NAS Grid Benchmarks: A Tool for Grid Space Exploration

Michael Frumkin
NASA Advanced Supercomputing Division
M/S T27A-2, NASA Ames Research Center
Moffett Field, CA 94035-1000
frumkin@nas.nasa.gov

Rob F. Van der Wijngaart
Computer Sciences Corporation
M/S T27A-1, NASA Ames Research Center
Moffett Field, CA 94035-1000
wijngaar@nas.nasa.gov

Abstract

We present a benchmark suite for computational Grids. It is based on the NAS Parallel Benchmarks (NPB) and is called NAS Grid Benchmark (NGB) in this paper. We present NGB as a data flow graph encapsulating an instance of an NPB code in each graph node, which communicates with other nodes by sending/receiving initialization data. These nodes may be mapped to the same or different Grid machines. Like NPB, NGB specifies several different classes (problem sizes). NGB also specifies the generic Grid services sufficient for running the suite. The implementor has the freedom to choose any Grid environment. We describe a reference implementation in Java, and present some scenarios for using NGB.

1 Introduction

The NAS Parallel Benchmarks (NPB) were designed to provide an objective measure of the capabilities of hardware and software systems to solve computationally intensive Computational Fluid Dynamics problems relevant to NASA. They are considered representative of an important segment of high performance scientific computing. At the time of NPB's inception in 1991 there were no accepted standards for programming parallel computers, and there was great diversity in hardware systems. It was deemed that any specific benchmark implementation would be unfairly biased towards a certain system configuration or programming paradigm. Hence, the first version of NPB, referred to as NPB1 [1], consisted of a paper-and-pencil specification, with virtually all aspects of the implementation left to the user. A reference implementation, mostly in Fortran, was provided for the convenience of users, but no claims were made about algorithmic efficiency or appropriateness for any particular system.

Despite its apparent lack of concreteness, NPB1 was embraced by vendors and researchers, and served as a fruitful testing ground for programming tools and compiler optimizations. Once a certain convergence in programming paradigms was reached, MPI (Message Passing Interface) being the first generally accepted standard, a source code implementation, termed NPB2, was released [2], which became the de facto yardstick for testing (parallelizing) compilers and tools. Recently, NPB2 was extended to other programming paradigms: OpenMP, HPF (High Performance Fortran) and Java [5, 6, 7].

Computational and data Grids [8, 16] are currently in a state of development comparable to that of high performance computers at the end of the 1980s. Several prototype Grid tools exist (e.g. Globus [9], Legion [18], CORBA [3], Sun Grid Engine [17], Condor [11]), whose relative merits are not well understood. Here we present a new benchmark suite, the NAS Grid Benchmarks (NGB), which aims to provide an exploration tool to Grid developers and users, similar to those which NPB provided to high-performance computing developers and users. NGB addresses one of the most salient features of Grid computing, namely the ability to execute distributed, communicating processes.

Our pencil-and-paper specification will serve as a uniform tool for testing functionality and efficiency of Grid environments, giving Grid developers a clear target. However, Grids add new dimensions to the computational process that require us to reconsider the traditional approach to benchmarking, which focuses on testing the limits of a single computer system. For example, the Grid is heterogeneous, involving different computers, networks, file servers, and execution environments. Moreover, Grid configurations are dynamic, deprecating common metrics like peak performance and throughput, and highlighting others, such as latency and failure localization.

Keeping these features in mind, we propose job turnaround time as the most important quantitative metric for benchmarking the Grid. Derivative metrics, such as aggregate cost of resources (disk space, CPU time, memory,

network bandwidth) used to complete the benchmark, are currently considered too poorly defined to have utility outside the benchmarker's own organization. NGB will provide a more detailed report on the performance of Grid components, including time to communicate data between any two benchmark tasks executed on (potentially different) platforms, and wall clock time for each task. But since NGB does not specify how many or which resources to employ, the detailed report is considered diagnostic in nature.

While it may be desirable to determine Grid efficiency in terms of performance achieved divided by resources used, this is an unattainable goal in the near future, since it requires normalization of resources (Grid *fungibility*). But it may well be useful for a Grid developer to examine increase, or degradation, of NGB performance if a single resource within a baseline configuration is varied, such as number of processors in a parallel computation, or number of network hops traversed in a distributed computation.

