Reflections on the Creation of a Real-time Parallel Benchmark Suite

Brian VanVoorst[1], Rakesh Jha[1], Subbu Ponnuswamy[1], Luiz Pires[1],
Chirag Nanavati[1], David Castanon[2]

[1]Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55441
{vanvoors, jha, subbu, pires, nanavati}@htc.honeywell.com

[2] ALPHATECH, Inc
50 Mall Road
Burlington, MA 01803
dac@bu.edu

Abstract. Standard benchmark suites are a popular way to measure and compare computers performance. The Honeywell Technology Center has developed two benchmarking suites for parallel computers, the C3I Parallel Benchmark Suite (C3IPBS)[1] and more recently, the Real Time Parallel Benchmark Suite (RTPBS)[2]. The C3IPB suite was similar to other benchmarks but focused on a new domain. The development of the real-time benchmark suite presented a number of new challenges. In this paper we share some of the lessons we have learned about the art of making parallel benchmarks.

1. Introduction

Benchmarks are used to evaluate and compare the performance of systems, using well-defined specifications and metrics. The Real Time Parallel Benchmark (RTPB) suite is a collection of benchmarks designed to help real-time C3I (command, control, communications and intelligence) parallel application designers evaluate parallel computing systems for real-time performance. Real time C3I applications impose unique requirements on a computing platform, and at the time this project was conceived, existing parallel benchmark suites (even those focusing on C3I applications) did not adequately address these needs. The results obtained from this benchmark

[1] This work was sponsored by USAF Rome Laboratory, contract number F30602-94-C-0084.

[2] This work was sponsored by DARPA via USAF Rome Laboratory contract number F30602-94-C-0084.

suite should help system architects, integrators and software designers better understand the factors that affect C3I real-time applications on parallel systems. Two goals of this projects were: 1) to help reduce the cost and time of development cycles, and 2) to facilitate discussions between the C3I community and parallel system architects.

Benchmarks suites come in many forms. The RTPB suite is a "pen and paper" benchmark suite like NAS 1.0 parallel benchmarks [15], meaning that the benchmarks themselves are specifications. The specification describes (in great detail sometimes) exactly what the benchmark is: what computations are to be performed, what input should be accepted and what the expected output should be, as well as a several other lower level details. In a sense, the specification states the objective of the benchmark and provides a detailed set of rules. When someone implements a benchmark based on the rules from the specification, then they have created a *benchmark implementation*, which is one realization or instance of the benchmark. Pen and paper benchmarks are used when there are considerable differences in the architectures on which that the benchmark may be implemented. By using the pen and paper approach, we let implementers choose the best possible implementation approach for a target platform. This is important, as we feel it would be improper to mandate one implementation (e.g. MPI) to be run on all machines when it is possible that on a given machine (e.g. a shared memory machine) a different implementation approach could produce better results. A side effect of pen and paper benchmarks is that they require considerable effort, both in developing specifications and in create implementations.

This paper summarizes our work on the benchmark suite and highlights some of the lessons we learned about benchmarking. This paper does not discuss the actual results from the benchmarks, as they have been discussed in other places [2, 4, 6]. The rest of this report is organized as follows: Section 2 describes the goals of the project, Section 3 presents a brief account of our accomplishments, Section 4 describes what we learned, Section 5 presents our conclusions.

2. Goals of the Project

The amount of work to be done by a C3I system is increasing rapidly as the quantity of the available information to be processed increases, and the expected quality of the results increases. Furthermore, the need for C3I systems with ever faster response times drives the need for increasingly shorter real-time deadlines. As an example, next generation sensors will produce tarabytes of data per second that must be processed in real-time as they scan the surface of the planet. To complicate matters, cost constraints force system designers to adopt commodity COTS components as the building blocks for their systems instead of special purpose hardware. The combination of these factors requires the use of commercial parallel processing systems for real-time C3I applications.

A system designer must consider several factors when choosing hardware for an embedded C3I application. Some of the relevant factors that must be considered include: single processor speed, scalability, real-time support and performance, efficiency of the algorithm on the machine, problem mapping, communication and mem-

ory requirements, fault tolerance, reliability, size, weight, power and the list goes on. Even if there are just a handful of architectures to consider, the amount of trade-off analysis required is staggering. Because of the amount of work involved, it is usually only the easily available metrics that are used for comparison (size, weight, power, processor speed in Mhz, published communication latency). The most important factor, the interactions between the actual algorithm and the system under study, are rarely considered. The Real Time Parallel Benchmark Suite provides efficient mechanisms to make these comparisons easily.

