How Well Can Simple Metrics Represent the Performance of HPC Applications?

Laura C. Carrington, Michael Laurenzano, & Allan Snavely

San Diego Supercomputer Center
CSE Dept. University of California
San Diego, CA 92093
{lcarring,michaell,allans}@sdsc.edu

Roy L. Campbell, Jr.

Army Research Laboratory
Major Shared Resource Center
Aberdeen Proving Ground,
MD 21005
rcampbell@arl.army.mil

Larry P. Davis

High Performance Computing
Modernization Program Office
Arlington, VA 22201
larryd@hpcmo.hpc.mil

Abstract

In this paper, a systematic study of the effects of complexity of prediction methodology on its accuracy for a set of real applications on a variety of HPC systems is performed. Results indicate that the use of any single, simple synthetic metric to predict performance does an inadequate job, and the use of a linear combination of these simple metrics with optimized weights also performs poorly. Better, however, are methodologies that rely on the convolution of an application "transfer function" based on tracing information with system performance data measured by simple benchmarks. This latter methodology can predict performance with an average accuracy of 80%, based on the current work.

1. Introduction.

Regardless of the underlying intent, a ranking of HPC systems has been of keen interest to many, as was demonstrated by the development of the Top 500 [1] (in 1993) using a simple benchmark (LINPACK) [2]. Now, with a plethora of tests available and a better understanding of the factors that impact performance, a strong desire to develop a new ordered list – one that accounts for all major system attributes (as well as cross-terms encountered when multiple attributes are exercised simultaneously) – has been commonly conveyed. In response, IDC released its Balanced Ratings [3] in November of 2001, covering three major categories (processor, memory, and interconnect performance). Some thought a successor to the Top 500 had finally been found, but many soon realized that the mapping of the performance of multiple categories into one score was highly subjective and workload-dependent. Therefore, the notion of ranking systems by a single metric outside of the context of a predefined workload, for the most part, had been dispelled.

Although this paper will not produce actual rankings, such rankings could be achieved by comparing the performance of applications across architectures (e.g., system X is 50% faster than system Y for application Z). This work will explore the extent and implication of ranking architectures through the estimation of performance for 10 DoD HPC Modernization Program (HPCMP) [4] systems with respect to a number of synthetic metrics – some workload-independent and others directly related to portions of the target workload defined by the DoD HPCMP 2005 Technology Insertion (TI-05). For each of five TI-05 application test cases, the base set of estimates will be validated against real application runtimes (see Appendix I tables 5-8) to determine the correlation of each estimator (or metric) to true performance data.

The 10 target systems span 9 distinct architectures ranging from a single OS, global-shared memory design to a multiple OS, distributed memory design. The architectures in order of their installation within the HPCMP are shown in Table 1, and a list of actual systems is shown in Table 2.

Table 1. Architectures used in study.

Make	Model	Processor Speed (GHz)	Interconnect
SGI	Origin 3800	0.400	NUMALink
IBM	Power 3	0.375	Colony
HP	SC45	1.000	Quadrics
IBM	p690	1.300	Colony
IBM	p690	1.700	Federation
LNX	Xeon	3.060	Myrinet
SGI	Altix	1.500	NUMALink
IBM	p655	1.700	Federation
IBM	Opteron	2.200	Myrinet

Table 2. Systems used in study.

HPCMP Site	Architecture	Compute Processors
ERDC	SGI_O3800_400MHz_NUMA	504
MHPCC	IBM_P3_375MHz_COL	736
NAVO	IBM_P3_375MHz_COL	928
ASC	HP_SC45_1GHz_QUAD	472
MHPCC	IBM_690_1.3GHz_COL	320
ARL	IBM_690_1.7GHz_FED	128
ARL	LNX_Xeon_3.06GHz_MNET	256
ARL	SGI_Altix_1.5GHz_NUMA	256
NAVO	IBM_655_1.7GHz_FED	2832
ARL	IBM_Opteron_2.2GHz_MNET	2304