We expect that initial quantitative NGB performance, even on nominally identical Grid configurations, will show poor repeatability, because performance is often influenced by (currently) unpredictably fluctuating loads on certain Grid components. As Grid computing matures, more emphasis will be placed on reliability and quality of service. This will be achieved by more explicit control over Grid resources that will allow the user to allocate processing power, memory, network bandwidth, and disk space. NGB can help determine how well claimed resources are actually being delivered. However, such resource control will not be part of the NGB specification, just as the ability temporarily to suppress time sharing on the processors executing an NPB code was not part of the NPB specification.

We provide an NGB reference implementation in Java—a prototype of which is described in this paper—for the convenience of Grid developers. This implementation allows us to demonstrate some NGB usage scenarios, and to show some interesting effects that users may encounter when benchmarking a Grid.

2 Grid Benchmarking Requirements

Grids provide informational and computational environments that deliver new qualities and services to users. Grid benchmarks should explore these new parts of the Grid space and should avoid mapping previously covered computing areas. The Grid's informational component should provide fast, well focused and detailed search, storage, and retrieval of data, reliable collaboration between users, access to remote instruments, etc. These services, which are relatively application specific, require their own set of benchmarks, to be developed elsewhere. We concentrate, instead, on the computational aspect of the Grid environment, i.e. services used primarily for running computationally demanding jobs and processing substantial data sets. Hence, useful benchmark metrics are turnaround time and throughput. Turnaround time is time between starting a job and obtaining the resulting data. Throughput is submission volume possible without affecting the turnaround time.

For benchmarking throughput, the Grid has to be stressed to the limit of one of its resources. This can be done, for example, by submitting a large number of instances of the same job with different initial data (parameter study). A throughput benchmark must be intrusive, consuming at least one of the Grid resources completely. We consider intrusion undesirable for a Grid benchmark, one of whose important functions is to provide continual information on the health of the Grid environment. Hence, we will not measure throughput.

NGB tasks are defined in terms of data flow graphs (see Sections 3 and 4), whose nodes and arcs represent computations and communications, respectively. An NGB measurement of Grid performance is a report on the execution trace, which includes durations of execution of each node and of transmission along each arc. Summation of such durations along a critical path in the instantiation of the NGB data flow graph (Section 4) on the Grid gives turnaround time, which is also reported.

NGB should be *representative* of tasks typically executed on the Grid, but should also specify well-defined, measurable quantities of work. This precludes, for example, any interactive or nondeterministic processes. A benchmark performance figure is meaningless if the results are wrong, so it is important that a reliable *verification test* be provided.

NGB should also measure, to some extent, the data transfer capabilities of the communication network, particularly *latency* and *bandwidth*. Latencies are automatically included in the turnaround time if communicating tasks of the NGB are executed on a nontrivial subset of the Grid. In order to test bandwidth the benchmark has to send sizeable data volumes. Suitable candidate applications for Grid computing are relatively coarse-grained, as latencies between geographically separated Grid platforms are often large. This characteristic should be reflected in NGB.

NGB should contain *little initialization data*. This is a consequence of the paper-and-pencil specification of NGB, since any initialization data should be described in just a few pages of typed text. While this requirement could be waived, in principle, once a down-loadable NGB source code implementation is provided, we consider a fairly small source code distribution size desirable.

Grid resources are typically controlled by a certification mechanism that allows a legitimate user to access resources anywhere on the Grid. This means that the user's ability to run jobs is independent of having accounts on the individual machines constituting the Grid. NGB should test the availability of this and other basic Grid services. Paraphrasing

the nomenclature of [8, p. 37], we consider the following set of basic services necessary and sufficient for running jobs on the Grid: *authenticate, create task, communicate.* The NGB uses these three services and can be executed in any Grid implementation that provides them.

