Asymmetric Load Balancing on a Heterogeneous Cluster of PCs

Christopher A. Bohn *

Air Force Research Laboratory
Wright-Patterson AFB, OH, U.S.A.
christopher.bohn@sn.wpafb.af.mil

Gary B. Lamont
Jeffery K. Little Richard A. Raines
Air Force Institute of Technology
Wright-Patterson AFB, OH, U.S.A.
{gary.lamont jeffery.little richard.raines}@afit.af.mil

Abstract

With commercial supercomputers and homogeneous clusters of PCs, static load balancing is accomplished by assigning equal tasks to each processor. With heterogeneous clusters, system designers have the option of adding newer hardware that is more powerful than existing hardware. When this is done, assignment of equal tasks to each processor yields suboptimal performance.

This research addresses techniques by which the sizes of the tasks are suitably matched to the processors and memories. Thus, more powerful nodes do more work, and less powerful nodes perform less work. We find that when the range of processing power is narrow, some benefit can be achieved with asymmetric load balancing. When the range of processing power is broad, dramatic improvements in performance are realized – our experiments have shown up to 92% improvement when asymmetrically load balancing a modified version of the NAS Parallel Benchmarks' LU application on a heterogeneous cluster of Linux-powered PCs.

Keywords: Pile of PCs, Load Balancing, Linux

1 Introduction

Traditionally, supercomputers are designed with the objective of achieving the greatest computational and communication performance physically possible; the U.S. Department of Energy's Accelerated Strategic Computing Initiative is the current embodiment of this niche. At the other extreme are low-cost computer architectures, where the performance is subordinate to the end-user price; commodity personal computers (PC) fill this role. Between lay the designs that focus on the price-performance ratio, exemplified by scientific workstations [2]. Advances in the performance of commodity PCs and commodity networks without corresponding increases in price led to the discovery that limited supercomputing performance can be realized with clusters of PCs, at a price-performance ratio an order of magnitude better than is possible with typical supercomputers [3]. The AFIT Bimodal Cluster (ABC) is one such system.

The ABC is a continuously-evolving Pile of PCs (PoPC) built with the just-in-time approach to hardware configuration. Early in the project, we decided future expansion of the ABC would not be limited by previous design decisions; as such, the ABC's hardware would be heterogeneous.[1] Currently, the ABC consists of twelve Intel Pentium and Pentium II processors, interconnected by a 100 Mbps Fast Ethernet switch. It can operate under either or both Microsoft Windows NT and Linux, but only Linux is within this paper's scope.

Consequent from the decision to make the ABC heterogeneous is that the performance realized by new hardware would be limited by

*The material reported herein is based on the author's thesis submitted in partial fulfillment of the requirements for the Master of Science in Computer Engineering degree at the Air Force Institute of Technology, Wright-Patterson AFB, OH

[1] In the context of this document, the ABC is heterogeneous in that the processors are clocked at different rates and have different implementations of the Intel Architecture, and that the memories are different sizes and are clocked at different rates.

older hardware. If workloads were matched to processors' capabilities, this limitation would be overcome and older hardware would continue to contribute to the solution of computational challenges. Thus, obsolescence of older technologies would be delayed, further reducing the cost of high performance computing.

2 Background

While static and dynamic load balancing for homogeneous parallel computing platforms is well-studied, load balancing for heterogeneous parallel systems is a relatively new subject of investigation with a less-explored landscape [4]. On a heterogeneous platform, the goal is the same: to minimize idle processor time and, by extension, to lower the execution wall-clock time. This is done by distributing the work such that no processor is waiting for the completion of another [5]. The critical problem is that the load balancing techniques developed for homogeneous systems are based on fixed parameters; in a heterogeneous system, these parameters are not always known *a priori* [4].

Addressing this problem, researchers at the University of Paderborn describe a *dynamic* load balancing technique that uses observed computational and communication performance to predict the time a task would complete on a given node and the time needed to query a node [4]. But incorporating dynamic load balancing in previously-written applications that do not have embedded dynamic load balancing is impractical. Further, dynamic load balancing is not needed if static load balancing can achieve the desired performance.

At the other extreme is the approach used by researchers at the Universidade de Coimbra, in which the issue of load imbalance is ignored. Their nodes are workstations that are dynamically "donated" to the project, using the internet for interprocess communication. Load balancing is achieved by reducing the problem to the finest grain possible and never expecting a worker process to execute more than one simple task at a time [6]. For a parallel processing platform that does not suffer from the high latencies of the internet, this approach would cause interprocess communication to dominate the performance to an undesirable degree.

