IS-M--860

IS-M 860

CONF-961104--4

# An Analytical Model of the HINT Performance Metric

*Quinn O. Snell*
*Scalable Computing Laboratory*
*236 Wilhelm Hall, Ames, IA 50011*
snell@ameslab.gov
http://www.scl.ameslab.gov/Personnel/quinn.html

*John L. Gustafson*
*Scalable Computing Laboratory*
*236 Wilhelm Hall, Ames, IA 50011*
gus@ameslab.gov
http://www.scl.ameslab.gov/Personnel/john.html

**Abstract**

The HINT benchmark was developed to provide a broad-spectrum metric for computers and to measure performance over the full range of memory sizes and time scales. We have extended our understanding of *why* HINT performance curves look the way they do and can now predict the curves using an analytical model based on simple hardware specifications as input parameters. Conversely, by fitting the experimental curves with the analytical model, hardware specifications such as memory performance can be inferred to provide insight into the nature of a given computer system.

**Keywords**

computer performance, model, benchmark, supercomputer.

This work was supported by the Applied Mathematical Sciences Program of the Ames Laboratory, U.S. Department of Energy under contract number W-7405-ENG-82.

## 1 Introduction

The HINT [1,2] performance metric was developed at Ames Laboratory to gauge the overall performance of a wide variety of computing machines. Unlike other benchmarks which determine the amount of work to be done *a priori*, HINT fixes neither the problem size nor the execution time of the problem to be solved. Consequently, it measures the performance of a computer across all memory regimes. The output of a HINT performance measurement produces a graph which is a rigorous measurement of the QUality Improvement Per Second (QUIPS) in an answer. The graph reveals the performance range of the tested computer from *burst speed* for very small problems to *endurance speed* for large problems that may have to use mass storage.

HINT also consolidates precision, memory size, memory speed, and arithmetic speed into a single figure of merit. That metric, the Net QUIPS, is the area under the HINT graph. More memory in a given regime keeps the graph on a plateau, thus increasing the Net QUIPS. If a computer has insufficient

8

memory, slow instruction execution, long pipeline startup, a "memory hog" operating system, low arithmetic precision, or other design shortcomings, some part of the HINT graph will fall and the Net QUIPS will be reduced.

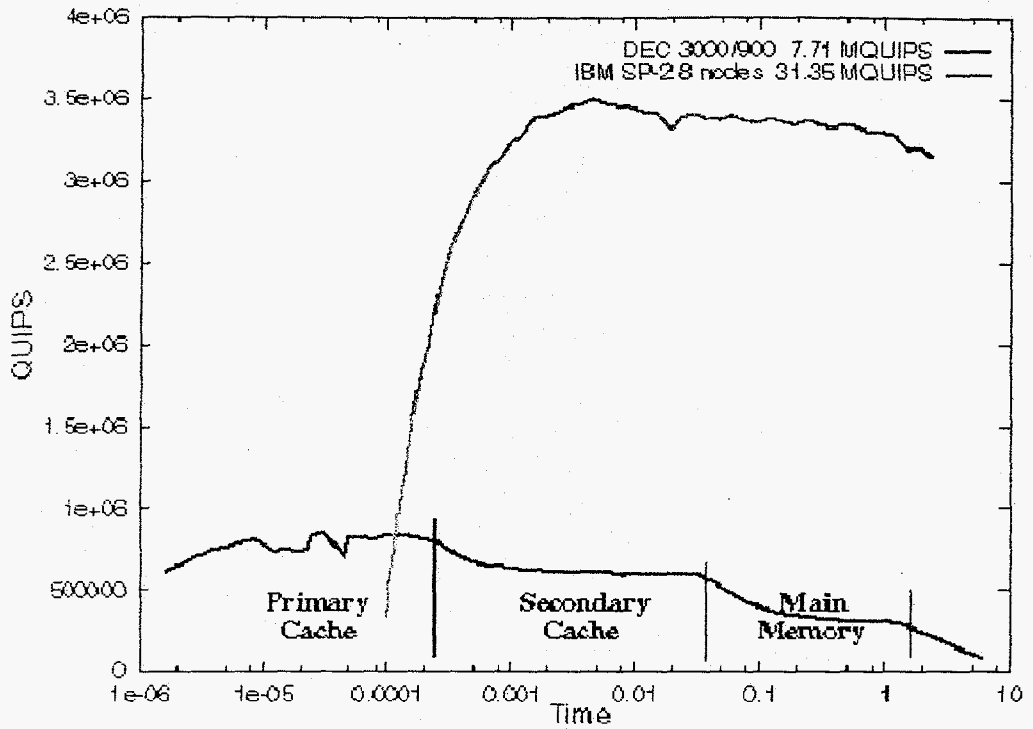

**Figure 1:** Example HINT graph of a DEC 3000/900 versus an IBM SP-2

Any digital computer, from a person using an abacus to the largest parallel supercomputer, can be measured and accurately compared using HINT. Comparisons across different architectures are valid and the differences are readily visible using the HINT graphs. Figure 1 demonstrates this by comparing a DEC 3000/900 workstation to the combined power of eight nodes of an IBM SP-2. Note that while the workstation has a quick startup time, once the parallel overhead is overcome, the SP-2 far outperforms the workstation. The graph also shows the three performance levels of the DEC workstation. These levels correspond directly to the memory hierarchy. This particular workstation has 16 Kbytes of primary data cache, 2 Mbytes of secondary unified cache, and 256 Mbytes of main memory. Clearly, the HINT graph gives a visualization of computer performance and architecture differences previously unavailable.

Although HINT has been widely accepted and seen great success in predicting the performance of existing computer systems, it has not been used in the design of new computers. HINT performance knowledge is based on a prior run of the benchmark. The performance analysis cannot be run on a computer that does not exist outside of the mind of the computer designer. Therefore, it is difficult to incorporate HINT into the design process.

