Request Sequencing: Optimizing Communication for the Grid

Dorian C. Arnold[1], Dieter Bachmann[2], and Jack Dongarra[1,3]

[1] Department of Computer Science, University of Tennessee, Knoxville, TN 37996
{darnold, dongarra}@cs.utk.edu
[2] Computer Graphics and Vision, Graz University of Technology, Inffeldg. 16/E/2, A-8010 Graz Austria
bachmann@icg.tu-graz.ac.at
[3] Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

Abstract. As we research to make the use of Computational Grids seamless, the allocation of resources in these dynamic environments is proving to be very unwieldy. In this paper, we introduce, describe and evaluate a technique we call request sequencing. Request sequencing groups together requests for Grid services to exploit some common characteristics of these requests and minimize network traffic. The purpose of this work is to develop and validate this approach. We show how request sequencing can affect scheduling policies and enable more expedient resource allocation methods. We also discuss some of the reasons for our design, offer the initial results and discuss issues that remain outstanding for future research.

1 Introduction

As the number of applications that can benefit from Grid Computing [1] infrastructure increases, there are many critical issues that need to be efficiently resolved. Determining which resources provide the quickest, least expensive route to computational solutions is an active area in this research community. Examples of efforts focused on this area are AppLeS[2] and the Network Weather Service[3].

The vBNS[4] and Myrinet[5] represent two technologies that connect computational resources at high speeds in settings from local to global area networks. However, with the speed of processors increasing at a rate much greater than the speed of networking infrastructure, data transfer continues to bear large overhead for many applications of high-performance computing. Yet, straightforward ways for increasing application performance by optimizing communication often go overlooked. Our research on call sequencing for Grid middleware aims to take a significant step in this direction.

1.1 Goals

Computer applications generally exhibit two common characteristics: large input data sets and data dependency amongst computational cycles. The goal

of this effort is to employ simple, yet highly effective, strategies for exploiting these characteristics. We believe that not enough attention has been given to examining ways to effectively distribute application data amongst the different components of a Grid. We qualify this last statement by saying that data partitioning has been well researched for parallel programming, but in cases where computational modules execute concurrently with no data exchange, often the same data is unnecessarily transported multiple times between the same components.

1.2 Positioning Our Work

This paper explores the design, implementation and initial results of what we call request sequencing. This term encompasses both an interface to group a series of requests and a scheduling technique, viz. one that uses data persistence and a Direct Acyclic Graph (DAG) representation of computational modules. Our motivation was to allow users to take advantage of data redundancies within a sequence of requests and optimize data communications. We developed and tested our ideas using the NetSolve system described in Sect. 2. Below, we offer the relationship of this research to other research that has been or is being done.

Our scheduling work is reminiscent of techniques utilized in schedulers that execute "batches" of processes on parallel machines. We create task graphs or DAGs that represent execution dependencies and schedule them for execution [6]. J. Dennis researched data flow scheduling techniques for Supercomputers[7]; our work presents a similar idea for Grid environments.

Ninf [8] is a functional metacomputing environment that shares many similarities with NetSolve. The project implemented a strategy to increase parallelism[9]. Similar to this work, they group together requests and execute the modules simultaneously, when possible. Their main focus is on parallel module execution, and it is not stated whether redundant messages are sent or not. Our focus is on minimizing network traffic; our design ensures that no unnecessary data transfer takes place.

Condor [10] is a high-throughput computing system that manages very large collections of distributively owned workstations. The Directed Acyclic Graph Manager[11] (DAGMan) is a meta-scheduler for Condor jobs. Users can submit batch jobs to the Condor system and use DAGMan to pre-define execution order. Once again, however, the main focus is on parallel execution and not data transfer. As an extra burden, the data dependency analysis is left to the user.

The contribution put forth by this paper is a thorough understanding of an approach to optimize data transfer in Grid settings and the empirical data to justify using this approach. We also offer discussion of scheduling in this environment. Section 2 of this paper presents details about NetSolve, which is our deployment environment. Section 3 describes the design and implementation of the sequencing interface, the server data persistence and execution scheduling. Section 4 contains the experimental test cases and the results that validate this strategy. Finally, Sect. 5 summarizes the work and discusses future research goals.