The benchmarks provide a new metric for comparison that allows systems to be evaluated side-by-side in an "apples-to-apples" manner. This new metric captures the real world interactions between applications and hardware with real input data, and reflects the kind of performance that will be seen when the system is deployed.

Consider the following example of how a system architect might use such a benchmark suite. Suppose a system architect has a C3I application with intensive computational requirements and real-time deadlines. From the system architect's point of view, the best possible situation is that there already exist benchmark results that capture the main computational and real-time requirements of the application. If the results are not available, but a similar benchmark exists, the system architect can port and run the benchmark on the platform of interest. If there is no such benchmark available, the architect can follow the methodology provided to specify a new benchmark and ask that vendors run this benchmark and submit results for comparison. Either way, in short order a fair comparison can be made between the platforms under consideration.

A secondary benefit is that the benchmarks serve to increase awareness and interest in real-time C3I applications on parallel machines. The NAS parallel benchmarks [15], which have become a standard for comparing machines for scientific computing, serve as an example. Because of the popularity of the NAS benchmarks and the emphasis placed on their results, vendors compete to improve their performance for CFD type applications. If the Real Time Parallel Benchmarks can foster the same kind of interest, the DoD will see the benefit in the form of increased attention paid to their applications by researchers and hardware designers.

3. Scope and Accomplishments of the project

We defined ten benchmarks for the RTPB suite - two application level benchmarks (Adaptive beamforming and Multiple hypothesis testing), three kernel level benchmarks, (associative gating, constant false alarm rate, and matched filter) and five low level benchmarks (three clock benchmarks and two communication benchmarks). For each benchmark we created a sample implementation in C using MPI for communication, and collected results on two or more platforms. The kernel and application level benchmarks represent topics of significant interest to the C3I community, namely image processing, image understanding, signal processing, filtering, pattern recognition, and classification problems. The low-level benchmarks are of interest to anyone who is implementing a real-time algorithm on a parallel machine. The details of the

benchmarks are discussed in their respective specification documents [1, 3, 5]. A summary of our accomplishments in this project appear below, with cross references to detailed reports.

- A document describing the rationale and criteria for benchmarks selection [8].
- Creation of five low-Level benchmark specifications [1]:
  gclock, rtcomm1, rtcomm2, lclock and lsclock.
- Creation of three-kernel level benchmark specifications [3]:
  CFAR, Associative Gating and Matched Filter.
- Creation of two application-level benchmark specifications [5]:
  Multiple Hypothesis Testing, and Adaptive Beamforming.
- Creation of a benchmark real-time parallel benchmarking methodology [7].
- Creation of a portable and reusable benchmarking harness to facilitate the real-time benchmarking of parallel machines efficiently.
- Creation of data sets, acceptance tests, and results for each of the 10 benchmarks.
- Creation of a portable sample implementation for each of the 10 benchmarks
- Performance results for each of the sample implementations for at least two different machines [2], [4], [6].

4. Lessons we learned about defining, implementing, and running benchmarks

In this section we share some of the lessons we learned from building and running benchmarks. While we primarily focus on the Real Time Parallel Benchmark Suite, many of the same issues were also seen in the creation of the C3I Parallel Benchmark Suite, and perhaps by other benchmark development teams as well. We hope that these lessons will be of benefit to others who create new benchmark suites in the future.

We do not provide results for any particular benchmark or machine, as that is not the purpose of this paper. Detailed observations on the results of each of the benchmarks are described in their respective reports [2,4,6].

Pen and Paper benchmarks are expensive to develop and implement

Pen and paper benchmarks are very flexible as they allow benchmark implementers to choose the best methods to implement the benchmark onto a specific machine. However, this flexibility comes at a cost. Benchmark specifications are expensive to develop, and new implementations are expensive to create. The decision to use Pen and Paper benchmarks or not is a philosophical one. We choose to use Pen and Paper benchmarks because we wanted to be as fair as possible to the variety of architectures used for embedded high performance computing. As standard and portable tools evolve (such as MPI, Open MP, libraries, and other API's) for embedded systems, it

may be possible to create “code” benchmarks, where the benchmarker need only run a standard piece of source code on any platform to gather results.

Selection of real-time parallel platforms to benchmark is limited

Most high performance computing centers provide access to platforms that are commercially available, and usually these centers’ computers have a workload made up mostly of scientific applications (CFD, computational chemistry, statics and dynamics modeling, etc). These applications are typically run in a batch mode from datasets stored on disk. Few of these machines have real-time operating systems installed (even though vendors may offer real-time environments, the majority of the users have no need for it). Similarly, most of the jobs on these machines are compute-bound, whereas C3I applications with real-time input often are input-bandwidth bound.