The target metrics require some general discussion before they can be identified. Simple benchmarks (such as High Performance LINPACK (HPL) [2], STREAM [5], the HPC Challenge Benchmarks [6], the PMaC HPC Benchmark Suite [7], and the DoD HPCMP TI-XX Synthetic Probes) are easily executed and their resulting performance can be readily compared against expected system performance, as derived from manufacturer specifications. Their general usefulness, however, is limited by their weak correlation to the performance of real applications (since applications identified by careful workload characterization best represent the computational demands placed on a production system). For example, Gustafson et al. [8] showed that HPL was in fact anti-correlated with the performance of several applications on several machines. In other words, (ignoring price) if the system with the highest HPL result were purchased, that system would not only be a sub-optimal choice based on the applications data, but it would also be the worst choice. Of course, such pitfalls can be avoided by simply using application benchmarking data to make procurement decisions; however, application execution can be tedious and costly in terms of both manpower and system execution time. Therefore, if simple benchmarks could be correlated to application performance, more streamlined acquisition strategies could be developed, effecting more economical submission preparations for vendors and less cumbersome submission assessments for customers. Unfortunately, it is unlikely that simple synthetic benchmarks alone will ever posses sufficiently reliable performance ties to applications. Therefore, using synthetic benchmarks within a performance modeling and prediction framework seems to yield a stronger correlation to application performance. This framework strategically applies an application-specific "transfer function" to the test results such that the performance of multiple applications can be estimated using one set of synthetic results. The "transfer function" is deduced via tracing to extract memory and communications signatures for target applications. The corresponding probes or predictive synthetics are divided into two major tests – MEMBENCH MAPS and NETBENCH. MEMBENCH MAPS determines the memory bandwidth versus message size for unit and random stride cases, while NETBENCH determines the interconnect bandwidth and latency. An enhanced version of MEMBENCH MAPS was also developed to determine the memory bandwidth effects of loop data dependencies and branches within a loop.

Using five application test cases from the DoD HPCMP TI-05 Benchmarking Suite (AVUS-Standard, AVUS-Large, HYCOM-Standard, OVERFLOW2-Standard, and RFCTH-Standard), this paper will correlate actual performance to that of a wide variety of synthetics. These synthetics range from simple tests to probes tied to a predictive model. The complexity of the predictive model will be gradually increased by adding notional terms to its "transfer function." More specifically, the results of the tests denoted in Table 3 will be correlated to the performance of five TI-05 application test cases.

Section 2 describes each of the TI-05 application test cases in greater detail, and Section 3 describes the underlying framework for the predictive metrics. Section 4 reveals overall results, while Sections 5 and 6 reveal system-specific and application-specific results, respectively. Conclusions are provided in Section 7, and more detailed background and related work is provided in Section 8.

Table 3. Synthetic metrics used in study.

#	Type	Name or Description
1	Simple	HPL
2	Simple	STREAM
3	Simple	HPC Challenge Random Access (GUPS)
4	Predictive	HPL for floating point work
5	Predictive	HPL for floating point work; STREAM for memory access
6	Predictive	HPL for floating point work; STREAM for stride 1 memory access; GUPS for random stride memory access
7	Predictive	HPL for floating point work; MEMBENCH MAPS for memory access
8	Predictive	HPL for floating point work; MEMBENCH MAPS for memory access; NETBENCH for communications work
9	Predictive	HPL for floating point work; ENHANCED MEMBENCH MAPS for memory access; NETBENCH for communications work

2. DoD HPCMP TI-05 Application Test Cases.

The five application test cases are described in more detail below. Each test case was executed at 3 different processor counts, ranging from 16 to 384 processors.

AVUS STANDARD & LARGE

AVUS was developed by the Air Force Research Laboratory (AFRL) to determine the fluid flow and turbulence of projectiles and air vehicles. Its standard test case calculates 100 time-steps of fluid flow and turbulence for a wing, flap, and end plates using 7 million cells. Its large test case calculates 150 time-steps of fluid flow and turbulence for an unmanned aerial vehicle using 24 million cells.

HYCOM STANDARD

The Naval Research Laboratory (NRL), Los Alamos National Laboratory (LANL), and the University of Miami developed HYCOM as an upgrade to MICOM (both well-known ocean modeling codes) by enhancing the vertical layer definitions within the model to better capture the underlying science. HYCOM's standard test case models all of the world's oceans as one global body of water at a resolution of one-fourth of a degree when measured at the Equator.

OVERFLOW2 STANDARD

OVERFLOW-2 was developed by NASA Langley and NASA Ames to solve CFD equations on a set of overlapping, adaptive grids, such that the grid resolution near an obstacle is higher than that of other portions of the scene. This approach allows computation of both laminar and turbulent fluid flows over geometrically complex, non-stationary boundaries. The standard test case of OVERFLOW-2 models fluid flowing over five spheres of equal radius and calculates 600 time-steps using 30 million grid points.

RFCTH STANDARD

Sandia National Laboratories (SNL) developed CTH to model complex multidimensional, multiple-material scenarios involving large deformations or strong shock physics. RFCTH is a non-export-controlled version of CTH. The standard test case of RFCTH models a ten-material rod impacting an eight-material plate at an oblique angle, using adaptive mesh refinement with five levels of enhancement.