Researchers at Brigham Young University explored a third approach that uses *static* load balancing. Their compute nodes are workstations donated from within the university, using the university local-area-network for interprocess communication. Rather than relying on run-time performance, they measure the relative capabilities of each node by executing the HINT benchmark [7] once on each node and storing the results for future reference [5]. This approach is the best suited for "regular" applications which have a deterministic path to solution. Nonetheless, it is computationally expensive to prepare, scales poorly, and is computationally expensive to port. Since HINT *must* be executed on each donated workstation, then a workstation introduced to the system for the first time cannot be used until benchmarked with HINT. Further, if the application is ported to a new system, then the entire system must be initially rated with HINT.

Analytic and experimental cluster load balancing efforts continue, evidenced in recent journals and conference proceedings.

3 Approach

Having considered how others have addressed the asymmetric load balancing problem, we now consider our particular approach.

3.1 Asymmetric Load Balancing

To demonstrate the benefit of asymmetric load balancing, we must select an appropriate application for our tests. We select the NAS Parallel Benchmarks' (NPB) LU simulated computational fluid dynamics (CFD) application [8] because of three major factors [9]:

- it is designed specifically to have communication and computation patterns similar to "real" CFD applications;
- it provides "self-verification" to establish the solution is correct and that our modifications do not affect the correctness; and

- it is a well-known and easily accessible piece of software, which makes it easier for others to reproduce our results or to compare their own results with ours.

In the unmodified LU code, the problem is partitioned among processors in block-checkerboard fashion [9]. We modified this to a columnwise block-striped partitioning for two reasons. First, it simplifies the task of asymmetrically adjusting the partitions. Second, it allows us to remove LU's power-of-two requirement for the number of processors; this permits us to determine if asymmetric load balancing allows the weaker processors to contribute to a faster solution, as described in Section 3.3.

The first obstacle that must be overcome is Fortran 77's lack of dynamic memory allocation. This means the memory allocation must be specified at *compile-time*. However, knowledge of which processors are being used is not determined until *run-time*, so the problem set must be partitioned after memory allocation.

The obvious solution is to use Fortran 90, but a Fortran 90 compiler was not available for these experiments. Our solution is to allocate sufficient memory on each processor to hold the largest possible partition. The problem with this solution is that the extra memory management imposes an extra overhead from which the original application does not suffer. To determine the merits of asymmetric load balancing, we compensate for this "memory penalty" by comparing the load balanced performance against the performance of the application without asymmetric load balancing but with the extra memory management.

3.2 Measurement of Compute Node Performance

Our algorithm must determine the performance of each node. While run-time performance certainly provides the most accurate assessment, we use static load balancing, which precludes knowledge about run-time performance. Thus, we must obtain this performance information before partitioning the problem. In selecting a measurement technique, there are certain properties we desire. We want it:

- to accurately predict the performance our application realizes while remaining a general indicator of performance so that others can make use of its measurements;
- to be scalable and portable – that is, we do not want to fully benchmark every new node as the ABC grows, and we want it to be easily transferred to other clusters; and
- to be computationally inexpensive.

3.2.1 Inexpensive Metrics

We first explore a couple approaches to measuring system performance that are computationally inexpensive. They do not appreciably impact the run-time performance, even when the measurement takes place at run-time.

The first of these two metrics takes advantage of the Linux file structure. Like all UNIX systems, Linux has a directory called `/proc`. One of these files, `cpuinfo`, contains information such as the processor's manufacturer and model, as well as a performance-related value called "BogoMIPS." BogoMIPS, meaning "bogus MIPS," is calculated when Linux boots to calibrate certain timing loops [10]. BogoMIPS does not appear to be a good general or specific indicator of performance, but we shall consider it since accessing this value is so inexpensive.

An alternative is to write a simple routine that loops through a series of floating point operations to create a crude Mflops rating. Like BogoMIPS, this Mflops rating is not necessarily a good indicator of performance, but it is also computationally inexpensive. Further, unlike BogoMIPS, this rating is not dependent on the operating system's file structure.

