Distributed Array Query and Visualization for High Performance Fortran

Steven T. Hackstadt and Allen D. Malony

Computer & Information Science Department
University of Oregon, Eugene, OR 97403 USA

Abstract. This paper describes the design and implementation of the Distributed Array Query and Visualization (DAQV) system for High Performance Fortran, a project sponsored by the Parallel Tools Consortium. DAQV's implementation utilizes the HPF language, compiler, and runtime system to address the general problem of providing high-level access to distributed data structures. DAQV supports a framework in which visualization and analysis clients connect to a distributed array server (*i.e.*, the HPF application with DAQV control) for program-level access to array values. Implementing key components of DAQV in HPF itself has led to a robust and portable solution.

1 Introduction

During the execution of a parallel program it is often necessary for the user to inspect the values of parallel data structures that are distributed across the processing nodes of a parallel machine. Similarly, to interact with parallel applications (*e.g.*, for computational steering), it may be necessary to access and analyze distributed data at runtime. Any tool that addresses a particular user need where distributed data access capabilities are required, whether it be debugging, performance analysis, or application interaction, has either to implement these capabilities itself or rely on some other (open) infrastructure to provide them. In the former case, tool implementations tend to become specific to low-level system details, the tool implements only what is necessary for its purposes, and abstractions of distributed data access tend not to be developed. The common anecdote from users that `PRINT` is the only good tool is indicative of these problems: tools for parallel systems are often complex, not integrated, and vary widely across machines. `PRINT` is the only thing that they can rely on.

The case of relying on an existing infrastructure for distributed data access also suffers because no such infrastructure currently exists. One might think that `PRINT` provides suitable functionality. But the problem is really the level of the tool's interface to the infrastructure; `PRINT` would require significant scaffolding to give it a high-level interface to an external tool. In a similar vein, many parallel debuggers are not always aware of how parallel data structures are distributed. Whereas they provide low-level mechanisms to get the data, the higher-level semantics about what the data is (*i.e.*, the type and characteristics of the distributed data object) are generally not utilized in the infrastructure and are not apparent in its external interface. However, it is exactly these semantics that are helpful in developing open frameworks and portable tools.

S. Hackstadt and A. Malony, *Distributed Array Query and Visualization for High Performance Fortran*, Proceeding of Euro-Par '96, Lyon, France, August 26-29, 1996

The DAQV project highlights the dilemma between the two alternatives described above and how it faces tool designers. As a project sponsored by the Parallel Tools (Ptools) Consortium, DAQV set out to address the user requirement of being able to access and visualize distributed array data coming from a parallel program. Initially, it placed more emphasis on the environment for array interaction (query specifications, array operations, visualization types, etc.) than the underlying infrastructure for interfacing with the running program [14]. However, based on feedback from the Ptools community [16], it quickly became clear that DAQV's mission was too broad and that it should be more focused. The most interesting feedback was from users who helped refine the focus of the project to create a robust and well-defined infrastructure rather than a multitude of user interface features. Users made it clear that the most important contribution that the project could make was to provide a technology that lets them "get the data" (*i.e.*, a high-level `PRINT` capability) and use other tools to "work with the data." The goal then became to develop interfaces for both (1) low-level extraction of data from the program, and (2) the higher-level request/delivery of data to an external client (*e.g.*, for visualization). Users also acknowledged the importance of HPF as the primary language target for the DAQV project.

The result is the DAQV tool as described in this paper. We present this background because we think that the Ptools project evaluation process was instrumental in forcing the design to seriously consider users' needs. This gave DAQV its focus, but how DAQV is integrated with the HPF language is what makes it unique. One final comment is that DAQV is more characteristic of a framework or meta-tool than of a tool. It provides the intended capabilities in a form that can be utilized in other tools that may want to do something with the data that DAQV provides; a visualization tool is but one example. The rest of this paper describes the DAQV framework for HPF. We proceed by discussing related work, design, functionality, and implementation. Examples of DAQV's application are then followed by a discussion of future work.

2 Related Work

We review the related research from four perspectives: language-level parallel tools, distributed data visualization, client-server tool models, and program interaction frameworks.