NGB will neither measure nor require security and fault tolerance, even though these are crucial ingredients of a successful Grid. They are very hard to quantify. However, it is envisioned that they will become an informal part of NGB performance reporting. For example, turning security on/off may affect NGB turnaround time, and NGB failure rates can indicate Grid reliability. Similarly, while it is vital to know which resources were involved in a certain performance result, we have no way of formalizing their characterization at present. We envision that resource usage will also become an informal part of NGB performance reporting, for studying the trade-off between turnaround time and consumed resources, and eventually for pricing Grid resources.

3 NGB Design

To meet the Grid benchmarking requirements we base NGB on two fundamental concepts, listed below.

Basic set of Grid services. Services necessary and sufficient for running NGB are authenticate, create task, and communicate (see Section 2). They do not assume any particular implementation, hence no Globus, Legion, Condor, CORBA, Grid Engine or other middleware bindings are specified.

Modular structure, open architecture. An NGB of a particular class (problem size) is specified by a data flow graph encapsulating NGB tasks (NPB codes) and communications between these tasks. Each node and arc of the graph is endowed with a verification test to determine correctness of its action. This makes it easy to change parameters, organize the benchmarks into classes, and diversify the tasks in the future. Users can augment or customize the NGB with their own favorite applications, as long as proper verification tests are supplied, but only the NAS NGB version allows comparisons among different users. The decision to use NPB codes in the baseline NGB, specifically BT, SP, LU, MG, and FT, is motivated as follows.

- The NPB codes are well-studied, well-understood, portable, and widely accepted as scientific benchmark codes.
- Solid verification procedures for NPB already exist.
- The NPB codes require no interaction, and no data files to start, in principle (but see next item).
- The NPB codes produce sizeable arrays representing solutions on discretization meshes. These can be used as input for any of the other codes, since each is based on structured discretization meshes covering the same physical domain. Hence, it is fairly straightforward to construct simple but plausible dependency graphs representing sets of interrelated tasks on the Grid, with significant data flows (solution arrays) connecting them.
- The granularity of the benchmark can easily be controlled by varying the number of iterations carried out by each NPB code.
- The NPB codes perform operations that can sensibly symbolize scientific computation (flow solvers: SP, BT, LU), post-processing (data smoother: MG), and visualization (spectral analysis: FT). Collections of such tasks are deemed suitable candidates for Grid computing.
- Good parallel versions of all NPB codes exist, which enables balancing the load of complicated Grid tasks by assigning different amounts of computational resources to different subtasks.

4 NGB Data Flow Graph

An instance of NGB comprises a collection of NPB codes, each of which solves a problem defined on a fixed, logically cubic discretization mesh. Each NPB code (BT, SP, LU, MG, or FT) is specified by class (mesh size), number of iterations, provider(s) of the input data, and consumer(s) of solution values. Hence, an instance of NGB is specified by a *Data Flow Graph* (DFG), see Figure 1. The DFG consists of nodes connected by directed arcs. It is constructed such that there is a directed path from any node to the sink node of the graph (indicated by *Report* in Figure 1). This is necessary to ensure that any failing node will be registered.

All NPB's mesh based problems are defined on the three-dimensional unit cube. However, even within the same problem class (S, W, A, B, or C) there are different mesh sizes for the different benchmark codes. Discretization points of meshes of different size generally do not coincide. In order to use the output from one NPB code as input for another, we interpolate the data tri-linearly, and take arithmetic averages of multiple inputs. The methods used by NPB preserve numerical stability under these operations.

DFG Node. Each node (except *Launch* and *Report*) represents a single computational task. It has a set of input and output arcs, an interpolator, a solver, and a verifier. If a node is connected to the source node (indicated by *Launch* in Figure 1), it receives control directives to initiate the computation. Otherwise it receives input data from other nodes through its input arcs, verifies correctness of the input data

