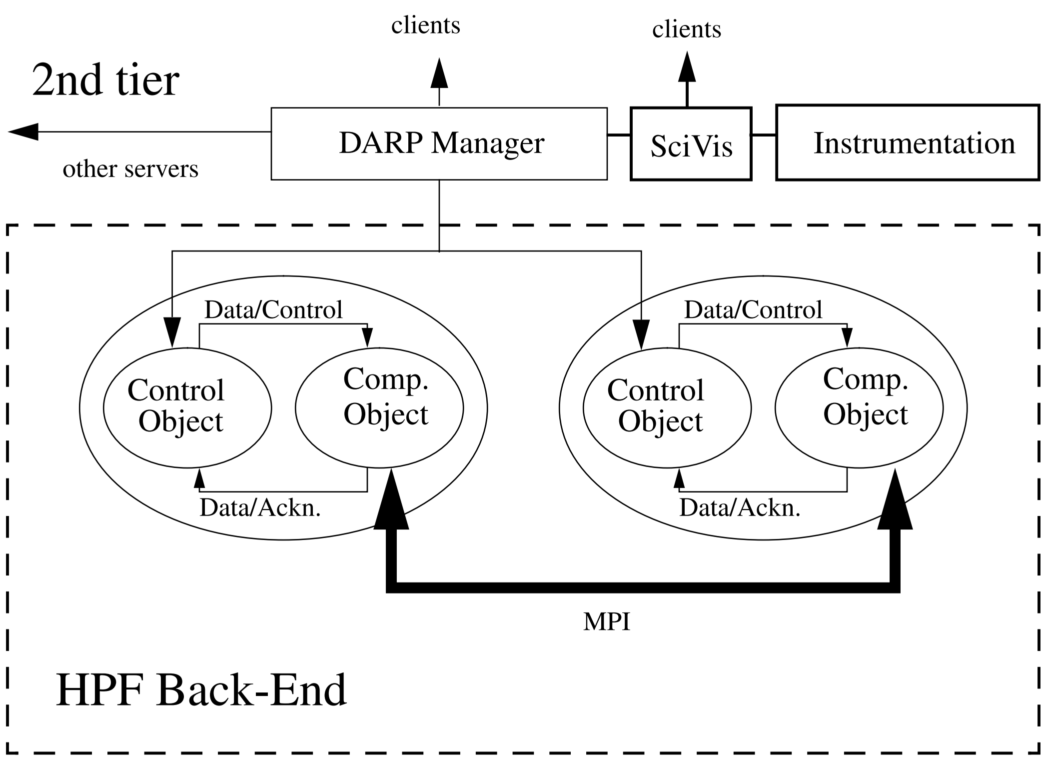


Figure 2: Current architecture of DARP. The DARP server is a part of the instrumented HPF application and it is replicated over the nodes participating in the computation. The client communicates with only one server on a selected node. On this node the server acts as a manager.

Figure 3. 2nd-tier DARP manager controls HPF back-end and communicates with other servers.



The interprocessor communication required by the distributed application is not implemented using the Web based protocols (such as CORBA IIOP), as is the case of the client-manager interactions. Instead, we use the native HPF runtime support or MPI directly. For a meta-computations, in our approach controlled by a network of managers, we consider replacing low level MPI by Nexus[5] and other services provided by Globus[6] as the high performance communication layer.

4. Instrumentation of the code

The instrumentation of the code involves three steps:
1. Adding server functions
2. Insertion calls to the HPF server before each HPF statement
3. Identification of the types of all variables used in the application

The process is fully automated and requires no user intervention. The instrumentation is done by a preprocessor that transforms a valid HPF source code into an instrumented one. The instrumented code is a valid HPF code, itself, to be compiled by a generic HPF compiler.

We build the preprocessor using the HPF Front End (HPFfe)[7] developed at NPAC within the PCRC consortium[8]. HPFfe is based on the SAGE++ system [9], which in addition to parsing, provides the means to access and modify the abstract parse tree (AST), the symbol table, the type tables, and source-level program annotations. For our purposes, we developed functions that identify attributes of all variables used in the HPF application (including the data type and runtime memory addresses) and operates on the AST to insert variables' "registration" calls (allowing the server to determine the size and location of the data to be sent) and calls to the server.

Since HPF is a superset of Fortran 90 we can apply our preprocessor to any sequential Fortran code, in particular, to a node code of any parallel application developed in Fortran that uses explicit calls to a message passing library such as MPI or PVM. Capability to process HPF compiler directives enhances our system in that we can preserve the information on the intended data distributions and assertions on the (lack of) data dependencies.