2 An Overview of NetSolve

The NetSolve project is being developed at the University of Tennessee and the Oak Ridge National Laboratory. It provides remote access to computational resources, both hardware and software. Built upon standard Internet protocols, like TCP/IP sockets, it is available for all popular variants of the UNIX operating system, and parts of the system are available for the Microsoft Windows '95, '98 and NT platforms.

Figure 1 shows the infrastructure of the NetSolve system and its relation to the applications that use it. NetSolve and systems like it are often referred to as Grid Middleware; this figure helps to make the reason for this terminology clearer. The shaded parts of the figure represent the NetSolve system. It can be seen that NetSolve acts as a glue layer that brings the application or user together with the hardware and/or software it needs to complete useful tasks.

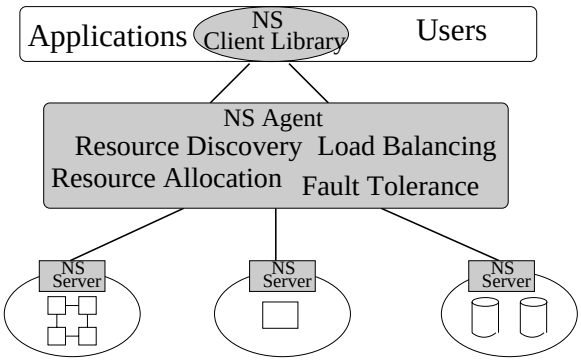

Fig. 1. Architectural Overview of the NetSolve System

At the top tier, the NetSolve client library is linked in with the user's application. The application then makes calls to NetSolve's application programming interface (API) for specific services. Through the API, NetSolve client-users gain access to aggregate resources without the users needing to know anything about computer networking or distributed computing. In fact, the user does not even have to know remote resources are involved.

The NetSolve agent maintains a database of NetSolve servers along with their capabilities (hardware performance and allocated software) and dynamic usage statistics. It uses this information to allocate server resources for client requests. The agent finds servers that will service requests the quickest, balances the load amongst its servers and keeps track of failed ones.

The NetSolve server is a daemon process that awaits client requests. The server can run on single workstations, clusters of workstations, symmetric multiprocessors or machines with massively parallel processors. A key component of

the NetSolve server is a source code generator which parses a NetSolve problem description file (PDF). This PDF contains information that allows the NetSolve system to create new modules and incorporate new functionalities. In essence, the PDF defines a wrapper that NetSolve uses to call functions being incorporated.

For more detailed information on the NetSolve system and its usage, refer to [12].

3 Sequencing Design and Implementation

As stated in Sect. 1.1, our aim in request sequencing is to decrease network traffic and overall request response time. Our design needs to ensure that i) no unnecessary data is transmitted and ii) all necessary data is transferred. We also need to cut execution time by executing modules simultaneously when possible. We do this by performing a detailed analysis of the input and output parameters of every request in the sequence and producing a DAG that represents the tasks and their execution dependencies. This DAG is then sent to a server in the system where it is scheduled for execution.

3.1 The DAG Model

Kwok et al. [6] offers a very good description of the DAG:

> The DAG is a generic model of a parallel program consisting of a set of processes (nodes) among which there are dependencies. A node in the DAG represents a task which in turn is a set of instructions that must be executed sequentially without preemption on the same processor. A node has one or more inputs. When all inputs are available, the node is triggered to execute. The graph also has directed edges representing a partial order among the tasks. The partial order introduces a precedence-constrained directed acyclic graph and implies that if $n_i \rightarrow n_j$, then n_j is a child which cannot start until its parent n_i finishes and sends its data to n_j.