3.2.2 Detailed Benchmark

Since we wish to obtain an accurate measure of the nodes' relative capabilities, we cannot rely solely on inexpensive metrics. We considered four common benchmarks, LinPack, NPB–serial, SPEC, and HINT. We choose HINT [7] because it evaluates processor and memory performance for any datatype and returns a single "QUIPS" value, which corresponds well with the relative performance NPB obtains [5]. We conclude HINT is a good metric for our specific case and general use as well.

The difficulty, though, is that HINT requires hours to execute because it continues to refine its solution to greater and greater detail until the computer cannot provide further improvements. Snell, *et.al.* [5], overcome this problem by executing HINT once on each node and storing the results for later use. While this eliminates the need to execute HINT at run-time, it suffers from not scaling well and not being cheaply portable, as mentioned in Section 2.

We overcome the necessity to execute HINT on each node by making the observation that, while the system as a whole is heterogeneous, many nodes are similar, even identical, to each other. If we could take advantage of this knowledge, then we need only run HINT on one node of each type. The issue, then, is how to determine the type of node on which a process is executing so that this information can be mapped to the HINT result. The answer is found in the inexpensive benchmarks. On the ABC, which currently uses exclusively Intel processors, the BogoMIPS value is sufficient to uniquely identify the type of processor. Should other manufacturers' processors be used, then the processor manufacturer & model is also required.

We, therefore, developed a library that can obtain any of the three metrics described: BogoMIPS, Mflops, or QUIPS. It also generates and makes use of maps from BogoMIPS to QUIPS and from Mflops to QUIPS, amortizing the cost of executing HINT over all future uses of these maps. On the ABC, HINT was only executed on four of the twelve nodes, reducing the overhead of executing HINT by two-thirds. Further, by making use of known values and interpolation rules, we do not necessarily need to benchmark new nodes with HINT when expanding the ABC or when porting the libary and map files to another system.

3.3 Design of Experiments

We are forced to limit the size of the problem we test to LU's A-class problem. The S- and W-classes are dismissed, as they do not sufficiently task the system. Memory limitations on one of the nodes precludes the B- and C-class problems when overallocating memory.[2]

The original LU application requires a power-of-two number of processors, so our primary points of comparison are those with two, four, and eight processors. We could blindly execute the software on every combination of two, four, and eight processors, but this is impractical and unnecessary. As the processors are not unique, we need not use every possible combination of processors to characterize the system; we could use every unique combination of 200 MHz, 333 MHz, 400 MHz, and 450 MHz processors. Given the finite time available for experiments, even this is undesirable.

Instead, we consider exactly what we wish to learn, namely, how asymmetric load balancing affects the performance. This can be obtained by examining different ranges of capabilities. At one extreme, we include both the 450 MHz processor and the 200 MHz processor. At the other extreme, we use only the 450 MHz processor and 400 MHz processors. Between the extremes, the least powerful node is a 333 MHz processor. Table 1 lists the combinations of processors we use for the experiments.

We also wish to know if asymmetric load balancing allows us to make use of the weakest processor in the cluster, or if the performance with load balancing is worse than the performance achieved without using that processor at all. This leads to tests using one, three, and seven processors in combinations that match the broadest combination of two, four, and eight processors, except for the absence of the 200 MHz processor. This is not possible with the unmodified code's checkerboard partitioning, but the versions with striped partitioning can still be used for these tests.

When comparing the performance obtained with asymmetric load balancing against that

[2] Studying the source code reveals that the A-class problem requires about 40 MB of memory, the B-class 160 MB, and the C-class 640 MB. One of the ABC's nodes has only 32 MB of main memory and 64 MB of swap space. Since our current asymmetric load balancing implementation requires sufficient memory on each node for the entire problem set, only the S-, W-, and A-class problems fit in this node's virtual memory.