An analytical model, described here, allows performance prediction based on a small set of design statistics. It presents the computer architect with a powerful tool to build a balanced computer by varying design parameters based on cost and performance. The model also benefits users of existing computers. Consider a computer user trying to determine whether or not to upgrade from 128 K secondary cache to 512 K secondary cache. Advertisements indicate 'huge' performance increases. One could run HINT on the existing configuration, match the QUIPS curve using the analytical model to determine the input statistics, and then increase the secondary cache size input to the analytical model which would then predict the performance increase. At this point, an educated decision could be made based on cost increase versus performance increase. By fitting the model to existing HINT graphs, a user can find out the true computer performance parameters and compare them against manufacturer claims. For example, in one case we discovered a so-called "secondary data cache" was nothing of the kind, and served only as an extra memory for instruction storage.

This paper focuses on the design of an analytical model of the HINT performance metric called AHINT. The model allows the user to enter design statistics such as processor speed, memory regime sizes and speeds, number of processors, and communication latency for parallel and distributed computer systems. Section 2 presents the model and describes the approach taken to calculate the execution time of the HINT kernel. The results in Sections 3 and 4 present the use of AHINT in matching existing computer systems. Finally, conclusions are presented in Section 5.

## 2 Model Description

HINT tests a computer's performance by rigorously bounding the area under the curve $f(x)=(1-x)/(1+x)$ in the range $[0,1]$ using hierarchical integration. This is accomplished by dividing the largest remaining rectangular error in half and determining the area now known to be above and below the curve. After each division, knowledge is gained and quality is improved. The term *quality* is defined as the reciprocal of error, where error is the difference between the upper and lower bounds of the area under the curve. See [1] for details. If there were no precision loss, quality and the number of iterations would be identical, so it is a linear work measure.

The analytical model presented here is based on the assumption that execution time is the time to execute the instructions and the overhead associated with fetching memory that is not immediately available in the primary processor cache. Clearly, this is a high-level view of performance; further clarification is needed for each area.

Instruction execution time is based on the number of each instruction type, the amount of time to execute that type, and in some cases execution order. In order to avoid the complexity associated with this, each instruction is assumed to execute in the average number of cycles per instruction (CPI). Therefore, instruction execution time, in processor cycles, is equal to the CPI [3] times the number of instructions.

Memory fetch overhead is a much more complicated issue. AHINT assumes that all the data needed for the instructions are in the primary cache. This is not always the case. Therefore, the amount of time to copy the data into primary cache must be calculated based on the current location of the data. The model assumes the principle of *inclusion*; all the data in a nearby level of memory are included in the distant levels.

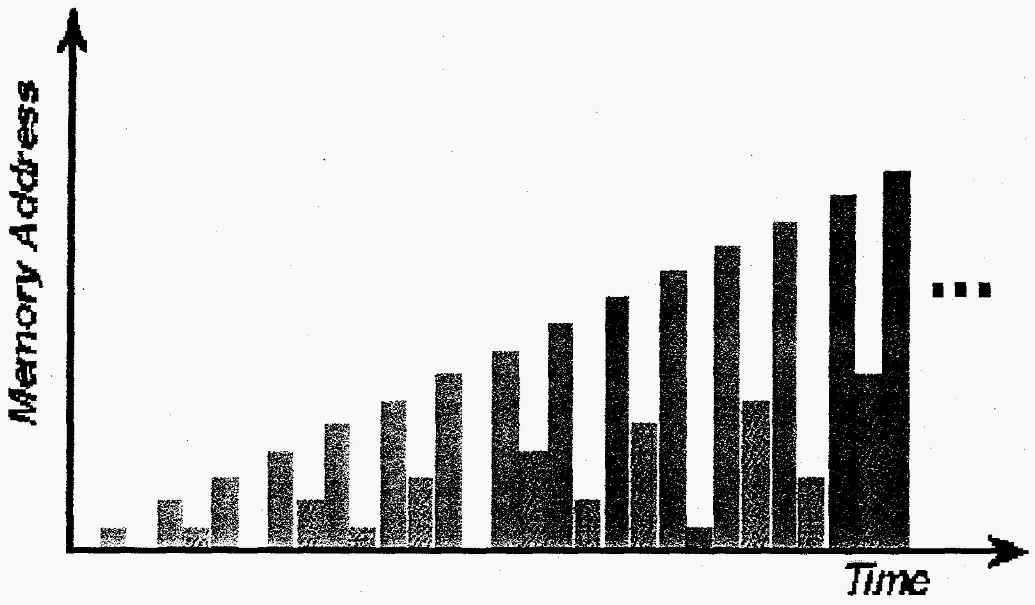

**Figure 2:** HINT memory access pattern

The total number of blocks accessed by HINT increases linearly with the number of kernel iterations. However, the access pattern during the iterations is not linear. It is *fractal* in nature, as shown in Figure 2. Every other access address is linearly increasing. Of the accesses remaining, every other access address is linearly increasing at half the rate, and the pattern continues. The effect of the above pattern is that once cache size is exceeded, the block to be accessed is replaced just before it is needed. This forces a cache miss and the memory block must be reloaded into the cache from the next lower memory regime. This cache miss and the time to load the cache is the memory fetch overhead that must be calculated. Also, since every other memory access is linearly increasing, one half of the memory accesses will not be in primary cache.

The calculation of memory access overhead can best be explained through the following example: Consider a hypothetical computer system with three memory regimes. The primary cache is 16 Kbytes with a miss penalty of 3 cycles, secondary cache is 1 Mbyte, and main memory size is 64 Mbytes with latency 70 ns. Further suppose that the processor speed is 150 MHz and the average CPI is 1.8. For a full list and discussion of the input parameters and their effect on system performance, see section 5. Given this configuration, the time to execute $10^5$ iterations can be calculated as follows:

First, calculate the time to execute the instructions. This is simply the number of instructions multiplied by the number of iterations times the average CPI. In most RISC systems, HINT averages 200 instructions per iteration. Therefore, instruction time in cycles is

$$200 \text{ instructions per iteration} * 10^5 \text{ iterations} * 1.8 \text{ CPI} = 3.6 \times 10^7 \text{ cycles.}$$

The total number of data blocks accessed by HINT grows linearly with the number of iterations. Two data blocks are accessed during each subdivision (one data block is added and one is reused) yielding $2 \times 10^5$ block accesses. Each data block contains information describing one subinterval. The necessary data is ten numbers using the computation data type plus one number of the index data type. Typically, computation data is 64-bit IEEE floating point and indexing is done with 32-bit integers, but this is flexible. For a HINT run using 64-bit IEEE floating point numbers, the resulting data block size is 84 bytes. Since the primary cache is 16 Kbytes, it would hold approximately 195 data blocks, accounting for 390 data block accesses. Due to HINT's memory access pattern, half of these accesses will miss. The remaining 195 block accesses are in primary cache so there is no overhead associated with accessing these blocks.