3. Performance Prediction Methodology.

Two main methodologies of performance prediction are explored in this work: simple and predictive. For the simple methodology a single benchmark or metric is used to predict the performance of the application on the target machine. In the predictive methodology, trace data from the application is used along with a set of simple benchmarks to yield a more application specific prediction. The details of these methodologies are described below.

For exploring the predictive power of the simple benchmarks in isolation, a very simple methodology is adopted. For each application, the predicted wall-clock time on a target system is calculated according to Equation 1 as a function of wall-clock time on the base system (namely, the NAVO p690) and the results for the set of simple synthetics.

$$T'(X,Y) = \frac{R(X)}{R(X_0)} \cdot T(X_0,Y) \qquad (1)$$

where $T'(X,Y)$ is the predicted wall-clock time for application Y on system X, $R(X)$ is the result of a specific simple benchmark for system X, X_0 denotes the base system, and $T(X,Y)$ is the measured wall-clock time for application Y on system X. In other words, the performance for a specific application is assumed to be faster or slower according to the ratio of the simple benchmark results for system X and the base system X_0.

Errors reported throughout this paper are calculated according to Equation 2.

$$\%\ Error = \frac{T'(X,Y) - T(X,Y)}{T(X,Y)} \cdot 100 \qquad (2)$$

Negative error indicates the prediction was faster than the actual runtime, while positive error indicates the prediction was slower than the actual runtime. After calculating signed error for each experiment, absolute error is calculated to ensure the magnitude of each deviation is considered when averaging across experiments, preventing error cancellation.

For the predictive metrics, a more sophisticated framework is used. An operation count per type (e.g., floating point, strided memory, and random memory) is accumulated via MetaSim Tracer [9] through instrumentation and dynamic tracing on a base system, while the rates for each operation type are measured via synthetic probes (simple benchmarks) on a target system.

Instrumentation and dynamic tracing are often quite expensive in terms of dilated execution time, especially when the address of each memory reference is the target

attribute, as is the case for Metrics #6-#9. In response, MetaSim has been carefully streamlined for speed, imposing approximately a 30x slowdown on an instrumented application. Unfortunately, the execution time for a TI-05 application test case can require 1-4 hours to execute without instrumentation, making tracing quite time-consuming, even with efficient methodology in place. Therefore, dilated execution time must be a weighed consideration when evaluating metric accuracy (one should ask "was the increase in accuracy worth the effort?"), although tracing does incur a non-recurring and mitigated cost, as it is only required once per application on the base system and therefore does not need to be repeated for each target system.

Operation counts, once determined by tracing, are divided by corresponding operation rates using MetaSim Convolver [10] to yield an execution time for the current basic block per operation type. Execution time is subsequently "predicted" by summing the estimated execution time for all basic blocks and carefully taking into account the overlap of the different operation types.

For (predictive) Metrics #4 and #5, only floating point instructions and floating point plus memory load/store instructions are counted, respectively. Therefore, MetaSim Trace is not the most efficient means for collecting such dynamic operation counts, since MetaSim Tracer reviews each address generated by load/store instructions. For such a simple task, performance counters provide a more expeditious result [11]. MetaSim Tracer is, however, needed for (predictive) Metrics #6, #7, #8, and #9 as each requires the discrimination of unit and random stride memory instructions. MetaSim Tracer parses the address stream with a stride detector [12], thus determining what portion of memory references are stride-1, non-unit short strides (up to stride-8), and random stride.

For all predictive metrics (Metrics #4-#9), the floating point issue rate was assumed to be the per processor R_{max} from HPL. For Metric #5, the memory instruction rate was defined by the results of STREAM, while, for Metric #6, the unit and random-stride memory instruction rates were defined by the results of STREAM and GUPS, respectively. For Metrics #7-#9, the unit and random-stride memory instruction rates were assessed as a function of working-set size (by the MAPS portion of MEMBENCH) to better reflect the seemingly discrete levels of memory bandwidth observed in cache-based architectures. The MAPS results for Metric #9 included memory instruction rates for loop and control-flow dependency as a function of working-set size, in addition to the standard MAPS curves for unit and random-stride.

STREAM and GUPS are typically executed from main memory; whereas, MAPS measures the rate at which loads and stores (both strided and random) are performed from different levels of the memory hierarchy. MAPS is equivalent to launching multiple instances of both STREAM and GUPS at various "sizes" in order to span the various levels of cache (L1, L2, and L3 (if it exists)) and main memory. To further illustrate, Figure 1 shows unit-stride MAPS results for three different systems. (A subset of the 12 target systems was plotted to improve readability.)