5. DARP Front-End

The DARP front-end is implemented as a Java Applet (see fig.5). The user interacts with the code through an interface that closely resembles the interface provided by a typical debugger. The repertoire of the commands includes setting break- and action-points, stopping/resuming execution of the application (including stepping one instruction at a time or one iteration of a loop at a time), changing values of the application's variables, requests for data (including distributed arrays), dynamically linking and executing shared objects (including codes generated in the fly by the interpreter), and more. For a detailed description of the interface, see [10].

Note that since the code is instrumented on the source level, our "debugger" gives access only to the source level data. In particular, we are not able to provide a complete state of the machine (registers, buffers, etc.) at any given time, as many commercial debuggers do, and as it is recommended by the High Performance Debugging Forum[11]. Also, since at this time we address exclusively applications in HPF, we ignore several important features that are necessary to support a more general SPMD paradigm. In particular, we assume that interprocessor communications are facilitated by a bug-free HPF runtime system. However, the more advanced implementation of the DARP system with the independent DARP manager (c.f. Section 3) makes it possible to control applications that use explicit message-passing. Anyway, the DARP system is not designed to be a system level debugger. The intended functionality of the system is manipulation of large distributed data objects to, for example, investigate convergence and stability of algorithms used in scientific simulations.

Typically, a client-server architecture is used to implement a portable debugger for distributed systems (c.f. [12-17]). Our approach is unique in that we use a three-tier architecture. Therefore we can easily integrate our source level debugger with the HPF interpreter and a visualization tool which together comprise a powerful application development environment.

6. Integrated Environment for HPF Compiler and Interpreter

The architecture of this system allows for real-time interaction with an executing HPF code. At each synchronization point (when the DARP server is accepting requests), the data can be extracted and processed as if an explicit call to an HPF extrinsic procedure was made. HPF statements, in particular, can be executed in such an interactive fashion. In this way, the system achieves the functionality of an HPF interpreter.

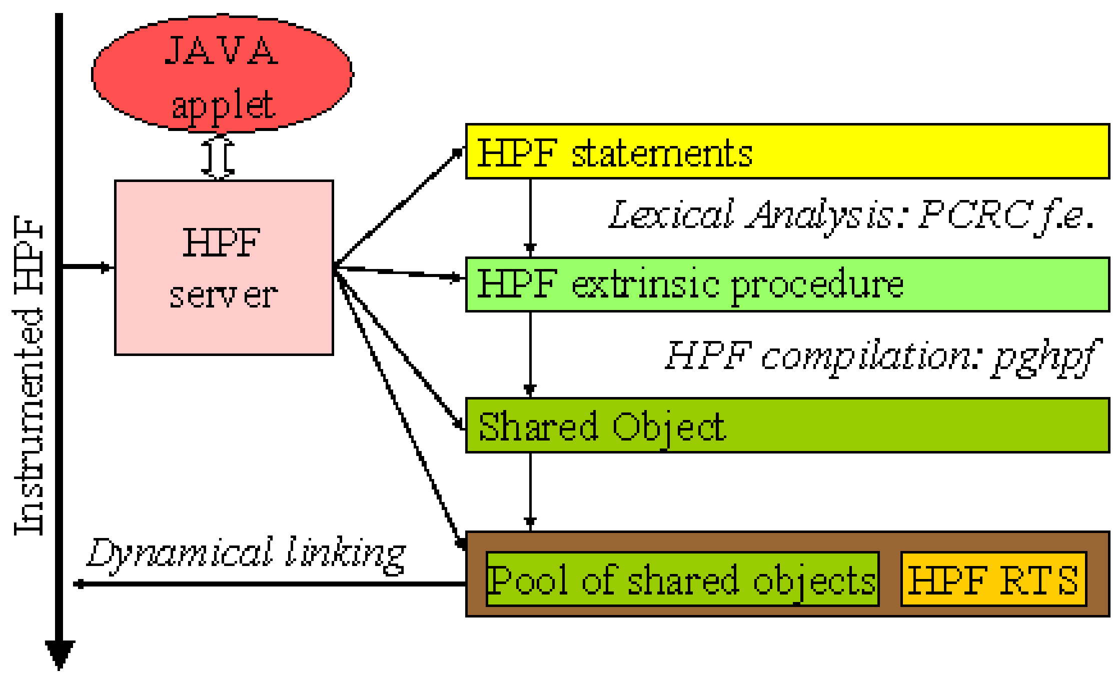


Figure 4. HPF interpreter

The interaction between the running application and user's commands is based on dynamical linking of UNIX shared objects with the application. This way any precompiled stand alone or library routine with a conforming interface can be called interactively at a breakpoint or at selected action points. In order to execute a new code entered in a form of a HPF source it must be first converted to a shared object. To this end we use the HPFfe to generate a subroutine from a single HPF statement or a block of statements, and then compile it using an HPF compiler, as shown in the Fig. 4. Since any “interpreted” code is in fact compiled, the efficiency of the resulting code is as good as that of the application itself. Nevertheless, time needed for creation of the shared object is prohibitively long to attempt to run the complete applications, statement after the statement, in the interpreted mode. On the other hand, the capability of data manipulations and visualization at any time during the execution of the application without recompiling and rerunning the whole application proves to be very time effective.