The secondary cache is 1 Mbyte and thus could hold approximately 12483 data blocks, but inclusion must be considered, yielding 12288 data blocks. Again, of the 24576 accesses at this level, one half will miss cache. The time for accessing blocks at this level is the miss penalty of the primary cache plus the access time of this level. For our example, this is 3 + 1 = 4 cycles. The overhead associated with these accesses is simply

$$12288 \text{ data block accesses} * 21 \text{ words per block} * 4 \text{ cycles per access} = 1.03 \times 10^6 \text{ cycles.}$$

Finally, the remaining $1.87 \times 10^5$data blocks accesses must all be loaded from main memory. If memory speed is 70 ns, then access time is 10.5 cycles plus 4 cycles miss overhead from the previous levels. Therefore, the remaining data block accesses account for

$$1.87 \times 10^5 \text{ block accesses} * 21 \text{ words per block} * 14.5 \text{ cycles per access} = 5.7 \times 10^7 \text{ cycles.}$$

The total time to execute $10^5$ iterations is

$$3.6 \times 10^7 \text{ cycles for instructions} + 5.8 \times 10^7 \text{ total cycles memory overhead} = 9.4 \times 10^7 \text{ cycles.}$$

For a 150 MHz processor, this is 0.6275 seconds. The total time for execution must also account for function call overhead. From experimentation, a machine of this caliber typically has a function call overhead of 350 ns which is only significant for the extreme left portion of the HINT curve.

With no precision loss, each subdivision improves the quality of the answer by one unit. The computer is using finite-precision arithmetic however, so there is a loss in quality due to discretization error. The quality at subdivision *i* can be determined as follows

$$Q = i - (i * (1 - scy / (scy + i - 1)))$$

where *scy* is the number of area units in the vertical coordinate axis for the function. It is generally the largest whole number expressible with half the available bits for data representation (mantissa only, if a floating point type is used). Using 64-bit IEEE floating point arithmetic, *scy* is 134217728. For the example, quality therefore does not increase perfectly linearly to 100000 but results in 99925.55 at this point. A computer is allowed to start with the knowledge that the area under the curve is between 0 and 1; quality is therefore 1/(1 - 0) = 1. QUIPS is change in quality divided by time. The resulting QUIPS for the hypothetical machine is

$$delta\ Q \,/\, \text{time} = (99925.55 - 1)/0.6275 = 0.159 \text{ MQUIPS.}$$

## 3 Results

In this section, the accuracy of the model is determined by comparing the output of the model with known results. All comparisons are based on previous runs of the HINT benchmark and parameters obtained using technical references [7,8,9] for the given systems. The number of hand-tuned parameters is surprisingly limited, and will be discussed in Section 5. This section will focus on four systems: an SGI Indy pc, an SGI Indy sc, a DEC 3000/900, and an SGI Power Onyx. The SGI Indy systems are based on the MIPS R4600 processor, differing only in the amount of secondary cache; the pc version has none, while the sc version has 512 Kbytes providing a controlled experiment in cache modeling. The DEC 3000/900 uses the Alpha processor and the Power Onyx uses the MIPS R8000 chip set. The main difference between the Alpha chip and the R8000 is the pipelined instruction execution of the R8000.