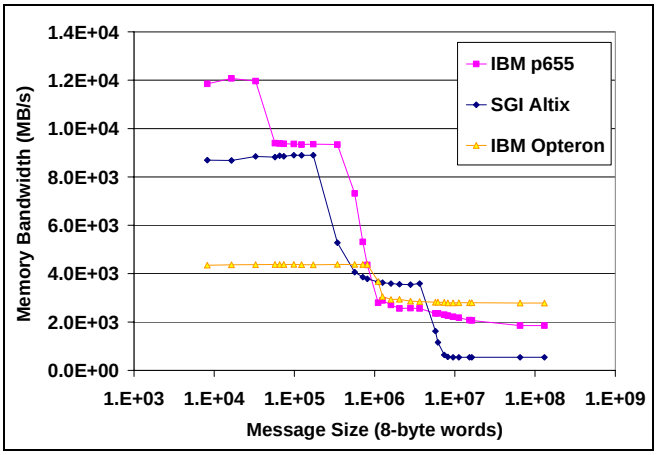

Figure 1. Unit-stride memory bandwidth versus message size for three target systems.

The lower right-hand portion of each unit-stride MAPS curve (plotted) corresponds to the STREAM score for each system, while the lower right-hand portion of each random stride MAPS curve (not plotted) corresponds to the GUPS score for each system. As shown, the IBM Opteron scored best for executions from main memory and would therefore score best for STREAM. However, if the size of STREAM were reduced to fit into L2 cache and subsequently L1 cache, the SGI Altix and IBM p655 would score best, respectively. Therefore, in general, the ranking of systems according to memory performance greatly depends on the stride signature of the application, since the spacing of memory references determines what level of cache is being exercised at any given time during execution.

For (predictive) Metric #8 [13], a notional term was added to account for interconnect performance, using MPIDTRACE [14] to count MPI communications events in applications and NETBENCH to measure the interconnect latency and bandwidth of the target system. For (predictive) Metric #9, a second notional term was added to account for loop and control-flow dependencies. Static analysis was applied to the binary executable for each application on the base system, so ILP limited basic blocks could be identified. Corresponding "dependency" MAPS curves then were obtained via ENHANCED MAPS (in addition to the standard MAPS curves) by inducing data and control-flow dependencies in the inner loop of both STREAM and GUPS.

4. General Results.

Performance was deduced using the nine synthetic metrics listed in Table 3 for each of five TI-05 application test cases at three different processor counts and was then compared to actual real run times to yield an absolute error for each metric and test case pair. The resulting absolute error was subsequently averaged per metric to provide an error profile for the suite of metrics. Table 4 presents, and Figure 2 shows graphically, a summary of the average absolute error for the nine target synthetics.

Table 4. Error assessment: metric results as compared to application's real run time.

# & Type	Metric Description	Average Absolute Error (%)	Standard Deviation (%)
1-S	HPL	63	68
2-S	STREAM	43	73
3-S	GUPS	33	27
4-P	HPL	63	68
5-P	HPL+STREAM	50	72
6-P	HPL+STREAM+GUPS	22	18
7-P	HPL+MAPS	24	21
8-P	HPL+MAPS+NET	22	18
9-P	HPL+MAPS+NET+DEP	18	18

In general, the correlation of the predictive metrics is increasingly better than that of the simple metrics, as notional terms are added to the "transfer function" portion of the predictive model. A more detailed description of the results for each metric is warranted.

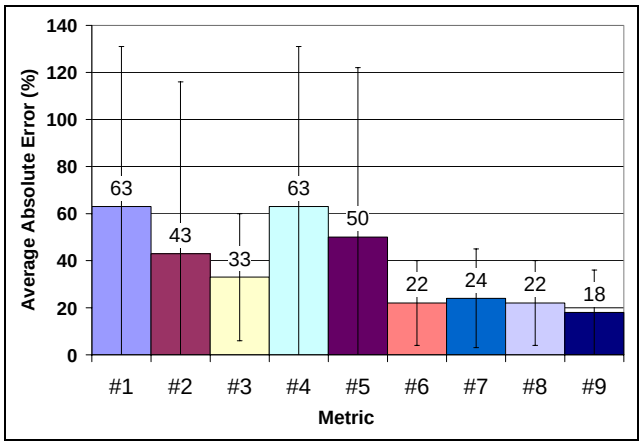

Figure 2. Graphical version of error assessment: metric results as compared to application's real run time.