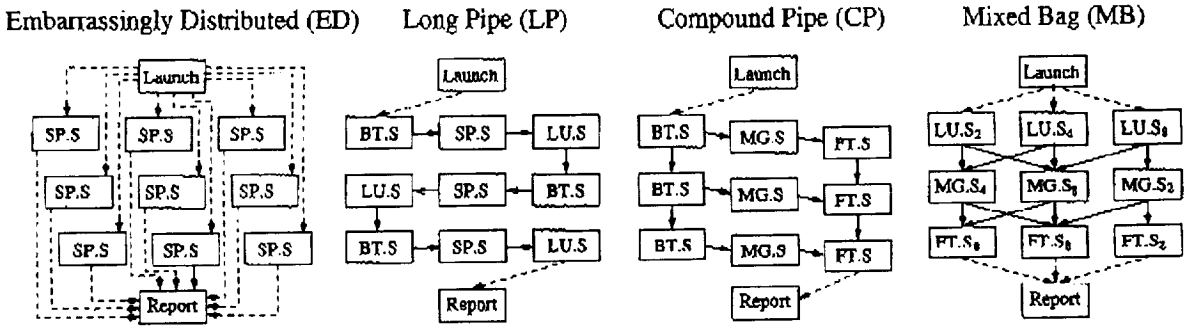

Figure 1. Data flow graphs of NGB, class S (sample size). Solid arrows signify data and control flow. Dashed arrows signify control flow only. Subscripts of MB tasks indicate the number of iterations carried out by an NPB code.

and interpolates it to calculate initial conditions. Each node applies the solver to the initial data, and verifies correctness of the computed result. If the node is not connected to the sink node (indicated by *Report* in Figure 1), it sends the computed solution to all output arcs. Otherwise it sends verification status and timing information to the sink node. Each node contains an instance of NPB2. The implementor is free to attach to the node information on the computational resources required for performing its functions, which can be used by a scheduler, see below.

DFG Arc. An arc has tail and head nodes. It represents transmission of data from the tail to the head. The implementor is free to attach to the arc information on the communication resources required for performing its functions, which can be used by a scheduler, see below. Communication along arcs takes place nominally through files (streams, sockets). These may not be cached, but must be created anew for each benchmark run. Dashed arcs in Figure 1 connect the nodes *Launch* and *Report* to the rest of the graph. They carry no computational data, but are required for control, status, and timing.

NGB Graph Set. NGB employs graphs named Embarrassingly Distributed (ED), Long Pipe (LP), Compound Pipe (CP), and Mixed Bag (MB), as shown in Figure 1. ED represents the important class of Grid applications called parameter studies, which constitute multiple independent runs of the same program, but with different input parameters. At NASA Ames, flow solvers—symbolized by SP—are often used for such studies. LP represents long chains of processes, such as a set of flow computations that are run one after the other, as is customary when breaking up very long running simulations into series of tasks. CP represents chains of compound processes, like those encountered when visualizing flow solutions as the simulation progresses (see itemized list in Section 3). MB is similar to CP, but now the emphasis is on introducing asymmetry. Different amounts of data are transferred between different tasks, and some tasks take longer than others, presenting a tough job to a scheduling agent that needs to map DFG nodes to resources.

Many important scientific applications consist of iterative processes that are naturally defined in terms of cyclic graphs. Since NGB measures turnaround time, all graphs in the graphs set are noncyclic and finite. They can nonetheless be used to mimic iterative processes, which have as basic building blocks (sets of) computational modules whose output can be transformed into input for the same module(s). LP, for example, can be viewed as a partially unrolled cyclic process.

Nodes and arcs are abstract components of the DFG. They are *instantiated* on a particular set of resources on the Grid, but we do not specify how or where a certain node or arc of the graph is instantiated. This is part of the benchmark execution. Executing an arc does not necessarily imply accessing a remote machine; multiple nodes may be mapped to the same platform.

An NGB of a particular class will be submitted for execution on the Grid, presumably to a scheduler. In case the user does not (want to) specify individual resources for nodes and arcs, the scheduler can make a decision about assigning appropriate Grid resources, depending on the availability of a description of resources required. This description may include number of floating point operations, memory size, and I/O volume.

5 Reference Implementation

In keeping with the spirit of the original NPBs, we provide a reference implementation that conforms to the rules of the NGB, and can serve as a starting point for developers or users of other Grid environments. It is envisioned that

standardization of Grid middleware will lead to one or more optimized source code implementations that can be downloaded and run as is (cf. [2]).

For the implementation we use the Java platform [10], since it has all the necessary features for creating a rudimentary Grid and NGB application in a compact and portable form. Moreover, a mapping of basic Grid services onto well-established Java constructs is relatively straightforward, and is specified as follows.