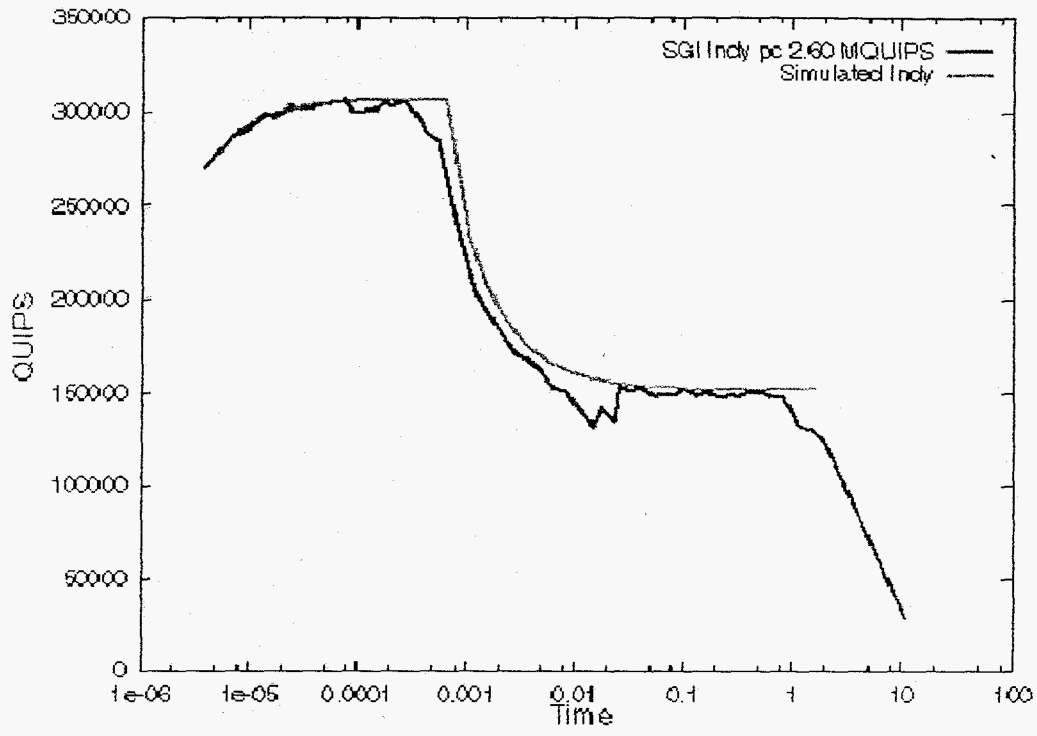

**Figure 3:** Actual and simulated SGI Indy pc

Most of the needed statistics for the SGI Indys were found in the technical reference for the R4600 processor. Like most processors, the R4600 can access a data element in on-chip cache in a single clock cycle. The cache miss overhead is three clock cycles. Figure 3 shows the data from an actual run of HINT compared to the analytical model results for the Indy pc. Note that while the simulated curve is smooth, the curve for the actual HINT run has noise. This effect is caused by operating system interrupts that steal cycles from HINT. The last drop in performance is due to HINT running out of real memory and using virtual memory paged in from mass storage. Currently, AHINT does not simulate this regime as it adds very little to the net performance of the system.

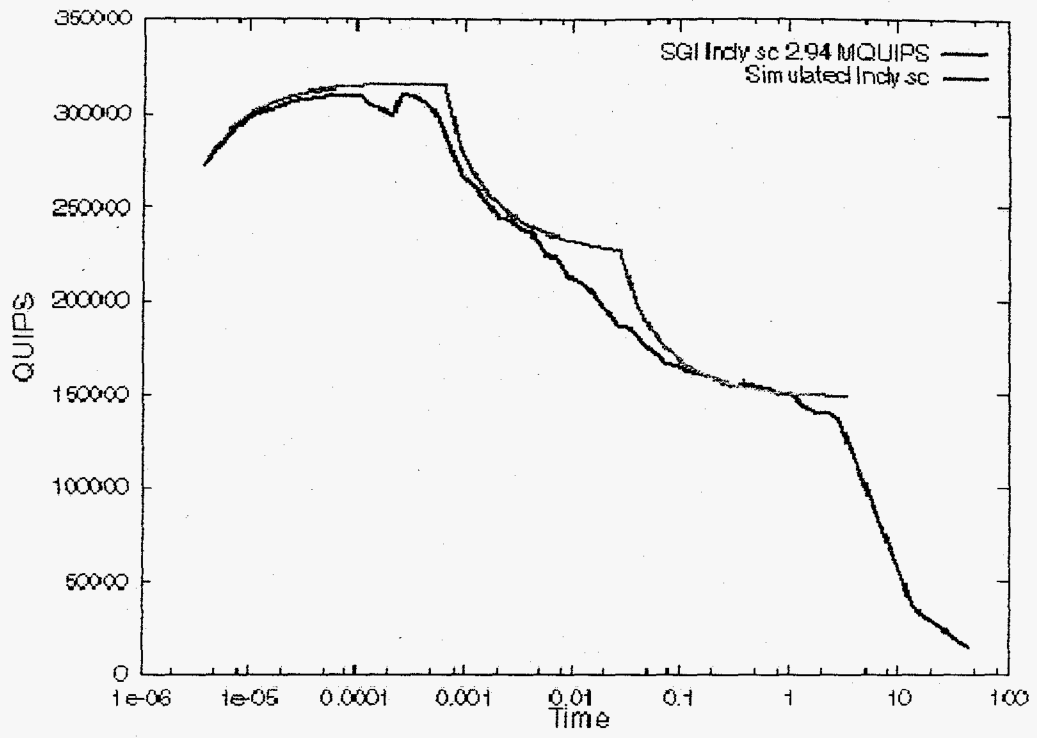

**Figure 4:** Actual and simulated SGI Indy sc

Figure 4 depicts the equivalent data for the SGI Indy sc. An interesting point to note on the graph is the elongated performance plateau for the simulated results. Why is there a mismatch in the range from around 0.002 to 0.1 seconds? The plateau corresponds directly to the secondary cache size. On the Indy sc, the secondary cache is a unified instruction and data cache; not all of the cache is available for HINT data. The extra performance in the simulated curve can be eliminated by reducing the size of the modeled secondary cache, thus reflecting the percentage of space available for HINT data. In this way, the *usable* memory sizes can be determined for an existing machine. Because of operating system demands, the available memory is often much less than the nominal memory; HINT and AHINT quantify this discrepancy precisely and explicitly.