Unfortunately, the embedded systems vendors who sell machines with parallel real time environments were often unable or unwilling to allow us free access to their machines for the purposes of benchmarking. We found that these vendors see benchmark suites as an expensive burden and they feel trapped: either they let others benchmark their machines (whom they do not trust to represent their product as well as they could themselves) or they do the benchmarking themselves (at a considerable cost in labor). Therefore most vendors will not report benchmark results unless the vendor is required to when competing for a procurement. The result is that sample implementations get run on publicly accessible machines that may not have a real-time environment. In the end vendors may produce results for their machines, or otherwise independent researchers with access to these platforms may collect results.

It is our recommendation that future benchmarking projects secure funding to pay for time on vendor’s machines, or establish partnerships in the program to ensure access to these platforms.

There is a trade-off between portable examples and collecting the best results

Our benchmarks, being pen-and-paper benchmarks, allow the benchmarker considerable freedom in tailoring a solution to the architecture of interest. This can be a very time consuming task. Our approach was to make an example implementation that had reasonable performance and that was very portable. This way others could quickly get the benchmarks to run on a new machine, and have a starting point for optimization.

Because of our interest in portability the RTPB suite examples are provided in C and use MPI for communication. Had OpenMP been available when we were benchmarking we would certainly have done OpenMP implementations, as in some cases a true shared memory implementation may have been more efficient. Unfortunately we did not have the resources to do many custom shared-memory implementations and OpenMP was not available.

Our reusable benchmark harness proved to be a useful tool

At the beginning of our work we created a portable benchmark "harness". The harness provided many of the tools and services we needed for benchmarking platforms that were common to all of our real-time benchmarks. For example, the benchmark harness handled issues of global time, sending periodic input to the benchmark (simulating a real system), collecting results from the benchmark node, timing the computations, checking results, enforcing real-time deadlines, and increasing the input rate. We re-used the harness on each of our application and kernel level benchmarks. Not only did it save us time, it provides each of our sample implementations with a common interface which should be beneficial to those who work with the sample implementation source code. Future benchmark developers may want to consider using our benchmark harness, or building a similar tool to facilitate their development.

C3I applications have different characteristics than traditional scientific applications

As we became more familiar with the C3I (command, control, communications and intelligence) applications we had selected for the Real Time Benchmark Suite, and its predecessor the C3I Parallel Benchmark Suite, we realized that C3I applications were different from traditional scientific. For example, most scientific applications deal with double precision computations (CFD being one example), where most C3I applications do not (such as image processing). Data structures differ too. Many scientific applications work on structured matrices, while many C3I applications use prediction and classification algorithms (such as hypothesis testing and gating) and use data structures like lists and trees. Even the I/O characteristics are different - many scientific applications load in a data set, perform a set of computations, and produce a set of results. Most C3I applications take in data periodically, process it, and deliver the results just in time to receive a new input data.

Since most high-end computers are optimized for scientific applications (the typically workload of a parallel machine) there is a potential problem. The high-end COTS hardware that is readily available for scientific applications is probably not well suited for C3I applications. One can speculate that this may get worse in the future. Many research centers that traditionally have purchased high end parallel machines are investigating the use of PC clusters to run their applications. The side effect of this may be that the low-cost COTS hardware of the future is optimized for desktop applications and gaming.

The lesson to be learned is that not all applications are the same, and there is a price to pay for using commodity parts. This is, after all, why we have different benchmark suites to represent different classes of applications. In this case we have learned that C3I applications are sufficiently different from the application mixes found on a typical parallel machines. With the benchmark suite we can quickly evaluate how different architectures perform on C3I applications.

Most C3I applications require a considerable amount of memory

Many of the kernel and application level benchmarks require nearly 256 Megabytes of memory per node, and in some cases even more. In the Associative Gating benchmark [4] it was noted that 512 Megabytes or more per processor would be required for an efficient distributed memory implementation. We believe that many future C3I applications implemented on parallel machines may be memory constrained on typical machine configurations.

This suggests that an area open for investigation is to determine how sensitive the class of C3I applications is to changes in the memory architecture. Our experience with benchmarking has shown that C3I applications differ from the typical scientific applications used in most parallel benchmarks. Since these more common benchmarks influence procurements and design decisions, it is possible that modern architectures are diverging from the needs of C3I applications. An investigation of the memory needs of C3I applications may influence system design decisions in the future.