There has been a strong advocation in the last few years for parallel tools to be integrated in parallel language systems and to support the language semantics in their operation. For instance, the research work integrating Pablo with the Fortran D compiler [1] and Tau with the pC++ compiler [2] demonstrates the importance of providing high-level semantic context. This is also clearly seen in the Prism [18] environment, which is, perhaps, the best example of the ease of use that can come from an integrated tool system. DAQV clearly follows in this spirit and goes one step further by actually using the language system itself for part of its implementation. This was also a feature of Breezy, a forerunner of DAQV that provided high-level program interaction for the pC++ system [3].

One of the key programming abstractions found in parallel language systems is data parallelism. Because distribution of parallel data is an important factor in the performance behavior of a program, viewing data and performance information in relation

S. Hackstadt and A. Malony, *Distributed Array Query and Visualization for High Performance Fortran*, Proceeding of Euro-Par '96, Lyon, France, August 26-29, 1996

to the distribution aids the user in tuning endeavors. The GDDT tool [13] provides a static depiction of how parallel array data gets allocated on processors under different distribution methods and also supports an external interface by which runtime information can be collected. Kimelman *et al*. show how a variety of runtime information can be correlated to data distribution to better visualize the execution of HPF programs [11]. The IVD tool [10] uses a data distribution specification provided by the user to reconstruct a distributed data array that has been saved in partitioned form. In DAQV's case, array reconstruction is done implicitly (and portably) by array access functions implemented in HPF.

The increasing importance of portability and extendability in parallel tools has evoked designs following client/server models. The Panorama debugger [15] demonstrates how the use of interoperating modules can lead to increased functionality and generality in debugging systems. The p2d2 debugger [4] extends this concept considerably by proposing a full client/server debugging framework with comprehensive abstractions of operating system, language, library interfaces, and protocols for distributed object interaction. DAQV is clearly adopting the client/server approach for similar reasons, but in contrast to these two particular tools, the functionality is at a higher level, which actually affords the possibility of layering DAQV on top of systems like Panorama and p2d2.

The final perspective is one of dynamic program interaction. There has been a growing interest in runtime visualization of parallel programs and computational steering. Implementing such support raises interesting user and implementation issues. On the one hand, runtime visualization for a particular application domain might be able to utilize domain knowledge to implement a system meeting certain performance constraints. The pV3 system [6] is a good example. However, such specialized implementations may be limited when considering the general runtime visualization problem. DAQV, in many respects, is a direct descendant of the Vista research [17] since it embodies many of the same design goals: client/server operation, automated data access support, runtime operation, and structured interaction. The improvement DAQV offers is in language-level implementation to increase portability. We believe that this will also improve DAQV's ability for computational steering. Although Vista was extended for interactive steering in the VASE tool [5], the steering operations were still very much dependent on the target implementation. Supporting steering at the application language level is a more robust, general solution and is something we intend to investigate in DAQV.

3 Design

DAQV is somewhat different from many of the systems and tools discussed in the previous section. Its goal is to "expose" the distributed data structures of a HPF program to external tools — and to do so in a way that does not require external tools to have any knowledge about data decompositions, symbol tables, or the number of processors involved. The problem, then, is to provide access at a meaningful and portable level — a level at which the user is able to interpret program data and at which external tools need only know logical structures. DAQV's design has been motivated by a range of factors, including input from the Ptools user group, the need to support semantic-based access to distributed arrays, and a desire to maximize DAQV's portability. The key characteristics of this design are discussed below.

Global View of Data. High Performance Fortran (HPF) supports a global name space [12]; the programmer views distributed arrays at a global level, often performing operations on whole arrays and referring to array elements with respect to the entire array, not some local array section on a particular processor. Distribution directives allow programmers to use their knowledge about the application and advise the compiler on data distribution schemes. But this concern for data distribution does not affect how the programmer references the data. HPF syntax insulates the user from ever dealing with an array in an explicitly distributed manner. For this reason, DAQV supports a similar perspective of program data when interacting with the user (through external tools).