Authentication is provided by the NGB Pad (a Java application written for NGB), which requires the user to type a user name and password. This authentication acts as a proxy for each NGB task request issued from the NGB Pad. Java support for restricted access to services provided by the reference implementation is contained in package `java.security`. We use Java to enable *task creation* through the object lookup mechanism and native process creation contained in the `java.rmi` and `java.jni` packages, respectively. *Communication* (along arcs of the graph) is handled by the remote method invocation mechanism and file I/O, contained in the `java.rmi` and `java.io` packages, respectively.

The reference implementation involves two levels: the Grid level, which provides the basic Grid services, and the application level, which makes use of the services to implement the NGB tasks. NGB developers are expected only to implement the application level.

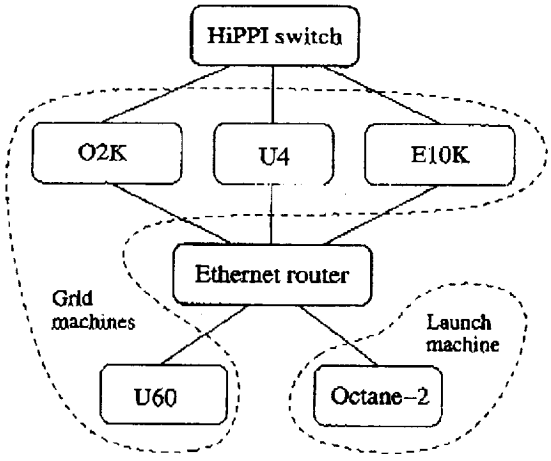

Figure 2. Network topology of machines used in the experiments.

Grid level. Our rudimentary Grid environment services the user's requests for running benchmarks and for reporting results. Installation of the environment by a Grid administrator involves the following steps:

- Install Java registry and bind services to the registry on all Grid machines (executing NPB codes). At present, the services are specific to NGB, but they can be made completely generic. Once they are, the Grid administrator no longer needs to be involved in installing user codes.
- Specify the security policy and eligible users.
- Install NGB Pads on Launch machines—from which NGB execution is initiated—to provide authentication and access to Grid services. At present, authentication and the application interface are integrated, which limits the generality of our rudimentary Grid environment.

Application level. The application level uses the basic Grid services to implement the NGB. Launch machines are created by the installation of the NGB Pad. The Pad, which can run on any machine that is able to access Grid machines via the HTTP protocol, is a graphical user interface (Figure 3) that encapsulates the basic Grid services. It reads a textual description of the DFG, turns it into a graph object, submits it for execution on the Grid specified within the DFG, and collects and displays the benchmark performance report. Currently, nodes of the DFG have to be mapped onto the Grid explicitly. The mapping includes the name of a Grid machine, number of processors used, and a programming paradigm (MPI, Java, HPF, OpenMP). We use an explicit mapping, since a connection with a Grid scheduler (resource broker) has not been implemented at the time of writing. In return, the user gets a benchmark performance report containing turnaround time and verification status for the whole benchmark, and for each node in the DFG.

Benchmark installation and execution involves the following steps:

- Compile and install NPBs on Grid machines (currently requires Grid administrator privileges and is therefore carried out in the installation step at the Grid level).
- Create DFG (in textual format).
- Invoke NGB Pad.
- Identify self as NGB Grid user.
- Submit DFG for execution on the Grid.
- Obtain benchmark report.

Experimental Setup. We installed the NGB services on a compact Grid comprised of 4 machines at NAS: a 24-processor SGI Origin 2000 (O2K), a 16-processor SUN E10000 (E10K), a 4-processor SUN Ultra-4 (U4), and a 2-processor SUN Ultra-60 (U60). The machines were connected by fast, switched Ethernet—supplemented with some HiPPI connections—with a typical round-trip time of 0.5 ms, see Figure 2. Only one processor was used on each of the machines. The NGB Pad ran on a single processor