#1 – Simple Metric (HPL)

Performance was estimated by taking the ratio of the per processor R_{max} scores for each of 10 systems, yielding an average absolute error of 63% with a standard deviation of 68%. The results confirm the common wisdom that, while HPL is a good measure of peak floating point issue rate, it is not a good predictor of absolute or even relative performance for real applications. Applications, to varying degrees, execute non-floating point operations that are not typically hidden by execution overlap, and a system's ability to execute such operations often weakly correlates with its floating point performance.

#2 – Simple Metric (STREAM)

Performance was estimated by taking the ratio of each system's ability to load unit-stride operations from main memory as measured by STREAM, yielding results that were marginally better than those for HPL – an average absolute error of 43% with a standard deviation of 73%. This result confirms what many now believe to be true – the relative performance of memory subsystems is likely a better indicator of relative application performance than HPL (or is at least equally as good), since memory access is increasingly becoming the primary bottleneck for today's systems for many applications.

#3 – Simple Metric (GUPS)

Performance was estimated by taking the ratio of each system's per processor ability to update values in random locations of main memory, yielding an average absolute error of 33% with a standard deviation of 27%. This result is an improvement to the results of the prior two metrics, and seems to support the idea that the expense of random access for today's systems tends to dominate the memory response for an application, even if the application performs only limited random access.

Viewed collectively, the first three assessments seem to suggest that the relative performance of target applications on specific systems cannot be reliably correlated with a single simple metric such as HPL, STREAM, or GUPS, considering the high percentage of average absolution error observed for each. In response, some have suggested that a fixed linear combination of simple metrics might be used. For example, as [15] explains, IDC's Balanced Rating simply combines the results for three metric categories (processor, memory, and interconnect) by normalizing performance for each to yield intermediate scores from 0% to 100% and then weighting each category equally to produce a composite score from 0% to 100%. Applying this methodology to predict performance of these applications using HPL, STREAM, and all_reduce (an MPI test within NETBENCH), an average absolute error of 35% with a standard deviation of 25% was observed. Using linear regression, category weightings which minimize estimation error were determined to be 5% (HPL), 50% (STREAM), and 45% (all_reduce), resulting in an average absolute error

of 33% with a standard deviation of 30% – still quite sizable. This seems to disprove the notion that a single "balanced rating" can significantly improve on a simple benchmark given that GUPS alone did this well. Therefore, to improve matters, the idea of variable weights for simple metrics is explored, such that the proportion of each metric is determined by the proportion of operations of each type found in the target application (as summarized by an application-specific "transfer function").

#4 – Predictive Metric (HPL)

As described in the Performance Prediction Methodology Section, the predictive framework counts operations of various types for each basic block of an application via tracing on a base system, and then measures the various operation rates on a target system. The convolver divides operation counts by corresponding operation rates to yield an execution time for the current basic block per operation type. Applying this technique using floating point operations only and assuming the floating point issue rate for a particular target system is the per processor R_{max} (from HPL), the execution time can be estimated by summing the estimated execution time for all basic blocks and subsequently assuming that the total execution time is indirect proportional to the per processor R_{max} in order to estimate the performance of other systems. As shown in Table 4, the results for this metric were the same as those for Metric #1, demonstrating that in the simplest case, the convolver's execution is identical to that of a pencil-and-paper calculation using the per processor R_{max} as the floating point issue rate.

#5 – Predictive Metric (HPL+STREAM)

Both floating point and memory operations were counted, while the floating point issue and memory rates were assumed to be the per processor R_{max} and the STREAM memory bandwidth, respectively. At this level of granularity, the predictive model results – an average absolute error of 50% and a standard deviation of 72% – were not much better than those for the simple metric using HPL (Metric #1).

#6 – Predictive Metric (HPL+STREAM+GUPS)

Floating point, strided memory, and random memory operations were collected, while the corresponding rates were assumed to be the per processor R_{max}, the STREAM memory bandwidth, and the GUPS memory bandwidth, respectively. Adding the discrimination between strided and random memory access seemed to make a substantial difference, as the average absolute error dropped to 22% with a standard deviation of 18%.

#7 – Predictive Metric (HPL+MAPS)

The per processor R_{max} was again used for the floating point issue rate, but the STREAM and GUPS memory bandwidths were replaced with MAPS curves (i.e., curves for unit and random-stride that describe the memory bandwidth versus message size, revealing the bandwidth characteristics for different levels of the memory hierarchy – L1, L2, L3 (if it exists), and main memory). The results for this metric, an average absolute error of 24% with a standard deviation of 21%, were marginally worse than those for Metric #6, possibly suggesting that the granularity added by MAPS is unnecessary.