Client/Server Model. DAQV is a software infrastructure that enables runtime visualization and analysis of distributed arrays. It is not a stand-alone application or tool that performs these tasks itself; rather, it must interoperate with external tools. The goal in this design is to allow the continued use of existing visualization and analysis tools. The feedback from the Ptools user group during design discussions was clear: they did not need another fancy, self-contained visualization tool; they just wanted improved facilities for querying and extracting distributed arrays such that their existing tools could be easily used with the system. To this end, we logically view the entire HPF program (not individual processes or processors) as a *distributed array server* to which external *client tools* connect and then interact with the program and its data.

Portability. Another goal in the design of DAQV is to minimize the degree to which DAQV is machine- or compiler-dependent. By targeting HPF in the first place, machine portability is inherent to the extent that a given HPF compiler is. As for compiler portability, DAQV primarily accomplishes this in three ways: key components of DAQV are implemented in HPF; compiler and runtime systems are utilized; and compiler-dependent code is minimized and isolated.

4 Functionality

This section describes, in general terms, the two levels of functionality supported by DAQV. Called *push* and *pull*, these two models differ in the degree of interactivity available with the HPF program at runtime.

Push Model. The *push model* forms the basis of DAQV and constitutes the simplest and least intrusive way to access distributed arrays from an external tool, or *data client.* The push model is implemented by inserting simple DAQV subroutine calls into the HPF source code. These calls allow the programmer to (1) indicate the DAQV model (*i.e.*, push or pull) to be used in the program, (2) register distributed arrays with DAQV, (3) set parameters for communicating with a data client, (4) make DAQV connect with a data client, and (5) send the data values of a distributed array to the data client.

S. Hackstadt and A. Malony, *Distributed Array Query and Visualization for High Performance Fortran*, Proceeding of Euro-Par '96, Lyon, France, August 26-29, 1996

The functionality of the push model is the practical solution to the feedback from the Ptools user group. That is, with minimal additions to the HPF source code, users can extract the data values of distributed arrays and visualize them with other tools, never having to worry about array reconstruction. The push model can be used to spot check the state of an array or to create animations of data values over the iterations of a loop. Multiple arrays can be pushed out of the program to multiple data clients.

Pull Model. The push model is adequate if the programmer knows at compile-time exactly which arrays they want to visualize and when they want to view them. However, to support a more exploratory and flexible approach to array visualization, DAQV supports the *pull model* which enables controlling program execution and selecting arrays for visualization through an external interface.

The pull model allows the user to repeatedly run an HPF code for a period of time and then extract data values from the distributed arrays of interest. Two types of clients are used in the pull model. As in the push model, data clients process data values from arrays. In fact, any data client that works in the push model also works in the pull model because the pull model is layered on top of the push model. In addition, though, the pull model requires a single *control client* to direct program execution and configure and initiate array transfers to data clients.

Thus, the primary conceptual difference between the push and pull models is where the decision to extract an array originates. The names "push" and "pull" are meant to reflect this difference in perspective. In the push model, the HPF program "pushes" data out, while in the pull model, an external client reaches in and "pulls" data out. However, the implementation of these very different conceptual models is built upon a common infrastructure supported by DAQV.

5 Implementation

Our goal with DAQV is to facilitate simple and useful conceptual models for distributed array collection and extraction, and to implement those models in a high-level, portable manner. This section briefly explains the two key components of the DAQV implementation: procedural interface and client/server interface.

Procedural Interface. DAQV's core requirement is to support interaction with HPF programs. In part, this interaction is similar to what might be provided by a HPF debugger. However, a HPF debugger may not provide a high-level interface for distributed array query, relying instead on low-level support for gathering array data on a node-by-node basis. Such system dependencies defeat DAQV's goal of portability, making it too reliant on the target compiler or machine. Instead, key components of DAQV are implemented as HPF subroutines, allowing array access and other functions to utilize the HPF compiler and runtime system implicitly and automatically. In this respect, we view DAQV as a HPF language-level tool design and implementation.

Thus, the high-level operation of DAQV demands a different method for interacting with the HPF program than what a HPF debugger can provide. Our solution is to implement DAQV as a library that is linked with a HPF object file, creating an executable that supports a procedural interface between the original HPF program and the

DAQV distributed array server. The routines in this interface handle initialization, registration of arrays, configuration of data clients, and data extraction.