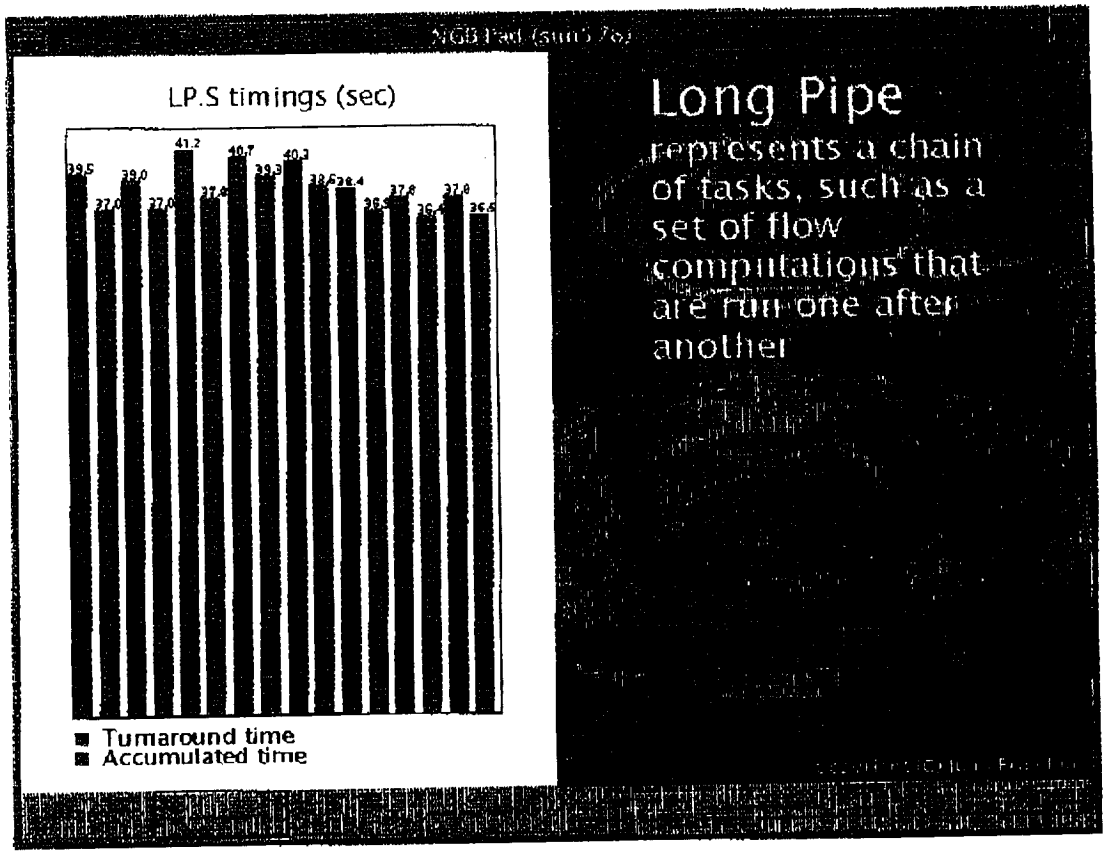

Figure 3. Prototype NGB Pad Graphical User Interface. Background artwork reproduced with permission of the artist, Julia Frumkin.

SGI Octane-2 by a user who did not have accounts on two of the four Grid machines.

Experimental Results. We executed the LP.S benchmark (Figure 1), mapped onto the ring U60-O2K-E10K-U4, in three modes, illustrated in Figure 4. Here each "wave" constitutes a snapshot of an instance of LP.S, which has at most one executing task (sharp peak) at any one time. Figures 4b and 4c each show snapshots of two instances of LP.S, mapped onto the ring of Grid machines in the same (b) and opposite (c) directions, respectively. The co-directed waves were started at different times, so that they never occupied the same machine simultaneously. We measured the turnaround time of LP.S and accumulated CPU time for all nodes of LP.S for a number of runs during a single day. Results shown in Figure 5 provide quantitative insight into Grid overhead and timing consistency.