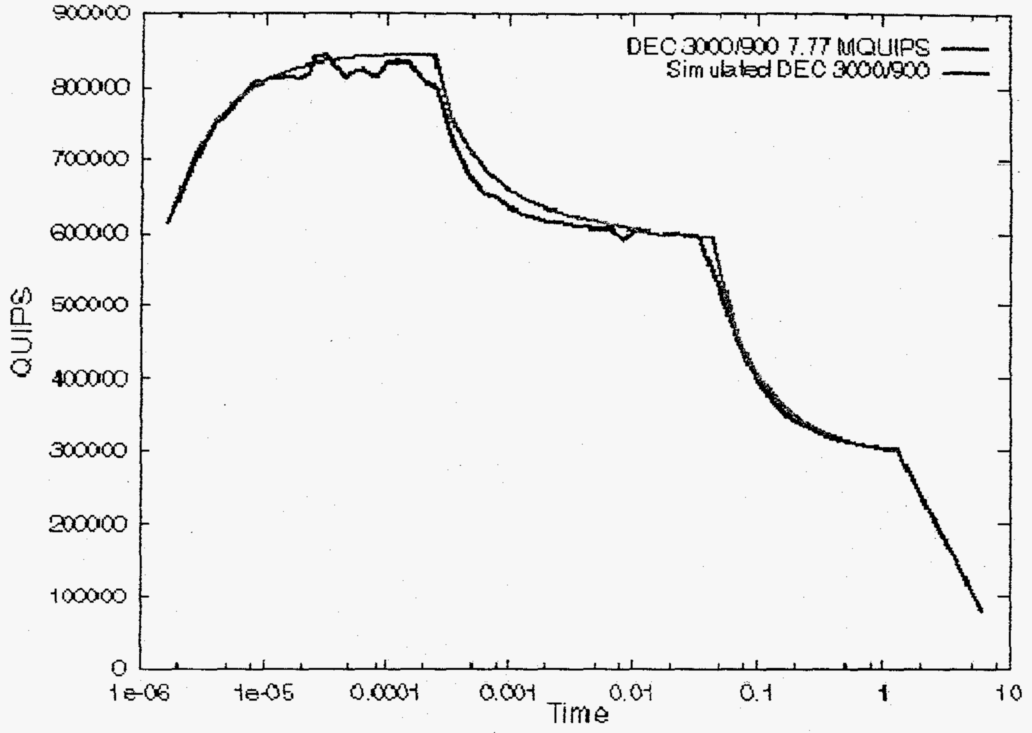

**Figure 5:** Actual and simulated DEC 3000/900

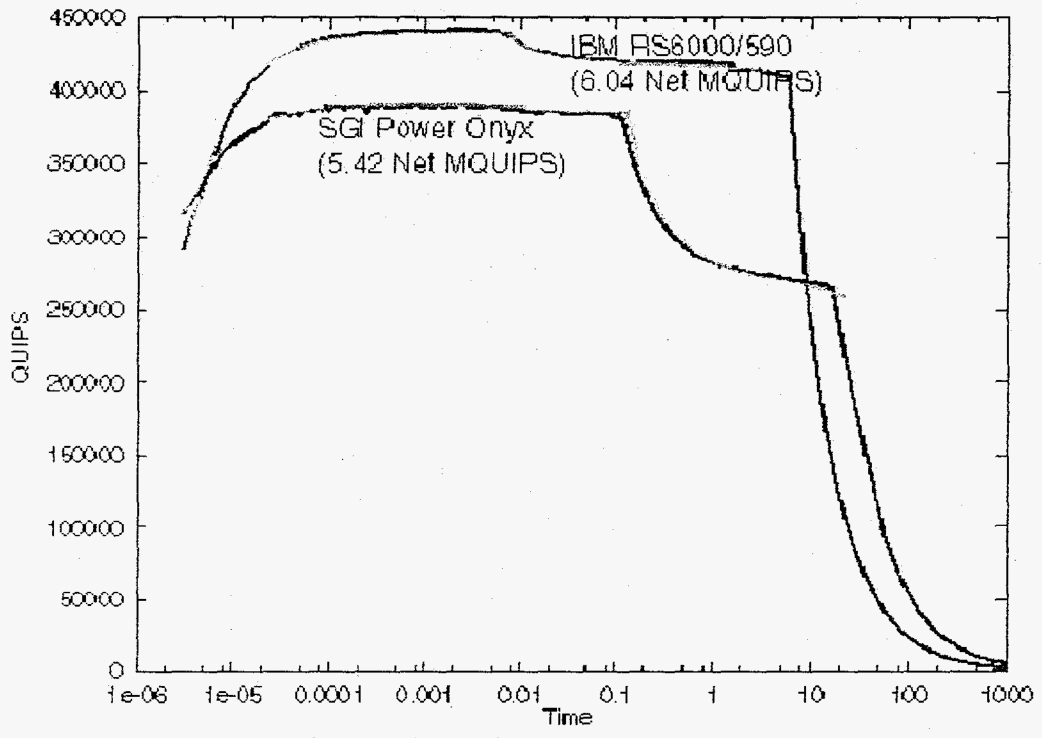

**Figure 6:** Actual and simulated SGI Power Onyx

Figure 5 shows the comparative results for the DEC 3000/900. The processor used in this machine is similar to that used in the CRAY T3D. It would be interesting to compare this with an actual T3D node, particularly at the point where shared virtual memory is invoked. The differences between the SGI Power Onyx and the IBM RS6000/590 are displayed in Figure 6. In this and many graphs that follow, AHINT is such a close match that the experimental measurements are covered up by the width of the analytic curve plot. To aid in differentiating the curves, actual HINT results are plotted in blue, while AHINT simulation results are in red. The Power Onyx runs at 75 MHz while the IBM runs at 67 MHz. Each has the ability to execute up to 4 multiply-adds in a single clock cycle. The actual HINT performance is lower, resulting in approximately 0.95 and 0.75 average cycles per instruction, respectively. Notice that the relative performance drop between cache and main memory is much smaller for the IBM. This is a result of hiding the memory fetch overhead in the pipeline. It was for this machine that the parameter indicating the percentage of hidden memory fetches was added to AHINT. Emphasis on multiply-add parallelism is the result of market pressure to perform well on LINPACK[10] and related arithmetic kernels. We feel emphasis on memory speed and size is now more essential to high performance on actual applications. The emphasis on memory access is clearly seen in the IBM curve while the SGI takes a deep hit when retrieving data from main memory.

## 4 Modeling Supercomputers

HINT uses domain decomposition to distribute the function domain among the processors of a supercomputer. Each processor can calculate a scattered portion of the domain and then it is free to compute the area bounds for that subpartition. Global knowledge of the area bounds is obtained via a global sum collapse of each processor's upper and lower bounds. The time for this sum collapse must be accounted for in the current analytical model. Since communication overlap with processing is not possible in this case, the new model is simply

$$\text{Execution Time} = \text{Serial Execution Time} + \text{Communication Time}$$

Since there are only two double-precision floating-point numbers per processor, all messages are latency bound. In most parallel supercomputers, the global sum is performed via an $O(\log n)$ collapse, where $n$ is the number of processors. Communication time becomes log $n$ messages multiplied by the message latency. There is a subtle additional time, however. On some systems, after the message is sent, there is some clean-up of system-maintained variables and queues. In a collapse involving more than one pairwise combine step, the clean-up time for the lower levels of the tree can occur in parallel with the remaining message passing. The processors involved in the last level of the collapse are unable to *hide* the clean-up time, leaving that time to be added into the communication time as a system overhead.

$$\text{Communication Time} = \log n * \text{Message Latency} + \text{Clean-up Time}$$

Based on this model, Figures 7 - 10 show the comparison of actual HINT results with AHINT simulated results for an nCUBE 2S, an IBM SP-2, an SGI Power Onyx, and a cluster of SGI Indy pc workstations. In all cases, the serial model was used to match the curves for a single processor, after which the parallel model was applied for each configuration. Note that the vertical scale is logarithmic here, for clarity.