Client/Server Interface. Conceptually, the client/server interface supported by DAQV exists between the entire HPF program (*i.e.*, all SPMD processes representing the HPF program) and external data and control clients. DAQV uses a bidirectional, string-based event protocol between the HPF server and its clients. In the push model, data clients respond to array data transmissions with a confirmation that signals DAQV to let the HPF code continue executing. The pull model uses a more complete event protocol that allows the control client to direct program execution, configure client communications, and extract array values.

One HPF process is designated as a "manager" by DAQV. The manager is responsible for handling all client communications, and the other HPF "worker" processes cooperate with the manager to effect DAQV operations. This distinction is only evident to DAQV; it has no effect on the regular execution of the HPF program. So, events sent to the server are actually received only by the manager process and must be "shared" with the other HPF processes by special mechanisms implemented in HPF.

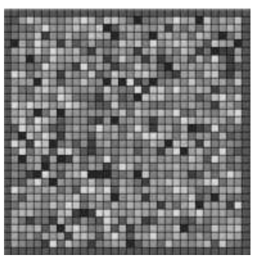
As an open framework that is intended to work with a variety of different data clients, efforts have been made to allow existing visualization and analysis tools to be easily ported for use with DAQV. First, the current implementation uses only sockets for communication. Second, a client must be able to parse data sent by DAQV. And third, a client must be able to respond over the socket with a simple string-based confirmation event after receiving data.

Control clients, on the other hand, require somewhat more sophistication. They have the same requirements as the push model, but in addition, they must handle control events. A C-library has been developed for supporting event processing, and support for other languages (*e.g.*, Tcl/Tk) will be forthcoming. Furthermore, a control client usually interacts with the user. The DAQV pull model and event specification establish a framework for other tool developers by providing a simple protocol that can be incorporated into other control clients to gain high-level access to distributed data.

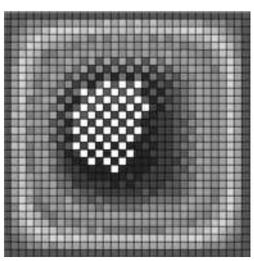
Language, Compiler and Runtime System Requirements. With a primary goal being portability, DAQV does not require any modifications to the HPF language implementation, however certain assumptions have been made in the DAQV reference implementation. Currently, DAQV assumes that HPF data parallelism is achieved with a SPMD execution model. (In many respects this is a more difficult problem when compared to, say, a multithreaded implementation.) An HPF program must be able to invoke C routines, and vice-versa. The process of array registration requires knowledge about the transformations applied to function and subroutine arguments by a given compiler. However, the information required is minimal and so far has not required vendors to divulge proprietary information. DAQV can optionally take advantage of compiler-specific tracing/profiling support, though this is not required by the reference implementation. Finally, in two instances, DAQV *requires* HPF distribution directives to be carried out by the compiler. That is, DAQV will not work if the directives are ignored. With these minimal requirements and assumptions, we feel DAQV will be able to achieve a high degree of portability across both machines and compilers.

6 Application: Laplace Heat Equation

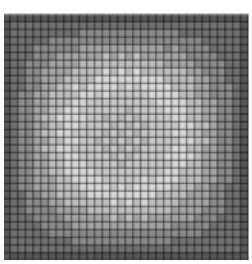
To illustrate what the current implementation of DAQV is capable of, we consider a HPF program that implements an iterative finite difference method for solving the Laplace Heat Equation [7]. The code continues until a steady state is reached to within some tolerance. DAQV is used in the push mode to visualize the heat flow through the two-dimensional surface at each iteration of the main loop. The sequence of images in Figure 1 was generated by a data client called Dandy. Implemented in Tcl/Tk, Dandy was easily ported from the pC++ Tau Tools [2] for use with DAQV. Under this scheme, when the program reaches the DAQV procedure that establishes communication with a data client, the Dandy client connects by a socket to the HPF program. At this point, execution resumes and the animation of the data values begins. At each loop iteration, the new array values are sent to Dandy and displayed. The Dandy interface allows the user to pause/resume the animation and redraw the display. Dandy automatically determines the range of the data values and maps them onto a fixed colormap. However, if the viewer wishes to fix the color range across several iterations, the automatic scaling feature can be disabled, allowing minimum and maximum data values to be set. This



(a)



(b)



(c)

Fig. 1. Convergence of the finite method for solving the Laplace Heat Equation can be seen by visualizing the two-dimensional array representing the heated surface.