The graphs show that the compact Grid—whose machines were not running in dedicated mode for our experiments—exhibited the timing consistency of a single processor in a multiuser mode, keeping variation of the turnaround time within 10%. We define Grid overhead as the difference between total turnaround time—which includes time spent in communications—and accumulated CPU time within the nodes of a *node critical path*. A node critical path is defined as the critical path of the instantiation of the DFG when ignoring all communication and synchronization times. For the LP benchmark the Grid overhead is simply the difference between turnaround time and accumulated CPU time. It follows from Figure 5 that the Grid overhead was below 5% percent for all cases, indicating that the communications did not dominate the benchmark executions, and that the Java server overhead was acceptable. Apparently, the load experienced by the NAS machines was steady during the period of testing.

We also made the following observations in our experiments, see Figure 5. Solitary waves (left) exhibited a steady

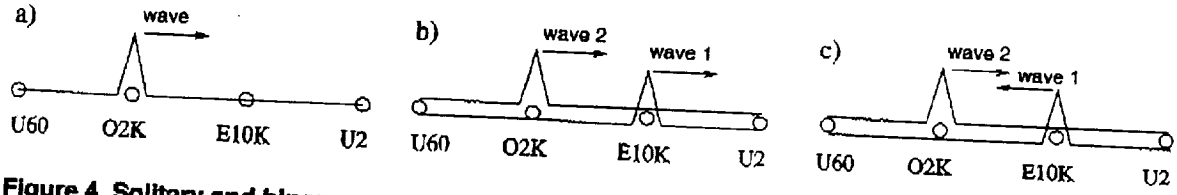

Figure 4. Solitary and binary waves in Grid space. Peaks indicate processor activity. The two binary waves travel in the same (b) and opposite (c) directions, respectively.

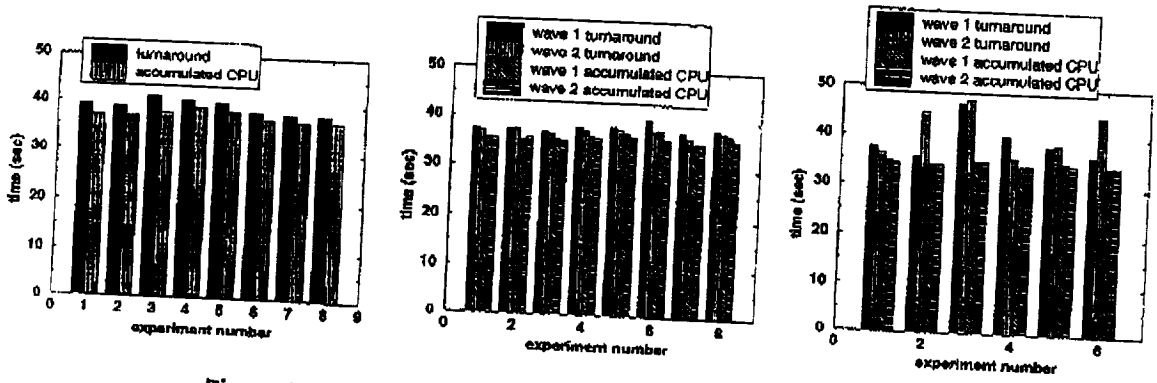

Figure 5. One-day sampling of compact Grid performance using L.P.S.

behavior in Grid space. Co-directed binary waves (middle) exhibited behavior similar to that of solitary waves. The second wave, traveling in the "wake" of the first, had a smaller average Grid overhead. Collisions of antidirected binary waves (right) resulted in occasional increase of Grid overhead.

While we should resist drawing deep conclusions from this simple experiment, it illustrates how NGB may serve to gain insight in the quantitative behavior of distributed applications on the Grid.

6 Related Work and Conclusions

A number of projects intend to probe the quality of Grid environments, for example, the Globus Heart Beat Monitor [15] (part of the Globus [9] metacomputing toolkit), the Network Weather Service [14], and the simulation projects WARMstones [4], MicroGrid [12], and Bricks [13].

The Heart Beat Monitor allows a process to be tracked and periodic heartbeats to be sent to one or more monitors, but it provides no performance statistics. The goal of the Network Weather Service is to provide accurate forecasts of dynamically changing performance characteristics from a distributed set of metacomputing resources. It does not aim to provide a standardized set of tasks whose performance on different hardware/software systems can be compared meaningfully.