Table 1: Combinations of processors used for experiments

	200MHz Pentium	333MHz PentiumII	400MHz PentiumII	450MHz PentiumII	Performance Improvement		
					BogoMIPS	Mflops	QUIPS
2 processors	1			1	$78.0 \pm 0.18\%$	$91.7 \pm 1.00\%$	$72.9 \pm 0.03\%$
		1		1	$15.8 \pm 3.31\%$	$15.7 \pm 4.32\%$	$12.5 \pm 9.55\%$
			1	1	$4.3 \pm 0.13\%$	$4.5 \pm 0.05\%$	$4.9 \pm 0.06\%$
4 processors	1	1	1	1	$65.1 \pm 0.05\%$	$74.4 \pm 1.37\%$	$87.9 \pm 0.17\%$
		1	2	1	indistinguishable	$-0.9 \pm 0.04\%$	$2.2 \pm 1.38\%$
			3	1	indistinguishable	$0.8 \pm 0.37\%$	$8.8 \pm 0.56\%$
8 processors	1	1	5	1	$55.8 \pm 12.9\%$	$49.5 \pm 16.9\%$	$68.0 \pm 8.1\%$
		1	6	1	$-4.3 \pm 1.0\%$	$-5.5 \pm 0.9\%$	$3.4 \pm 3.2\%$

without, we always test the set of values obtained with load balancing against the *best* performance obtained without. Likewise, when determining whether asymmetric load balancing permits improved performance by adding a weak processor, we compare against the best performance without the extra processor.

The remaining issue is statistical validation of our results. Ideally, we would run the tests dozens of times to obtain small confidence intervals. However, because of the time the tests require, we instead choose to execute each test five times, leaving the option open to run more tests if some results are statistically ambiguous. With only five measurement points, we cannot neglect the question of whether the performance results are normally distributed. We choose to err on the side of caution and use a test that does not rely on normality; specifically, we use the Wilcoxon signed rank test. In most cases, the results are clear-cut – either all data points show an improved performance, or none of them do. With five data points all greater than or all less than the best unbalanced performance, the Wilcoxon signed rank test allows us to conclude that the load balancing has (or has not) improved performance with a 0.03 level of significance.

4 Results & Analysis

The results of our experiments are summarized in Table 1 and Figures 1 & 2. As can be

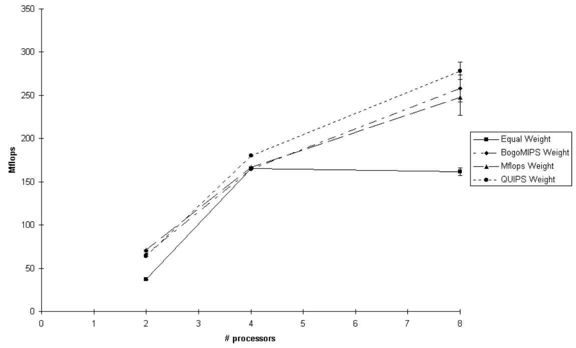

Figure 1: Performance by number of processors: broadest combination of processors

seen in Figure 1, when the range of processor capabilities is broad, all three weighting techniques provide an improvement over the "equal weight" performance (21.7% to 92.3%). Figure 2, on the other hand, shows that when the range of processor performance is narrow, asymmetric load balancing does not provide as much improvement (no greater than 9.7%).

Examination of Figures 3–5 shows that the 200 MHz Pentium always slows the arrival to solution when asymmetric load balancing is not used. Given this, might we have been better to not have used the Pentium? The answer, in the two- and four-processor cases, is no.

Examining the use of the 200 MHz Pentium and the 450 MHz Pentium II (Figure 3), we see that when using the best weighting (here,

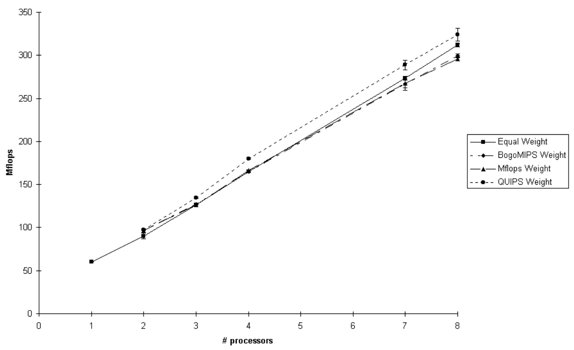

Figure 2: Performance by of number of processors: fastest combination of processors