#8 – Predictive Metric (HPL+MAPS+NETBENCH)

The per-processor R_{max} was yet again used for the floating point issue rate. Unit and random-stride MAPS curves were used to define the throughput for the memory subsystem at different hierarchical levels, and NETBENCH results were added to describe the communications signature of the target system (explained in greater detail in the Performance Prediction Methodology Section). The results showed a marginal improvement to those for Metric #7, yielding an average absolute error of 22% with a standard deviation of 18%.

#9 – Predictive Metric (HPL+MAPS+NETBENCH+ DEPENDENCY)

Finally, rate and probe assignments for Metric #8 were retained with the exception that MAPS was replaced with ENHANCED MAPS as the memory signature for the target system, noting that ENHANCED MAPS additionally accounts for the effects of loop data dependencies and branches within a loop. This addition resulted in a modest improvement over the results for Metrics #6-#8, yielding an average absolution error of 18% and a standard deviation of 18%.

Taken as a whole, the results for Metrics #6-#9 suggest simple synthetics may indeed be able to account for approximately 80% of relative performance across systems when viewed through an application-specific framework (i.e., by counting operations of various types for each application). Before making a final determination, however, that Metrics #7-#9 provide only nominal improvements, system-specific results in the next section are considered.

5. System-Specific Results.

Table 5 contains the average absolute error for each system and metric pair. Intuitively, one would expect that adding notional terms to the predictive model would reduce

error, especially for Metrics #6-#9 as granularity can clearly be said to increase with the number of the metric. The ASC SC45 and ERDC 03800 did indeed follow this trend; however, in general, the estimation space is more complicated than the artificial one described by this supposed trend, as the additional complexity that accompanies finer granularity typically leads to additional sources of error. For example, while Metric #6 distinguishes between strided and random memory access, Metric #7 adds additional complexity (and likely additional error sources) to distinguish between (presumed to be) fast running cache friendly loops and loops that actually fall out of cache.

Table 5. System-specific average absolute percent error.

	Metric #								
System	1	2	3	4	5	6	7	8	9
ERDC_O3800	37	12	83	37	84	35	29	20	22
MHPCC_P3	58	53	19	58	52	14	29	24	25
NAVO_P3	37	77	28	37	75	8	15	10	7
ASC_SC45	167	14	59	167	15	31	28	18	16
MHPCC_690_1.3	122	14	14	122	13	15	17	29	24
ARL_690_1.7	26	21	21	26	21	22	23	34	28
ARL_Xeon	42	37	23	42	37	21	64	39	21
ARL_Altix	193	281	64	193	272	36	25	27	26
NAVO_655	19	12	19	19	12	14	16	14	9
ARL_Opteron	20	29	45	20	27	44	30	32	26
OVERALL	63	43	33	63	50	22	24	22	18

Adding a term for predicting loop performance based on size and expected cache hit rate (as in Metric #7) performed slightly worse than simply assuming the unit and random-stride memory instruction rates were single invariant values obtained from STREAM and GUPS. Subsequently, adding a network term (as in Metric #8) improved predictions for 6 of 10 systems, had no net effect for 2, and worsened predictions for 2, yielding an overall modest reduction in average error (comparing Metric #8 to #7). Finally, adding a dependency term (as in Metric #9) to correct a basic flaw in the term introduced by Metric #7 (by identifying loops with performance-limiting internal branches and/or data dependencies) improved predictions as compared to those for Metric #6 for 7 of 10 systems, yielding a marginal overall improvement between the two metrics and suggesting that the addition of the notional terms associated with Metrics #7 and #8 may have merit once all new terms for Metrics #7-#9 are combined (as opposed to introducing the new terms individually).

6. Application-Specific Results.

Figures 3 through 7 provide an error assessment for Metrics #1-#9 broken down by application test case. As can be seen from these figures (and Table 5), this study is based on a substantial amount of data involving real applications and real systems. To quantify, five application test cases were executed at three processor counts each on 10 different systems, resulting in a total of 150 (5x3x10) observed application executions. In addition, a substantial number of predictions were made, as 9 metrics were applied to each of 150 execution observations for a total of 1,350 (9x150) predictions. Therefore, although these metrics have not been tested across all possible HPC systems and application domains, this study has thoroughly evaluated the metrics on a notable portion of the HPC space.