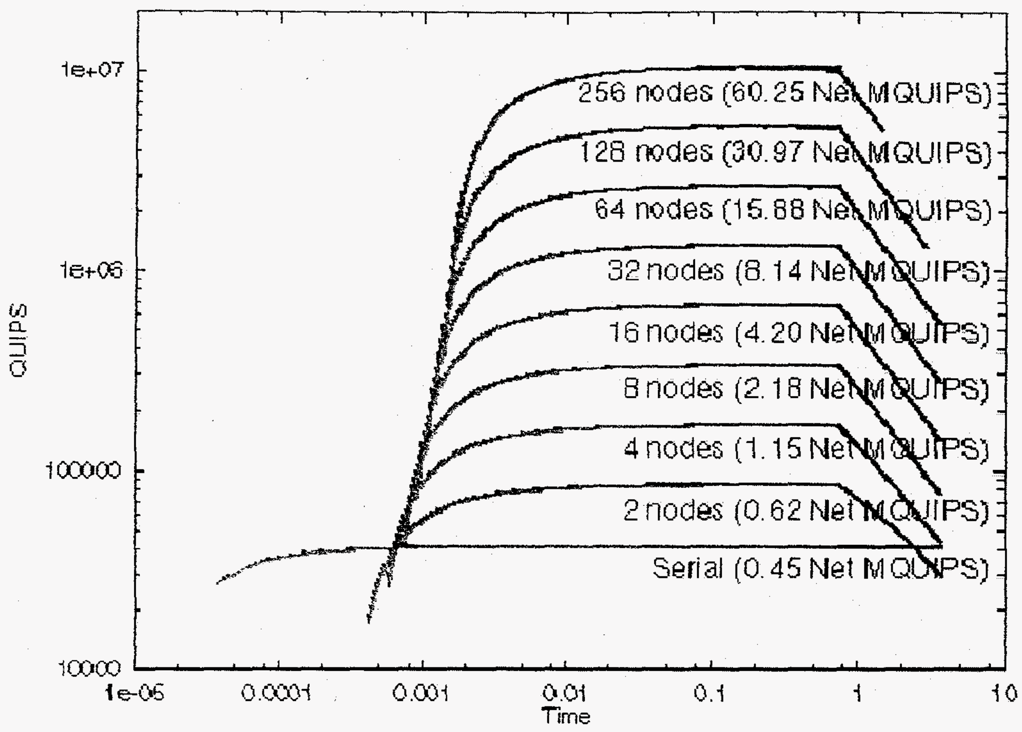

**Figure 7:** Actual and simulated nCUBE 2s

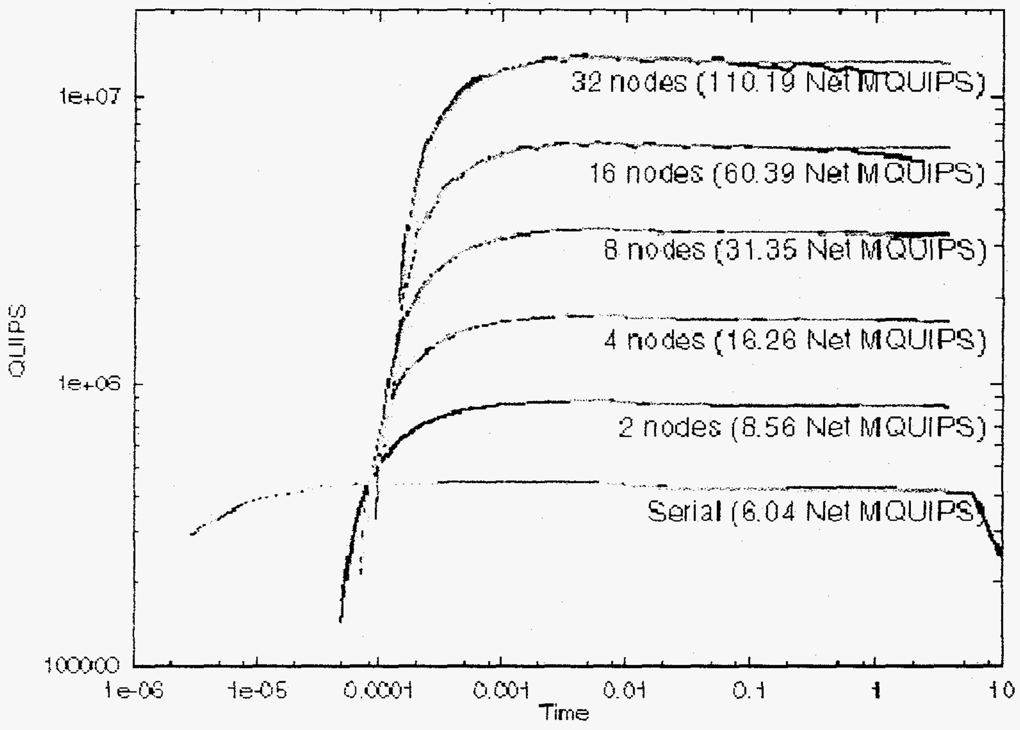

**Figure 8:** Actual and simulated IBM SP-2

The curves represented in Figures 7 and 8 show the efficacy of the model in estimating the performance of a parallel supercomputer. Once the performance of a single processor is known, the only remaining statistics are the message latency and message overhead. The message latency can be found in technical references, but the message overhead must be experimentally obtained. This was accomplished by adjusting the overhead until the curves matched for two processors and the maximum number of processors. Once these two curves fit, all the other configurations matched with no further adjustments necessary.