3.2 Data Analysis and the DAG

In order to build the DAG or task graph, we need to analyze every input and output in the sequence of requests. We evaluate two parameters as the same if they share the same reference. We use the size fields and reference pointer of the input parameters to calculate when inputs overlap in the memory space. NetSolve supports many object[1] types, including matrices, vectors and scalars; a decision was made to only check matrices and vectors for reoccurences. This was based on our belief that these are the only objects that tend to be large enough for the overhead of the analysis to pay dividends. We do, however, check all

[1] We use the term object to refer to a composition of native data types, as in a matrix object of native integers

inputs for execution dependencies (e.g. a scalar integer dependency may affect the execution graph.) This analysis yields a DAG. The graph is acyclic because looping control structures are not allowed within the sequence, and therefore, a node can never be its own descendant.

3.3 The Interface

In addition to the original function used for request submittal, two functions are implemented; their purpose is to mark the beginning and end of a sequence of requests. `begin_sequence()` takes no arguments and returns nothing; it notifies the system to begin the data analysis. `end_sequence()` marks the end of the sequence; at this point, the sequence of collected requests is sent to a server(s) to be scheduled for execution. As an enhancement, this function also takes a variable number of arguments describing which output parameters NOT to return. This means that if the intermediate results are not necessary for any local computations, they need not be returned. This is a part of the API as it is the user who should determine which results are mandatory and which are useless. Figure 2 illustrates what a sequencing call might look like. Two points to note in this example: i)for all requests, only the last parameter is an output, and ii)the user is instructing the system not to return the intermediate results of `command1` and `command2`.

```
...
begin_sequence();
submit_request("command1", A, B, C);
submit_request("command2", A, C, D);
submit_request("command3", D, E, F);
end_sequence(C, D);
...
```

Fig. 2. Sample C Code Using Request Sequencing Constructs

For the system to be well-behaved, we must impose certain software restrictions upon the user. Our first restriction is that no control structure that may change the execution path is allowed within a sequence. We impose this restriction because the conditional clause of this control structure may be dependent upon the result of a prior request in the sequence, and since the requests are not scheduled for execution until the end of the sequence, the results will probably not be what the programmer expects.

The other restriction is that statements that would change the value of any input parameter of any component of the sequence are forbidden within the sequence (with the exception of calls to the API itself that the system can track.) This is because during the data analysis, only references to the data are stored. So if changed, the data transferred at the end of the sequence will not

be the same as the data that was present when the request was originally made. We contemplated saving the entire data, rather than just the references, but this directly conflicts with one of our premises – that the data sets are large; multiple copies of these data are not desirable.

3.4 Execution Scheduling at the Server

Once the entire DAG is constructed, it is transferred to a NetSolve computational server. [6] offers a taxonomy of different graph scheduling algorithms in multi-processor environments. These algorithms take into account both node-computation and inter-node communication costs. In this first version of request sequencing, the NetSolve agent uses a larger granularity and decides which server should execute the entire sequence. We execute a node if all its inputs are available and there are no conflicts with its output parameters. The reason for this is that currently the only mode of execution we support is on a single NetSolve server – though, that server may be a symmetric multi-processor (SMP). We discuss our plans for expanding this model in Sect. 5.

For data partitioning, we transfer the union of the input parameter sets to the selected server host. This makes input for all nodes, except those which are intermediate output from prior nodes, available for the execution of the sequence. When we move to a multi-server execution mode for the sequence, we must enhance our data staging technique, and this is also discussed in Sect. 5.

Our execution scheduling algorithm, Fig. 3 is similar to that used in the computational steering system, SCIRun[13]. In essence, we execute all nodes with no dependencies, updating the dependency list as nodes complete, and then check for further nodes to execute. We keep doing this until all nodes have executed:

```
while(problems left to execute){
  execute all problems that have no dependencies;
  wait for at least one problem to finish;
  update dependencies;
}
```

Fig. 3. Pseudo Code for Scheduling Algorithm

3.5 Discussion

Figures 4 and 5 show the reduced network activity between client and server during execution of the sequence in Fig. 2. In the first case, input `A` is sent to the server twice, and output `C` and `D` are unnecessarily sent back to the client as intermediate output and then to the server once again as input. In the latter case,

these unnecessary transfers are removed. (These diagrams show three potentially different servers, but our current implementation sees this as three instances of the same server.) Our hypothesis is that this reduction in data traffic will yield enough performance improvements to make sequencing worthwhile.