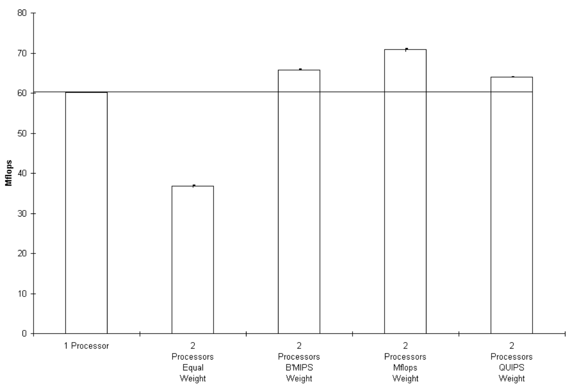

Figure 3: Comparison of two-processor performance with one-processor performance

the Mflops weighting), we realize a $17.8 \pm 0.61\%$ improvement over the best uniprocessor performance. Similarly, when comparing the three- and four-processor cases (Figure 4) using the QUIPS-weighted load balancing, the four-processor case with the Pentium outperforms the best three-processor performance by $5.9 \pm 0.10\%$. Finally, in the case of seven- vs. eight-processors (Figure 5), asymmetric load balancing does not allow the slowest processor to contribute to a faster solution. If the seven-processor case uses asymmetric load balancing, then it outperforms the eight-processor case, regardless of the weighting used.

Why doesn't load balancing with the eight-processor case permit us to make full use of the processors? The answer lies in a partitioning requirement from the original code, namely that no partition can be fewer than four elements wide, most likely to prevent interprocess communication from dominating the application. Honoring this, we wrote the load balancing algorithm to reshift the balance to prevent any processor from having a partition less than four elements wide. Using the weights returned from the NodeMetric library, the Pentium's "fair share" is a tile two or three elements wide, so the Pentium is still overtaxed, and the other processors are not being used to their fullest extent.

5 Future Asymmetric Load Balancing Efforts

We must reexamine why the LU application is written to prevent partitions narrower than four elements. We suspect the reason is not to place a lower limit on the size of the partitions, but rather to place an upper limit on the number of processors that may be used, to prevent interprocess communication from harming performance. So long as we do not exceed this upper limit, there is no reason the weakest processor cannot be responsible for a partition narrower than four elements. So long as its "fair share" is at least one element wide, the weakest processor should be able to contribute to the solution if not tasked with more than its "fair share." For this reason, the lower limit on the width of the tiles could be removed in the load balanced code. doing so should make more efficient use of the processors such that the eight-processor performance with the 200 MHz Pentium exceeds the seven-processor performance without the 200 MHz Pentium.

A Fortran 90 compiler does not have to overallocate memory to permit asymmetric load balancing. Instead, it can dynamically allocate memory at run-time, after the partition sizes have been determined. The application should be modified to make use of dynamic allocation, and the tests should be run again to ascertain the effect on performance that asymmetric load balancing with dynamic memory

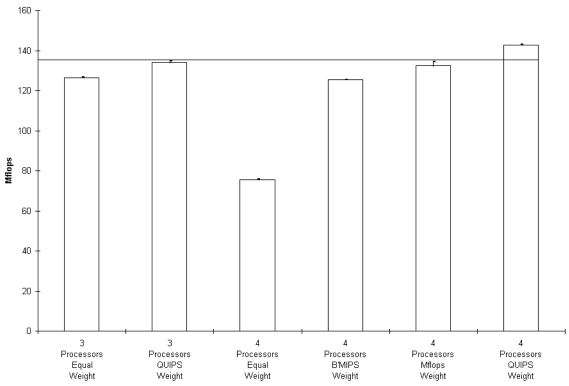

Figure 4: Comparison of four-processor performance with three-processor performance

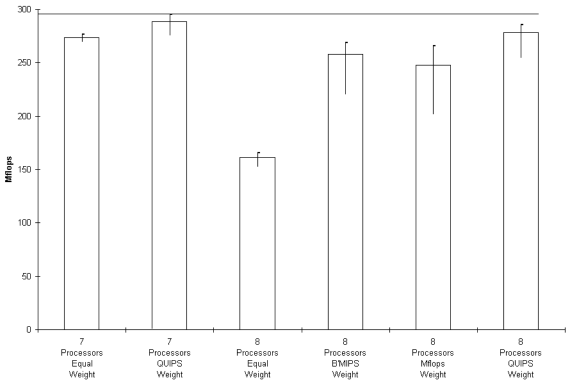

Figure 5: Comparison of eight-processor performance with seven-processor performance

allocation has. Once dynamic memory allocation is used, larger problem sizes can also be executed, since we no longer would be allocating more memory than is physically available on a given node. Based on the penalties imposed due to overallocating memory in these experiments, we expect dynamic memory allocation to provide a 5% to 20% performance improvement over the results reported here.