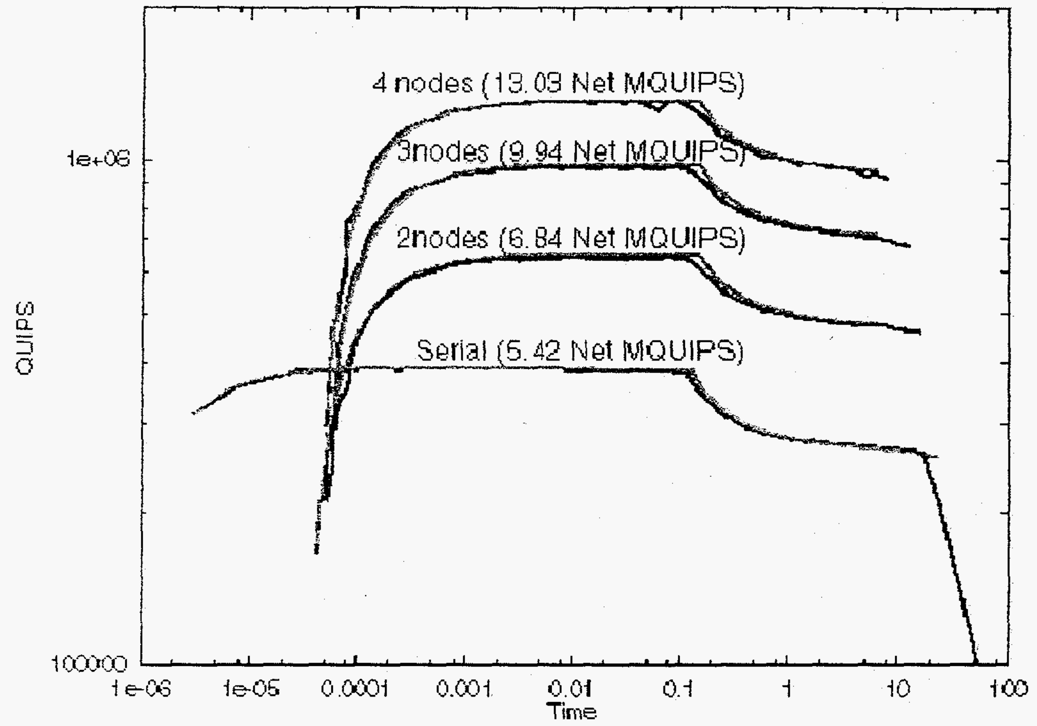

**Figure 9:** Actual and simulated SGI Power Onyx

The data in Figure 9 depict the results from matching AHINT results with actual HINT results for an SGI Power Onyx. When modeling a system such as this, shared memory contention must be considered. This effect was accounted for using the response time equations for an M/D/1 queue with an input rate equal to *n* times the input rate of a single processor, where *n* is the number of processors[4]. However, due to the large caches on the SGI and the eight-way interleaved memory, the memory contention was non-existent and thus had to be set to zero.

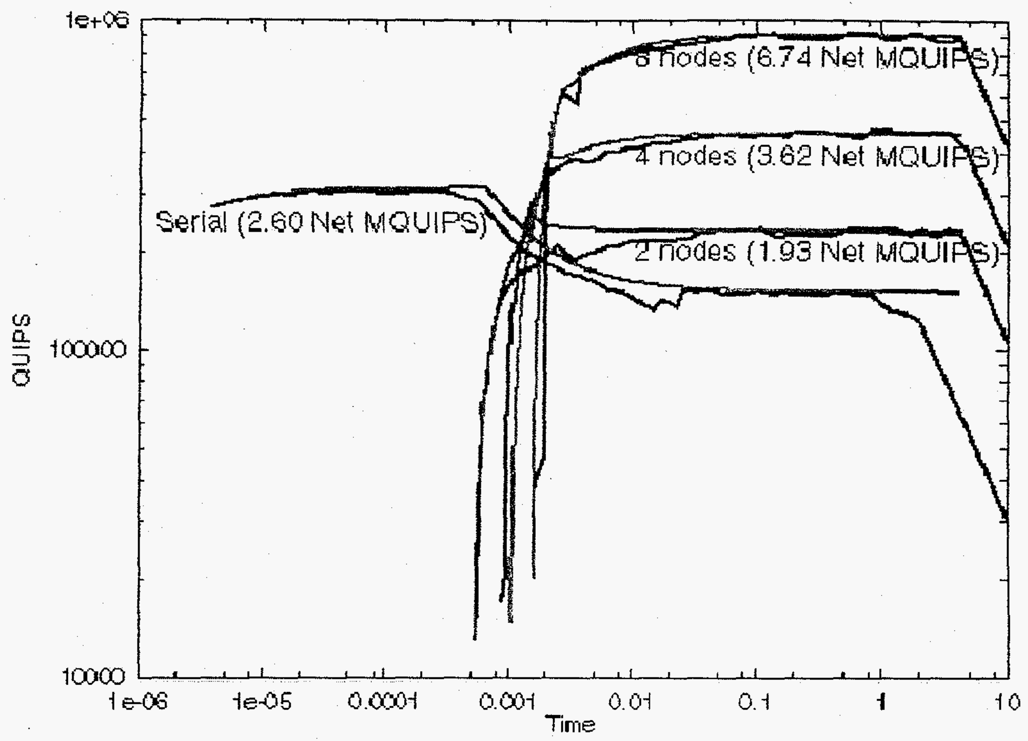

**Figure 10:** Actual and simulated SGI Indy pc workstation cluster

A recent trend is the use of workstation clusters and a high-speed network for parallel computing. While this provides a lower-cost solution than a supercomputer, there are drawbacks. The unpredictable nature of the network traffic and network contention make this a particularly difficult system to model. Figure 10 shows the results of attempting to model a cluster of SGI Indy workstations connected by an ATM network using MPI [11] as the communication infrastructure. Previous research at Ames Laboratory resulted in a network analyzer called NetPIPE [5,6]. This tool reported that the effective message latency for ATM is between 0.7 and 0.8 milliseconds. While this parameter was easily determined, clearly there are other factors that must be considered in this case. The initial points on the curve are closely matched by the model for 2 and 8 nodes, but unlike the previous results, using the same data for 4 nodes does not result in a match of the actual curve. Although the model is not as accurate in this case, it is still within five percent and does lend some insight. As is well known, the limitation of a workstation cluster is the high message latency. Also note that the graph clearly shows the result of the high latency. For programs with computation loop time less than 1 ms, a cluster of this nature should not be used. The performance for a serial processor is much higher in this case.