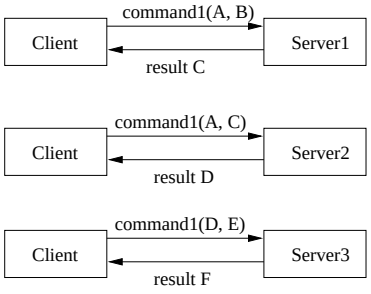

Fig. 4. Client-Server Data Flow Without Request Sequencing

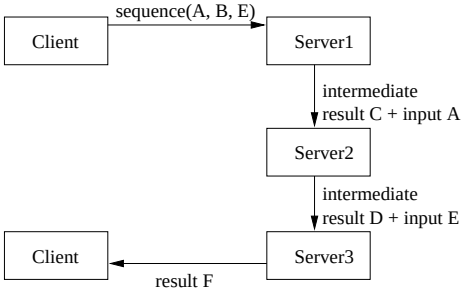

Fig. 5. Client-Server Data Flow With Request Sequencing

4 Applications and Initial Results

In this section, we discuss the applications that we used to test our request sequencing infrastructure. They are from the remote sensing/image processing domain, and as it turns out, it was the nature of some of these applications that led us to investigate request sequencing. The size of images that are analyzed can become very large and easily extend into the gigabyte range. It is also common in many image processing applications to execute a series of operations on an image, usually one transformation after another.

All the experiments were executed from NetSolve clients connected to a switched 10/100Mbit ethernet and crossing a 155Mbit ATM switch that is directly connected to the NetSolve servers. The NetSolve server was a SGI Power Challenge with eight R10000 processors. Graphed results are the averages of four independent sets of runs.

For the experiments, we varied the network bandwidth by using the NistNet [14] interface on a Linux router. A network performance testing tool, TTCP [15], which is able to generate TCP (and UDP) traffic on IP based networks, was used to obtain a correction curve for the values set by NistNet.

4.1 Linear Sequence: Principle Component Analysis

Multispectral or multidimensional remote sensing data can be represented by constructing a vector space using one axis per dimension. By calculating the covariance matrix [16], the axes are transformed into an uncorrelated system. This transformation is called Principle Component Analysis (PCA) [17].

In remote sensing, the PCA is used to reduce the number of channels for input images by moving the information towards the first bands. The application used for testing a linear sequence opens a 10MB image, performs a PCA and stores the transformed result. The structure is shown in Fig. 6.

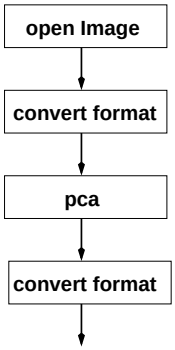

Fig. 6. Principal Component Analysis

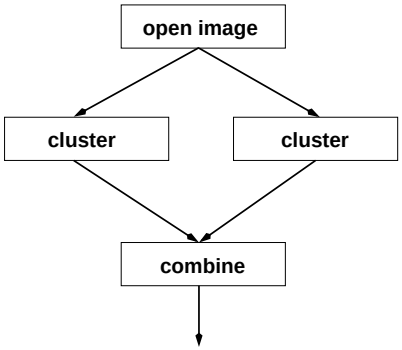

Fig. 7. Multimodal Image Clustering

Fig. 8 shows how the total response time (from request initiation to availability of results) varies with the bandwidth. It confirms our beliefs: with sequencing in place, there is a significant reduction in execution time of the PCA application. Similar results can be expected for applications that exhibit similar levels of parameter sharing, and in fact, our examples are not contrived, but represent realistic applications that scientists have used to support their research.

4.2 Parallel Sequence: Clustering

To handle multisource/multimodal satellite images or to improve clustering accuracy, several classification steps are performed, and their results are combined by a fusion module. Such a module can consist either of a simple pixel selection approach based on severity ratings or of a knowledge based combination module [17]. For our tests a pixel based approach has been chosen using an image size on the order of 1MB. This process is illustrated in Fig. 7.