7. Runtime Visualizations

For visualizations we are using the Scientific Visualization System, SciVis, developed at NPAC. It is a portable system developed entirely in Java. With a very rich, and user extensible, set of data filters, and full support for a collaborative use it is a very powerful tool for a rapid data analysis.

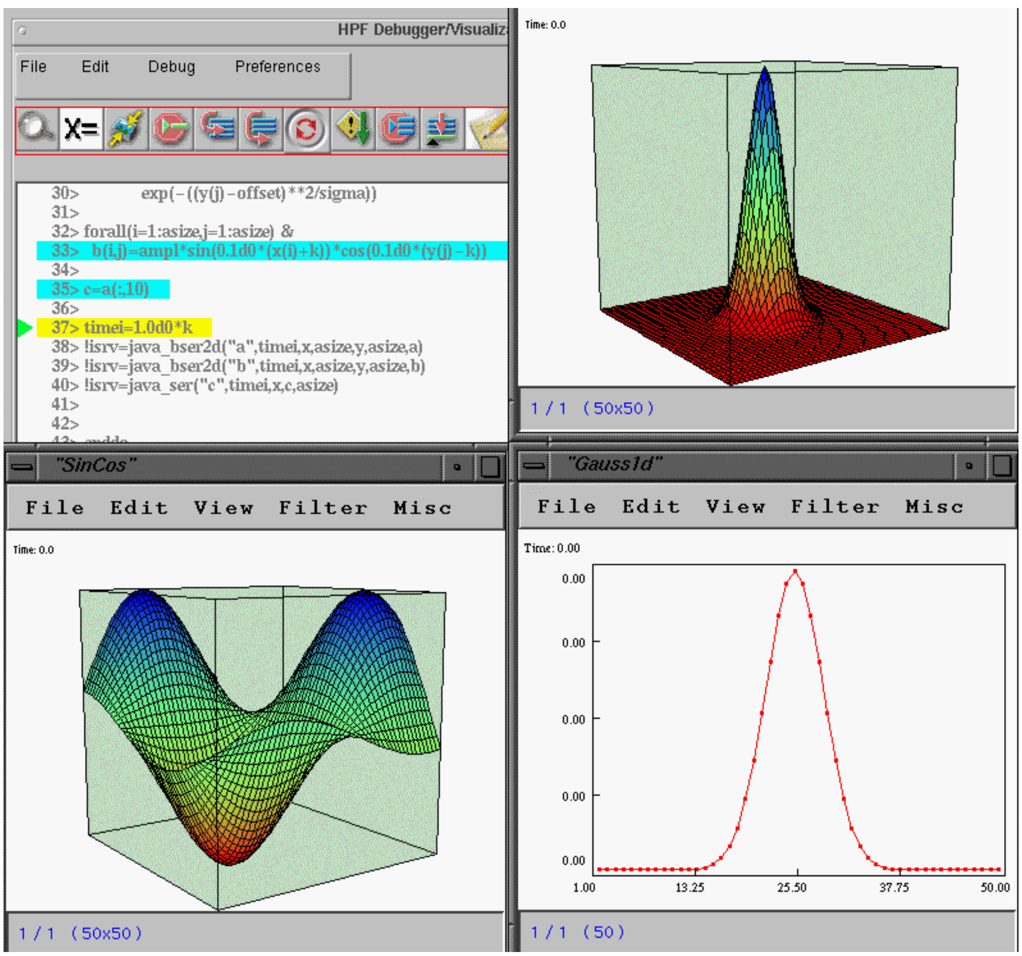


Figure 5. A screen dump of a DARP session. The upper right panel shows the front-end applet with a fragment of an HPF code. The action points at which SciVis proxy library is called are highlighted, and a triangle on the right points to the current line.

From the user point of view, it consists of a stand alone server, which is typically run on the user's workstation, and a client that supplies the data.