Of the simple metrics, HPL was not an accurate predictor for any of the 15 (application test case, processor count) pairings, given that HPL's best observed average absolute error was 55% (which occurred for AVUS Standard at 128 processors). In all but one case (OVERFLOW2 Standard at 48 processors), HPL was the worst of all of the predictors, although in a few cases it tied with HPL+STREAM. STREAM was a better predictor than HPL in all but one case (OVERFLOW2 Standard at 48 processors), although it never achieved better than 22% average absolute error. GUPS was a better predictor than STREAM in 11 out of the 15 possible cases, but never achieved better than 31% average absolute error. Therefore, if a simple metric for predicting relative performance were required to represent application performance, it seems GUPS would be slightly preferred over STREAM, and either GUPS or STREAM would be preferred over HPL.

Among the predictive metrics, Metric #4 simply provided a sanity test for the predictive method. Metric #5 (HPL+STREAM) performed worse than GUPS alone in all but one case (HYCOM Standard at 59 processor) – being only 1% better at that point. Metric #6 (HPL+STREAM+GUPS) performed substantially better than the prior two, since in 4 out 15 cases this metric was the best predictor for all nine metrics, and in 2 cases it tied Metric #9 as the best predictor. Therefore, the complexity added by Metric #6 (i.e., employing tracing to bin memory references into strided and random access) seems to have been worthwhile, validating the suspected need for an application-specific weighting of floating point, strided memory, and random memory operations.

Metrics #7 (HPL+MAPS) and #8 (HPL+MAPS+ NETBENCH) often performed slightly worse than Metric #6. The MAPS curve, Figure 1, certainly suggests that it would be valuable to distinguish memory performance at different levels of cache; however, without taking control and data dependencies into account (that may cause cacheable loops to execute slowly) MAPS appears to actually introduce error (comparing #7 to #6). Adding a network term does improve results (comparing #8 to #7), although not significantly because these application cases are not communication bound.

Metric #9 (HPL+MAPS+NETBENCH+DEPENDENCY), which retains the features of Metrics #7 and #8 (while adding data and control flow analysis), however, was the best of all the predictors for 8 of the 15 cases, and tied for best in 2 other cases.

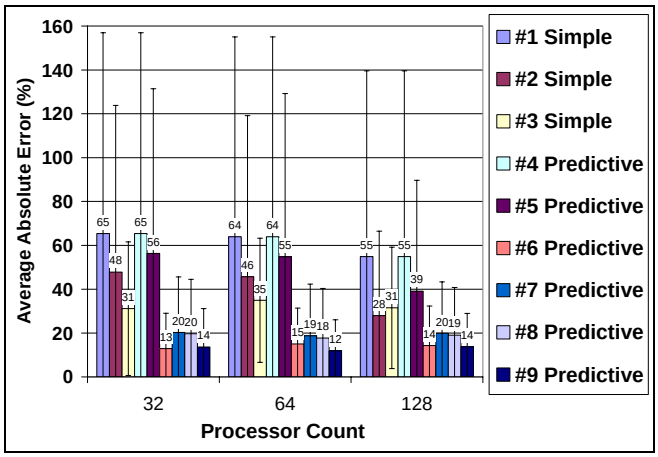

Figure 3. Graphical error assessment for AVUS Standard.

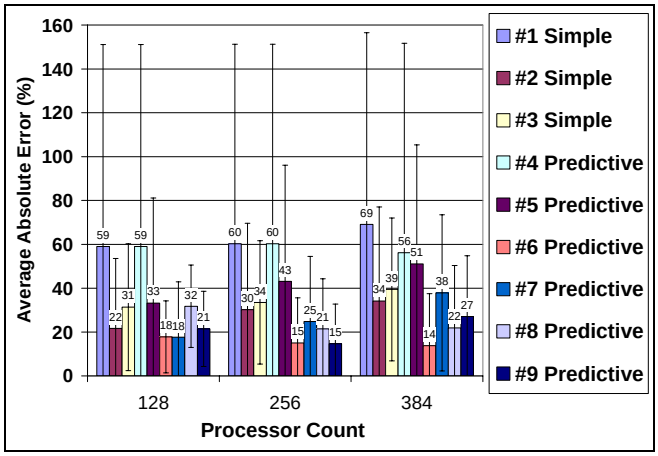

Figure 4. Graphical error assessment for AVUS Large.