Fig. 9 graphs the variation of response time with bandwidth for this parallel sequence. The shape is similar to that of the PCA application. Again, request sequencing yields decreases in execution time.

These preliminary results provide much encouragement for our continued investigation of request sequencing. Discussed more in the next section, parallel sequencing has the potential to be a very useful methodology in the area of Grid and metacomputing.

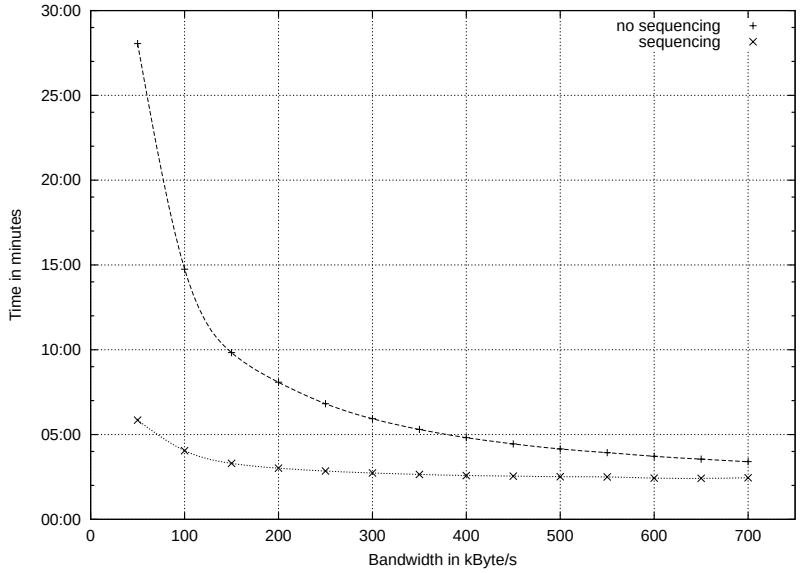

Fig. 8. PCA Sequence Executed on an SGI workstation

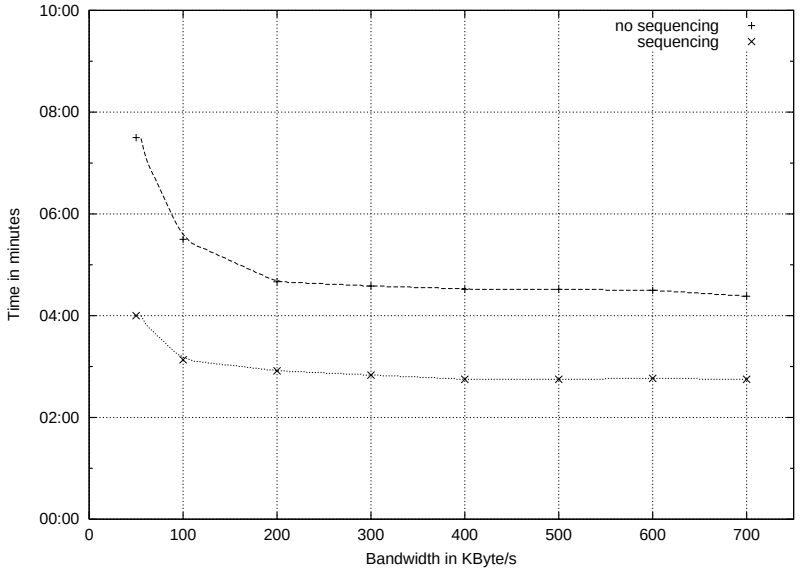

Fig. 9. Clustering Sequence Executed on Two Processors of an SGI workstation.

5 Conclusion and Future Work

In this paper, we have presented a general technique to reduce network traffic when executing several requests to a Grid Computing system. Our approach is to build a DAG whose structure exhibits the dependencies amongst the requests. This DAG is then scheduled for execution on a server. Our initial experiments with request sequencing are prominsing and show that we are able to significantly reduce execution time of our client application. As a final thought, we offer Fig. 10 which shows that even at its worst, request sequencing improves execution time by a factor of about 1.5. Though we have not proven this result, it is our belief that in most cases, sequencing should at least perform as well as no sequencing at all. (We plan to show this in future work.)