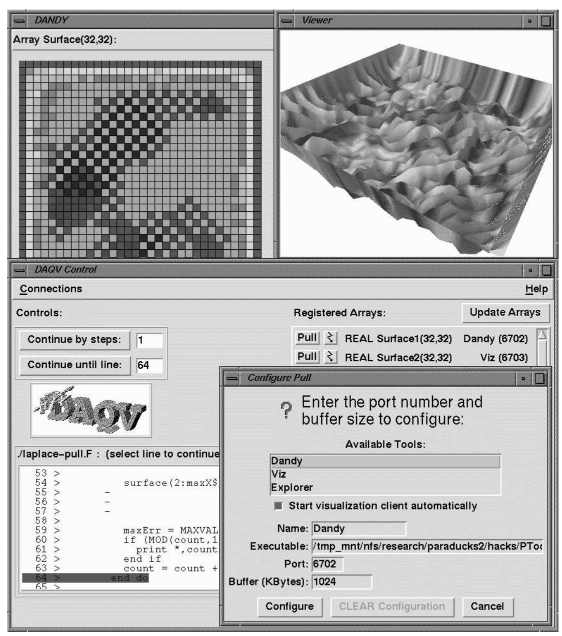


Fig. 2. A DAQV session with a control client. A simple two-dimensional data client and a three-dimensional data client both represent the 2D array from the Laplace Heat Equation code.

S. Hackstadt and A. Malony, *Distributed Array Query and Visualization for High Performance Fortran*, Proceeding of Euro-Par '96, Lyon, France, August 26-29, 1996

feature can also be used to identify outliers or to identify values beyond a certain threshold. For example, Figure 1(b) shows values above the specified range as white. As the algorithm nears convergence (Figure 1(c)), these values move into the data range.

The same application can also be used with DAQV's pull model. Figure 2 shows several windows from a DAQV session. A prototype control client interface (bottom, partially obscured) supports several DAQV features, including a source code browser, program controls, a list of HPF arrays that have been registered with DAQV. In this case, Surface, a 32x32 real-valued array, has been registered twice so that it may be viewed with two different data clients. Other controls allow the user to select a data client for an array and to send the values of an array to the appropriate data client. The first registered array, Surface1, has been mapped to the Dandy client in the upper left part of the screen, while Surface2 is being sent to a more complex, three-dimensional display created in the Viz visualization environment builder [8].

7 Future Work

In its current form, DAQV should be regarded as a reference implementation that demonstrates functionality, specifies components and their system requirements, and defines APIs and transport protocols. In fact, being the primary derivative of a Ptools project, the reference implementation serves as a prototype for vendors to evaluate and potentially adopt. If the Ptools reference platform were to be adopted by a HPF compiler vendor, several opportunities for improvement exist. In particular, DAQV lends itself nicely to compiler/preprocessor support for instrumentation.

Beyond the Ptools portion of the DAQV project, there are several interesting research directions for this work. One of the most important short-term goals is to evaluate the benefit of DAQV in a significant scientific application. At the University of Oregon, we are developing an HPF-version of a seismic tomography application for modeling mid-ocean ridges. The DAQV framework is being used to create visualizations that will allow researchers to better understand sea floor geology.

The DAQV framework is also being extended to allow distributed data to be altered as well as accessed. As a result, DAQV can be applied to application scenarios where changing runtime array data or program control variables is beneficial (*e.g.*, for computational steering in the seismic tomography application). In the near future, DAQV will be tested on other parallel machine platforms supported by PGI's HPF compiler. In addition, we hope to port DAQV to other HPF compilers. We also believe that the DAQV model (and parts of its implementation perhaps) can be applied in other parallel language systems where distributed data is involved, and we intend to retarget the DAQV reference code to the HPC++ [9] system. More generally, we see no major intellectual difficulties with porting DAQV to SPMD environments where some runtime means (language or library) for accessing distributed data exists.