A major issue in heterogeneous computing is how to measure and compare heterogeneous cluster performance.[12] AHINT and HINT solve this problem by providing a broad-spectrum measure that ports easily to heterogeneous clusters. A heterogeneous measurement with HINT was done on the I-WAY and was described in [13]. AHINT can also be easily extended to simulate heterogeneous clusters of computers.

## 5 HINT Curve Fitting

While most of the system parameters used by AHINT are obtainable from technical manuals, some cannot be determined accurately *a priori*. The previous results were generated by gathering as many of the system parameters as possible from technical specifications. The remaining AHINT input parameters were derived through a manual curve fitting process.

**Table 1:** AHINT Input Parameters

| Parameter | | | Units | How Determined |
|---|---|---|---|---|
| Processor Speed | | | MHz | Technical Specification |
| Number of Instructions | | | | Given; ~ 200 |
| Wordsize | | | bytes | Technical Specification |
| HINT Data Block Size | | | bytes | Derived from Data Type |
| Number of Cache Regimes | | | | Technical Specifications |
| | Regime 1 | Size | Kbytes | Technical Specifications |
| | | Access Time | cycles | |
| | | Line Size | bytes | |
| | | Miss Penalty | cycles | |
| | ....... | | | |
| | Regime *n* | Size | Kbytes | Technical Specifications |
| | | Access Time | cycles | |
| | | Line Size | bytes | |
| | | Miss Penalty | cycles | |
| Main Memory Size | | | Mbytes | Technical Specifications |
| Main Memory Speed | | | nanoseconds | Technical Specifications |
| Number of Processors | | | | Technical Specifications |
| Communication Latency | | | seconds | Technical Specifications |
| Message Overhead | | | seconds | Experimentally Obtained |
| Memory Fetches Hidden | | | % | Experimentally Obtained |
| Function Call Overhead | | | seconds | Experimentally Obtained |
| CPI | | | cycles/instruction | Experimentally Obtained |

Table 1 shows all the input parameters required by AHINT and the way each was determined. Notice the relatively small number of Experimentally Obtained parameters. While these parameters are not available in technical manuals, they can be determined for an existing machine by matching AHINT results with an actual HINT curve. The parameters can also be closely approximated by computer architects for machines in the design stage. The parameters obtained from Technical Specifications can be adjusted to reflect emperical data rather than nominal values.

The manual curve fitting revealed the effects of several variables on the HINT curve. Adjusting the CPI raises or lowers the entire HINT curve. Similarly, increasing/decreasing memory regime sizes lengthens/shortens the associated performance plateau. Since regime access times and miss penalties are additive, adjustments to the access time for regime *i* raises/lowers the curve for all levels *k*; where *k* >= *i*.

For parallel machines, changing the message latency changes the time for the global sum collapse and correspondingly changes the location of the initial points on the HINT curve. The lessons learned from the manual process have added insight to an automated HINT curve fitting program, currently under development. Given the data from an actual HINT run, the automated curve fitter will be able to vary the AHINT input parameters to match the model results with the actual results. This tool will be invaluable in understanding existing computer systems.

## 6 Conclusion

We have presented an accurate analytical model of the HINT performance metric. The model used is simple, yet predictive. The accuracy has been demonstrated by comparison of simulated results with actual HINT data. Because of its accuracy, the model can be effectively used in the design of a new computer. While HINT has become known as a powerful tool for analyzing the performance of an existing computer system, AHINT is an equally powerful tool that can be used to design a balanced computer system.

AHINT benefits computer design by providing an accurate estimate of the performance of a computer and allowing the architect to then balance the system for the best performance. It also allows users to assess the value of upgrades to their existing system. It can be used to reveal system parameters, particularly the performance of memory regimes, without reference to the manufacturer's claims. In this respect, AHINT appears unique among computer metrics.

## References

[1] J.L. Gustafson and Q.O. Snell, "HINT: A New way to measure computer performance", *Proceedings of the 28th Annual Hawaii International Conference on Systems Sciences, IEEE Computer Society Press, Vol. 2, pages 392-401.*

[2] HINT, http://www.scl.ameslab.gov/HINT.

[3] J.L. Hennesey and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, INC., San Francisco, CA, 1996.

[4] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, New York, 1991.

[5] Q.O. Snell, A.Mikler, and J.L. Gustafson, "NetPIPE: A Network Protocol Independent Performance Evaluator," *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[6] NetPIPE, http://www.scl.ameslab.gov/netpipe.

[7] *R4600/Orion MICROPROCESSOR Product Information*, http://www.mips.com/r4600/R4600_B.html, Silicon Graphics Inc., 1995.

[8] *Digital Technical Journal*, Vol. 7, Number 1, Digital Equipment Corporation, 1995

[9] D. Bhandarkar, *Alpha Implementations and Architecture: Complete Reference and Guide*, Digital Press, Newton, Massachusetts, 1996.

[10] J.J. Dongarra, *et al.*, "Performance of various computers using standard equations software in a FORTRAN environment," *ACM Computer Architecture News*, Vol. 18, Number 1, March 1990.

[11] Message Passing Interface Forum, MPI: A message-passing interface standard, *International Journal of Supercomputer Applications*, Volume 8, Number 3/4, 1994.

[12] S.L. Ambrosius, *et al.*, "Work-based performance measurement and analysis of virtual heterogeneous machines," *Proceedings of the 5th Workshop on Heterogeneous Computing*, Honolulu, Hawaii, April 1996.

[13] Q.O. Snell, A HINT on the Performance of the I-WAY, *GII Testbed Presentation* , Supercomputing '95.