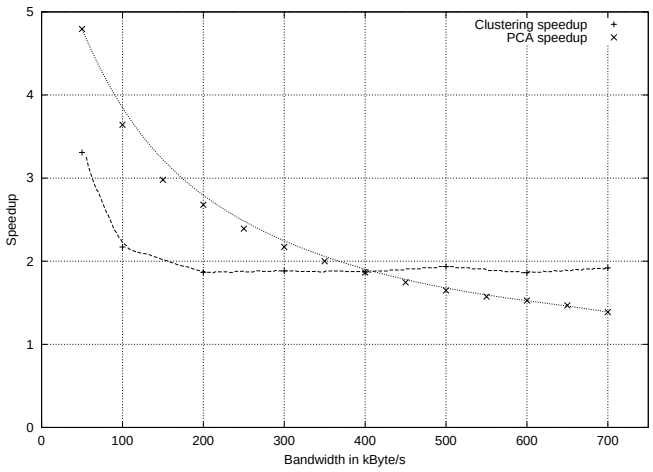

Fig. 10. Reduction in Execution Time due to Request Sequencing

Section 3.5 mentions that the sequences are currently restricted to execution on a single server. The next logical progression is to allow different components of the sequence to execute on different hosts. The implications are that no single server needs to possess all the software capabilities for the sequence. This also means that the modules will truly be able to execute in parallel even when no parallel machine is present. Scheduling techniques as discussed by [6] will be evaluated, and we will incorporate factors like computational and communication costs to better approximate optimal solutions. It makes little sense to execute the components of a sequence on various servers without taking data locality into account. Tools like the Internet Backplane Protocol[18] and Global Access to Secondary Storage[19] will be leveraged to provide all servers with convenient access to the necessary data.

References

1. I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.
2. F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proc. of Supercomputing'96, Pittsburgh, PA*, November 1996.
3. R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. Technical Report TR-CS96-494, U.C. San Diego, October 1996.
4. J. Jamison and R. Wilder. vBNS: The Internet Fast Lane for Research and Education. *IEEE Communications Magazine*, 35(1):60–63, January 1997.
5. N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A Gigabit per Second Local Area Network. *IEEE-Micro*, 15:29–36, February 1995.
6. Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, December 1999.
7. J. Dennis. Data Flow Supercomputers. *IEEE Computer*, 13(11):48–56, November 1980.
8. S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe, CA*, 1996.
9. H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi. Utilizing the MetaServer Architecture in the Ninf Global Computing System. In *Proc. of High Performance Computing and Networking, Amsterdam, The Netherlands*, 1998.
10. M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proc. of the 8th International Conference of Distributed Computing Systems, San Jose, CA*, pages 104–111, June 1988.
11. Madison Condor Team, University of Wisconsin. Condor Version 6.1.8 Manual, 1999.
12. H. Casanova and J. Dongarra. NetSolve's Network Enabled Server: Examples and Applications. *IEEE Computational Science & Engineering*, 5(3):57–67, September 1998.
13. S. Parker, D Weinstein, and C. Johnson. *Modern Software Tools in Scientific Computing*, pages 1–40. Birkhauser Press, 1997.
14. S. Parker and C. Schmechel. RFC 2398: Some testing tools for TCP implementors, August 1998.
15. Chesapeake Network Solutions. Network Performance Testing with TTCP. *Chesapeake Online: The Network Monitor*, 3(1), 1997.
16. R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, 1993.
17. J. A. Richards. *Remote Sensing Digital Image Analysis*. Springer, 2nd edition, 1993.
18. J. Plank, M. Beck, Elwasif W., , T. Moore, Swany M., and R. Wolski. IBP – The Internet Backplane Protocol: Storage in the Network. In *NetStore '99: Network Storage Symposium, Seatle, WA*, October 1999.
19. J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Sixth Workshop on I/O in Parallel and Distributed Systems, Atlanta, GA*, May 1999.