Scalability may be limited by factors other than aggregate processing power

Many of our benchmarks increase the rate of input to stress the machine's ability to meet deadlines and establish a breakdown point. When a breakdown point is found for a set of processors, the number of processors is increased and the experiment is performed again. If the benchmark implementation is scalable it is expected that the sustainable input rate would double as the number of processors is doubled. This conventional wisdom does not always hold true as it assumes the reason for a deadline miss is due to insufficient compute resources, and that adding more processors will allow for a shorter deadline. The speed at which input can be processed consistently without missing a deadline can be dependent on many other factors, including bisection bandwidth of the inner-connect and latency for communication. These resources must also scale as the number of processors increases. Furthermore, the number and duration of interruptions and delays introduced by the operating system services must continue to be reduced when using more and more processors to achieve perfect real-time scalability. The opposite is more likely to happen, as the OS must manage communication channels with more and more processors more interruptions are introduced. The result is that a benchmark with a deadline may not scale well even though the computations scale well. This is an accurate reflection of how the system will perform on real applications when deadlines are present.

Acceptance tests for real-time behavior must be carefully constructed

Acceptance tests are used in the RTPB suite to determine if a benchmark is meeting real-time deadlines. Most of our benchmarks receive periodic input and deliver periodic output, with the requirement that the output must be delivered before the next input arrives. Accurately testing this is more challenging than it may seem. The test must take into account measurement errors, granularity of the clock, clock skew and

make measurements on the appropriate processor. In some cases the input arrives from a different processor than the output in sent. In other benchmarks the processor to measure is the one that takes the longest time to complete, which may change from iteration to iteration. To address all of these issues we provide a benchmarking methodology document [7].

The importance of putting a benchmark in context to other tasks

When defining a benchmark it is important that its definition be faithful to the real world application it represents. This requires taking careful definition of the input and output requirements. Some questions that need to be considered are, "in the real world is the data read from disk, delivered from another process, or already in memory?", "should we start timing after the data is available on all processors, or when the first processor has the data available?", "do we end timing when the computations are ready and the results are finished, or do we need to include the time necessary to send the results on to the next computation?" "do our timing assumptions change with different architectures, does a shared memory implementation imply something different than a distributed memory implementation?" By addressing these issues a benchmark developer can help ensure that the benchmark is a realistic representation of the real world application.

The challenges in choosing input data

Choosing the input data and the method by which it is delivered to the benchmark is a very important task. We went to great lengths to make sure that the data we were using for the benchmark was representative of the real application. This is not always an easy task. Some computations are data dependent, so in order to exercise a benchmark implementation fairly several different data sets need to be run, which may require a considerable amount of input data and run time. Choosing what these data sets are can be time consuming part of benchmark development.

Another challenge can be the size of the input. Some data sets can be 100's of Megabytes, which makes the distribution of the benchmarks cumbersome. Our solution to the size problem was to create synthetic data set generators; small programs that the user could compile and run at their site to create a specific data set locally.

Deciding how the benchmark will receive the data is perhaps the biggest challenge. Because of the nature of the applications we were representing, most of our benchmarks received input periodically as messages from a processor outside of the computation. This was done to simulate the "real world" where a sensor (a radar for example) would be sending an embedded system data for processing. Managing the transmission of the data was particularly challenging, as the requirements changed from benchmark to benchmark. In some cases the data was broadcast, in others it was sent to a single processor. For some benchmarks the data was being generated by the sending processor, in other cases it was read from disk. Care had to be taken to make sure that the sending processor sent the data at exactly the right time. If the data was

delayed the benchmark implementation might miss a deadline through no fault of its own. To make this task more manageable we implemented these tasks in our reusable benchmark harness.

Finally, it is important to remember that different platforms have different representations for data types. Binary input files will fail on some systems if there is an assumption of byte order or of the size of a datatype (e.g. 32 bit vs. 64 bit).

Different clocks on the same system can give dramatically different results

For a benchmark, timing is everything. On a modern machine a programmer has access to many different functions that can be used for timing. Different clocks have different overhead. For example, one MPI implementation tested on a Mercury had a MPI_Wtime() call that gave considerably worse results than the Mercury's hardware clock. Similarly, we found that the SGI special hardware clock reading routines gave much better results than any other SGI operating system level clock calls. It is advisable that benchmarks and programmers familiarize themselves with all the clock options on a platform and select the clock that gives them the best granularity possible with the least overhead.

Machines that run time synchronization protocols can cause bad measurements

We found that machines that run clock synchronization protocols such as NTP can cause bad measurements in benchmarks. These protocols can cause clocks to "jump" to synchronize, which hinders the accurate timing of a benchmark, and leads to unreliable results. Our benchmark harness tests for this type of behavior, and reports it if detected. It is up to the user running the benchmarks to make arrangements to have the NTP service turned off, find a similar platform without NTP, or re-run the benchmark until it completes without clock adjustments during the benchmark execution.