The WARMstones project does plan to provide a touchstone for realistic comparisons between scheduling algorithms for wide-area distributed systems, but it will do so by simulating the system under consideration and deriving performance results from the simulation. The Bricks and MicroGrid projects have similar goals. MicroGrid specifically targets distributed applications executed under control of Globus [9].

NGB complements the above efforts by providing standardized tests that are not tied to any particular Grid middleware, but that nonetheless can be executed within a real Grid environment. This allows NGB to be used for regression tests of Grid services, and to compare between different Grid environments. A full specification and reference implementation will be released shortly.

Acknowledgements. This work was supported by the NASA High Performance Computing and Communications Program, RTOP #725-10-31. We are grateful to the members of the Algorithms, Tools, and Architectures group in NASA Ames' NAS division for constructive suggestions, and to Sandy Johan for reviewing our Java code.

References

[1] D.H. Bailey, J. Barton, T. Lasinski, and H. Simon (Eds.). *The NAS Parallel Benchmarks.* NAS Technical

Report RNR-91-002, NASA Ames Research Center, Moffett Field, CA, 1991.

[2] D.H. Bailey, T. Harris, W.C. Saphir, R.F. Van der Wijngaart, A.C. Woo, M. Yarrow. *The NAS Parallel Benchmarks 2.0.* NAS Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[3] R. Ben-Naten. *CORBA: A Guide to Common Object Request Broker Architecture.* McGraw-Hill, New York, 1995.

[4] S.J. Chapin, *WARMstones: Benchmarking Wide-Area Resource Management Schedulers.* Draft white paper, Syracuse University, http://www.hpdc.syr.edu/~chapin/currentproj.html.

[5] H. Jin, M. Frumkin, J. Yan. *The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance.* NAS Technical Report NAS-99-011, NASA Ames Research Center, Moffett Field, CA, 1999.

[6] M. Frumkin, H. Jin, J. Yan. *Implementation of NAS Parallel Benchmarks in High Performance Fortran.* Proc. International Parallel Processing Symposium, 1999, http://ipdps.eece.unm.edu.

[7] M. Frumkin, M. Schultz, H. Jin, J. Yan. *Implementation of NAS Parallel Benchmarks in Java.* Presented at a Poster session at ACM 2000 Java Grande Conference, 2000.

[8] *The Grid. Blueprint for a New Computing Infrastructure.* I. Foster, C. Kesselman, Eds., Morgan Kaufmann Publishers Inc., San Francisco, CA, 1999.

[9] I. Foster, C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit.* Int. J. Supercomputer Applications, 11(2):115-128, 1997, http://www.globus.org.

[10] C.S. Horstmann, G. Cornell. *Core Java 2, Volume 2: Advanced Features.* 4th edition, Prentice Hall, 1999, see also http://java.sun.com/j2se/1.3/docs.

[11] M. Livny, J. Basney, R. Raman, T. Tannenbaum. *Mechanisms for High Throughput Computing.* SPEEDUP Journal, Vol. 11(1), 1997, http://www.cs.wisc.edu/condor/.

[12] H.J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, A. Chien. *The MicroGrid: a Scientific Tool for Modeling Computational Grids.* Proc. Supercomputing 2000, Dallas, TX, 2000.

[13] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, U. Nagashima. *Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms.* Proc. High-Performance and Distributed Computing 8, pp. 97-104, 1999.

[14] R. Wolski, N.T. Spring, J. Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.* J. Future Generation Computing Systems, 1999, also UCSD Technical Report Number TR-CS98-599, 1998, http://nws.npaci.edu/NWS/.

[15] *Globus Heartbeat Monitor.* http://www.globus.org/hbm/.

[16] *NASA Information Power Grid.* http://www.nas.nasa.gov/IPG.

[17] *Codine 5.2 Manual, Revision A.* Sun Microsystems, Inc., Palo Alto, CA, September 2000, http://www.sun.com/gridware.

[18] *Legion 1.6, Developer Manual.* The Legion Research Group, Dept. Computer Science, U. Virginia, Charlottesville, VA, 1999, http://legion.virginia.edu.