Acknowledgments

Thanks to the Parallel Tools Consortium, The Portland Group Inc., David Presberg, Craig Lee, Kurt Windisch, Harold Hersey, Bernd Mohr, and Chris Harrop for their contributions to the design and implementation of DAQV. The full technical report describing DAQV can be found at http://www.cs.uoregon.edu/~hacks/research/daqv/.

References

[1] V. Adve, *et al.*, *An Integrated Compilation and Performance Analysis Environment for Data-Parallel Programs*, Supercomputing '95, Dec. 1995.

[2] D. Brown, S. Hackstadt, A. Malony, & B. Mohr, *Program Analysis Environments for Parallel Language Systems: The TAU Environment*, Proc. of the Workshop on Environments & Tools For Parallel Scientific Computing, Townsend, TN, May 1994, pp. 162-171.

[3] D. Brown, A. Malony, B. Mohr, *Language Based Parallel Program Interaction: The Breezy Approach*, Proc. of Intl. High Performance Computing Conference (HiPC'95), New Delhi, India, Dec. 1995.

[4] D. Cheng & R. Hood, *A Portable Debugger for Parallel and Distributed Programs*, Proc. of IEEE Supercomputing '94, Washington, D.C., Nov. 1994.

[5] R. Haber, *et al.*, *A Distributed Environment for Runtime Visualization and Application Steering in Computational Mechanics*, Univ. of Illinois at Urbana-Champaigne, CSRD Technical Report 1235, Jun. 1992.

[6] R. Haimes, *pV3: A Distributed System for Large-Scale Unsteady CFD Visualization*, American Institute of Aeronautics & Astronautics Paper 94-0321, Reno NV, Jan. 1994.

[7] P. B. Hansen, *Studies in Computational Science: Parallel Programming Paradigms*. Prentice Hall, Inc., Englewood Cliffs, NJ, 1995.

[8] H. Hersey, S. Hackstadt, L. Hansen, & A. Malony, *Viz: A Visualization Programming System*, Univ. of Oregon, Dept of Computer & Information Science, Technical Report CIS-TR-96-05, Apr. 1996.

[9] HPC++ Working Group, *HPC++ Whitepapers and Draft Working Documents*, Supercomputing '95 Workshop, URL:http://www.extreme.indiana.edu/hpc++/hpc++wp/, Feb. 1996.

[10] A. Karp & M. Hao, *Ad Hoc Visualization of Distributed Arrays*, Hewlett-Packard Technical Report HPL-93-72, Sep. 1993.

[11] D. Kimelman, P. Mittal, E. Schonberg, P. Sweeney, K. Wang, D. Zernik, *Visualizing the Execution of High Performance Fortran (HPF) Programs*, Proc. of the 9th Intl. Parallel Processing Symposium (IPPS), IEEE Computer Society Press, Apr. 1995, pp. 750-757.

[12] C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr., & M. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, MA, 1994.

[13] R. Koppler, S. Grabner, & J. Volkert, *Visualization of Distributed Data Structures for HPF-like Languages*, to appear in Scientific Programming, spec. issue on implementations of High-Performance-Fortran, 1995.

[14] A. Malony, J. May, A. Karp, D. Presberg, & V. Schuster, *Distributed Array Query and Visualization*, Parallel Tools Consortium Working Document, Jan. 1995.

[15] J. May & F. Berman, Panorama: *A Portable, Extensible Parallel Debugger*, Proceedings of the ACM/ONR Workshop on Parallel & Distributed Debugging, May 1993, pp. 96-106.

[16] Parallel Tools (Ptools) Consortium DAQV Working Group, *Ptools Meeting Summary: Distributed Array Query and Visualization (DAQV) Project*, available at URL:http://www.cs.uoregon.edu/~hacks/research/daqv/, May 1995.

[17] A. Tuchman, D. Jablonowski, G. Cybenko, *Runtime Visualization of Program Data*, Proceedings of IEEE Visualization '91, Oct. 1991, pp. 255-261.

[18] Thinking Machines Corp., *Prism*, available at URL:http://www.think.com/tmhtml/ProdServ/Products/prism.html, Feb. 1996.