Another direction involves problem space partitioning. To make use of a greater number of processors, we should reimplement a block-checkerboard partitioning. While more difficult to adjust the partitions than with block-striped partitioning, it is not impossible, and we would then be able to make use of $\mathcal{O}(n^2)$ processors instead of $\mathcal{O}(n)$ processors.

We also are interested in porting asymmetric load balancing to other operating systems. Using asymmetric load balancing on cluster of workstations using another UNIX system should be straight-forward, except that BogoMIPS would be unavailable – amortizing the computational cost of the HINT benchmark would be accomplished by mapping the Mflops value obtained when calculating pi to HINT's results. Another field of research using the ABC is parallel processing using Windows NT; we expect the greatest challenge in porting the NodeMetric library to Windows NT to be the semantics of system calls.

6 Conclusions

We have determined that for the two- and four-processor cases, the load balancing allows us to make full use of the available processors; if asymmetric load balancing were not used, we may realize better performance by leaving out the weakest processor. For the eight-processor case, asymmetric load balancing has not enabled full use of the available processors, because we left a legacy minimum partition width in the application that is now unnecessary. After correcting this problem, we expect future experiments to show that we can efficiently utilize all processors in the cluster.

We also observed that we have reduced the time needed to make use of the HINT benchmark. We used just over forty-three hours of processor time to build the NodeMetric library's maps for five intrinsic data types. Had we been required to execute the HINT benchmark on *every* node, even if only for the double-precision floating point version, then just over fifty-one hours of processor time would be required. Further, when we add nodes are add to the cluster, we need not first execute the HINT benchmark on them.

After correcting for the penalty imposed due to overallocating memory, we find that the QUIPS rating consistently provides better performance than the unbalanced code, regardless of the range of processor capabilities, up to eight processors. If the range of processor capabilities is sufficiently wide, then all three weighting techniques provide an improvement over the unbalanced code.

While tested with only one application to date, we believe asymmetric load balancing is a general-purpose tool that can be used with any data-decomposed regular problem and, with some extensions, can be used with irregular problems as well. We also expect that as more processors are used, asymmetric load balancing will continue to allow us to use clusters of PCs efficiently.

Many traditional assumptions about supercomputing platforms do not hold true with commodity clusters, particularly when we realize that PoPCs have certain growth potentials that are not possible with "big iron" machines, such as the ability to add the most powerful processors as they become available, rather than limiting growth to the addition of more processors identical to those already in place.

With proper load balancing, computational scientists and engineers using PoPCs can efficiently use both the newest hardware in the system and the oldest, without the older hardware limiting the system's performance. In so doing, we conclude that the removal of older hardware is unnecessary even when the newer hardware has more than twice the performance. Researchers are then able to get more use out of their research dollar, and obsolescence of the older hardware is delayed.

References

[1] C.A. Bohn. Asymmetric load balancing on a heterogeneous cluster of PCs. MSCE Thesis, AFIT/GE/ENG/99M-02, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 1999.

[2] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach, 2d ed*, page 17. Morgan Kaufmann, San Francisco, 1996.

[3] T. Sterling, T. Cwik, D. Becker, J. Salmon, M. Warren, and B. Nitzberg. An assessment of beowulf-class computing for NASA requirements: Initial findings from the first NASA workshop on Beowulf-class clustered computing. In *Proceedings, IEEE Aerospace Conference*, 1998. http://loki-www.lanl.gov/p312.ps.

[4] T. Decker, R. Lüling, and S. Tschöke. A distributed load balancing algorithm for heterogeneous parallel computing systems. In *Proceedings, International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 933–940, 1998.

[5] Q. Snell, G. Judd, and M. Clement. Load balancing in a heterogeneous supercomputing environment. In *Proceedings, International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 951–957, 1998.

[6] L.M. Silva. Number-crunching with java applications. In *Proceedings, International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 379–385, 1998.

[7] HINT (Hierarchical INTegration). ftp://ftp.scl.ameslab.gov/HINT.

[8] The NAS Parallel Benchmarks. http://science.nas.nasa.gov/NPB.

[9] D.H. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS–95–020, 1995. http://science.nas.nasa.gov/TechReports/NASreports/NAS-95-020.ps.

[10] W. Van Dorst. BogoMips mini–HOWTO. The Linux Documentation Project. http://MetaLab.unc.edu/LDP/ldp.html, 1999.