The architecture of the SciVis system makes it particularly attractive for integration with the DARP system. The SciVis client API allows us to design a proxy library in Java with a simple and very intuitive interface. The library, on the user behalf, causes automatic creation of a SciVis client routine that corresponds to the data type requested by the user. The client is then dynamically linked with the running application and executed at specified action point. This results in sending the data to the SciVis server which in turn display them on the user's workstation screen.

The same mechanism, with dedicated proxy libraries, can be used to integrate the DARP system with other software packages, such as computational libraries, data storage systems, or other visualization systems. Also, by using proxy libraries the DARP system may request or provide services from other tier-2 servers, or become a module in a data flow type computations[20].

8. Summary

By reusing commodity components and technologies we have built a powerful tool for data analysis and rapid prototyping to be used by an HPF application developer. The most important feature of the system is an interactive access to distributed data. This, in turn, makes it possible to select and send data to a visualization system at an arbitrary point of the application execution. Also, the data can be modified using either native HPF commands or dynamically linked computational modules.

Consistently with our HPcc strategy, the system implements a three-tier architecture: The Java front-end holds proxy objects produced by an HPF front-end operating on the back-end code. These proxy objects can be manipulated with an interpreted Web client interacting dynamically with compiled code through a typical tier-2 server (middleware). Although targeted for HPF back-end, the system's architecture is independent of the back-end language, and can be extended to support other high performance languages such as HPC++[18] or HPJava[19]. Finally, since we follow a distributed objects approach, the DARP system can be easily incorporated into a collaboratory environment such as Tango[4] or Habanero[21].

Bibliography

1. G. Fox, W. Furmanski, "HPcc as High Performance Commodity Computing", http://www.npac.syr.edu/users/gcf/hpdcbook/HPcc.html
2. K. Li, S. Klasky, "Scivis", http://kopernik.npac.syr.edu:8888/scivis/
3. W. Furmanski et al., "WebFlow", http://osprey7.npac.syr.edu:1998/iwt98/products/webflow/
4. M. Podgorny et al, "Tango, Collaboratory for the Web", http://trurl.npac.syr.edu/tango/
5. I. Foster, C. Kesselman, "The Nexus Multithreaded Runtime System", http://www.mcs.anl.gov/nexus/
6. I. Foster, C. Kesselman, "Globus", http://www.globus.org/
7. Guansong Zhang et al., "The frontEnd system", http://www.npac.syr.edu/users/zgs/frontEnd/

8. PCRC, http://www.npac.syr.edu/projects/pcrc/
9. F. Bodin, P. Beckman, D. Gannon, J. Gotwals, S. Narayana, S. Srinivas, B. Winnicka. “Sage++: An Object OrientedToolkit and Class Library for Building Fortran and C++ Restructuring Tools”, Proc. Oonski `94.
10. E. Akarsu, G. Fox, T. Haupt, “The DARP System”, http://www.npac.syr.edu/users/haupt/HPFI
11. High Performance Debugging Forum, http://www.ptools.org/hpdf/
12. Steven T. Hackstadt and Allen D. Malony “Distributed Array Query and Visualization for High Performance Fortran”, in Proc. of Euro-Par '96, Aug 1996, http://www.cs.uoregon.edu/hacks/research/daqv/.
13. Karsten Schwan, John Stasko, Greg Eisenhauer, Weiming Gu, Eileen Kraemer, Vernard Martin, and Jeff Vetter, “The Falcon Monitoring and Steering System”, 1996, http://www.cc.gatech.edu/systems/projects/FALCON/.
14. A. Tuchman, D. Jablonowski, G. Cybenko, “Runtime Visualization of Program Data”, in Proc. Visualization '91, (IEEE, 1991) 225-261
15. James Arthur Kohl and Philip M. Papadopoulos “The Design of CUMULVS: Philosophy and Implementation”, in PVM User's Group Meeting, Feb 1996. http://www.epm.ornl.gov/cs/cumulvs.html
16. Robert Hood., “The p2d2 Project: Building a Portable Distributed Debugger”, in Proc. of SPDT '96 SIGMETRICS Symposium on Parallel and Distr. Tools, May 1996
17. J. May & F. Berman, “Panorama: A portable, Extensible Parallel Debugger,” Proceedings of ACM/ONR Workshop on Parallel and Distributed Debugging, May 1993, pp. 96-106
18. D. Gannon et al., “HPC++”, http://www.extreme.indiana.edu/sage
19. B. Carpenter et al., Java-Ad, CRPC Tech Report
20. W. Furmanski, T. Haupt, “DARP System as a WebFlow module”, http://www.npac.syr.edu/users/haupt/HPFI/webflow/
21. “NCSA Habanero”, http://www.ncsa.uiuc.edu/SDG/Software/Habanero/index.html