Benchmarks are useful even to those who do not want to do benchmarking

Several other researchers have requested our benchmarks as sample applications to test their systems and subsystems. While this is an unexpected use of the benchmarks, it indicates that the benchmarks are fulfilling a need that was previously unnoticed, namely researchers desire freely accessible C3I applications to demonstrate and test their work. DARPA and other agencies that want to promote research in a particular domain may want to consider the benefit of making representative applications available to researchers. This may encourage more work in the agency's area of interest.

How to guard against benchmark obsolescence

Benchmarks grow old. What takes hours to run today will only take a few minutes to run tomorrow. The good news is that data sets are growing as fast (if not faster) than our computational ability, and that the necessary response time (which often determines real-time requirements) is becoming shorter and shorter. For this reason we have tried to make it possible to scale our benchmarks so that they can represent tomorrows problems as when today's problems are no longer challenging. For the RTPB suite this can be done in two ways: increasing the amount of data to be processed, or creasing deadlines. Furthermore we have created our benchmark methodology to provide guidelines for adding new benchmarks to the RTPB suite. We encourage other benchmark suite developers to consider how they may structure their benchmarks so that they can remain relevant as long as possible.

5. Conclusions

We have developed a real-time parallel benchmark suite for DoD applications, a first of its kind. The Real Time Parallel Benchmark suite fulfills a need and is complementary to work performed at MITRE [11, 14], and the C3I Parallel Benchmark Suite [12, 13] developed at Honeywell. Together these three projects have produced over twenty-five benchmarks for the C3I community. These benchmarks have the potential to assist future system designers in making decisions about the hardware and software they use. They can also be of benefit to other researchers who wish to use them for their own experimentation.

In creating these benchmarks we have identified many important issues that we hope can be of benefit to future benchmark developers.

6. Acknowledgments

We would like to thank the following computing centers for providing their resources: NASA Ames Research Center, Arctic Region Supercomputer Center, Maui High Performance Computing Center, University of Minnesota, Mississippi State University.

We would also like to thank the following individuals for their help and guidance: Ralph Kohler at AFRL, Jose Munoz at DARPA, Robert Bernecky at NRAD, Vipin Kumar at the University of Minnesota, Benoit Champagne at the University of Quebec, Richard Games at MITRE, Craig Lund at Mercury Computer Systems, Glen Ladd and Rob Moore at Hughes Research Labs, and David Bailey at the NASA Ames Research Center.

7. References

1. “Low-Level Benchmark Specifications”, Technical Information Report (C006), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
2. “Low-Level Benchmark Results”, Technical Information Report (C007), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
3. “Kernel Level Benchmark Specifications”, Technical Information Report (C008), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
4. “Kernel-Level Benchmark Results”, Technical Information Report (C009), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
5. “Application Level Benchmark Specifications”, Technical Information Report (C0010), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
6. “Application Level Benchmark Results”, Technical Information Report (C0011), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
7. “Benchmarking Methodology”, Technical Information Report (C005), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
8. “Benchmark Requirements and Selection”, Technical Information Report (C004), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
9. “Benchmark Distribution Plan Report”, Technical Information Report (C012), Rome Laboratory, Contract Number F30602-94-C-0084, August 1998
10. “A Real-Time Parallel Benchmark Suite”, B. VanVoorst, L. Pires, R. Jha, Proceedings of SIAM’s Parallel Processing for Scientific Computing, SIAM, March 1997.
11. “Real-Time Embedded High Performance Computing: Application Benchmarks”, C. Brown, M. Flanzbaum, R. Games, J. Ramsdell, MITRE Technical Report, MTR 94B0000145, October 1994.
12. “C3I Parallel Benchmark Suite Final Report”, Technical Information Report (A012), Rome Laboratory, Contract Number F30602-94-C-0084, May 1998
13. “The C3I Parallel Benchmark Suite - Introduction and Preliminary Results”, R. Metzger, B. VanVoorst, L. Pires, R. Jha, W. Au, M. Amin, D. Castanon, V. Kumar, Proceedings from Supercomputing 96, 1996.
14. “Benchmarking Methodology for Real-Time Embedded Scalable High Performance Computing”, R. Games, MITRE Technical Report, MTR96B0000010, March 1996.
15. “NAS Parallel Benchmark (Version 1.0) Results 11-96”, Subhash Saini, David Bailey, NASA Ames Technical Report NAS-96-18, November 1996.