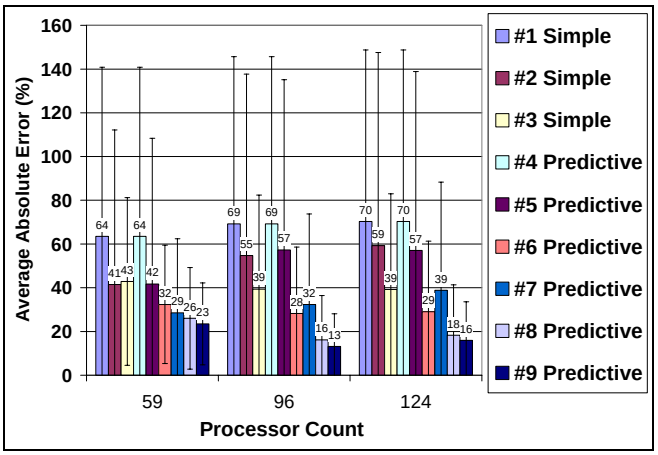

Figure 5. Graphical error assessment for HYCOM Standard.

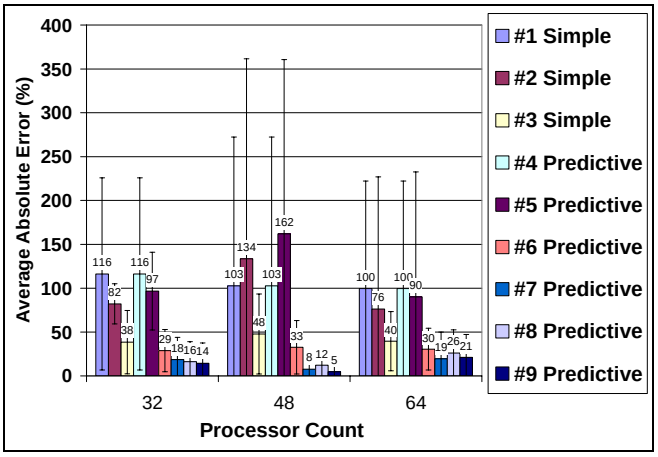

Figure 6. Graphical error assessment for OVERFLOW2 Standard.

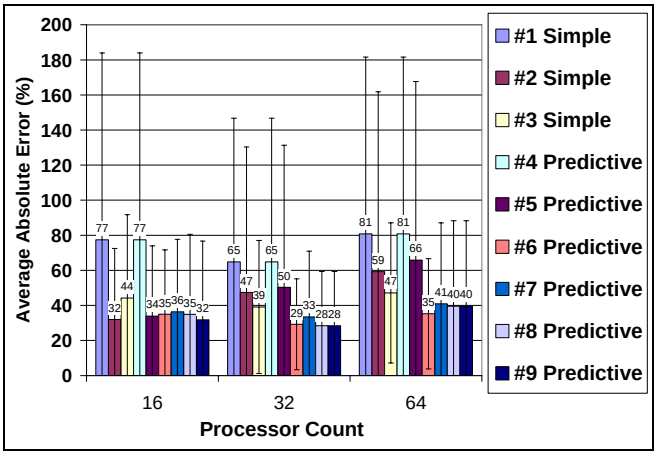

Figure 7. Graphical error assessment for RFCTH Standard.

Therefore, it seems that Metrics #6 and #9 provided the most consistent representation of the application test cases considered, realizing that #6 was less accurate on average than #9, but that #6 was also less complicated. In both cases, the expensive step of memory tracing was required,

although it should be noted that once tracing is completed for any one metric it is readily available for others.

7. Conclusions.

In this paper, several simple and predictive metrics were evaluated to see how well each could predict application performance and rank HPC systems. Predictive metrics combined simple metrics with application-specific data through a generic "transfer function" (deduced via tracing) in order to increase the degree of application representation of the simple metrics.

From the resulting data, the practical use of ranking or predicting system performance via single metrics such as HPL, STREAM or GUPS, seems to be quite limited, although the latter two metrics (which happen to be memory-oriented) are clearly more representative than the former (which happens to be floating-point-oriented). If, however, information about operation types specific to a target application is acquired (rather painfully through tracing when discriminating memory access types), then a few simple metrics can be combined and weighted appropriately to predict performance and rank with about 80% accuracy, at least in this particular fairly large study of quite a few HPC architectures and several HPC applications.

8. Related Work.

Several benchmarking suites have been proposed to represent the general performance of HPC applications. Besides those mentioned previously, probably the best known are the NAS Parallel [16] and the SPEC [17] benchmarking suites, the latter of which is often used to evaluate micro-architecture features of HPC systems. Both, however, are composed of "mini-applications", and are, therefore, fairly complicated to relate to the performance of general applications, as opposed to the simple benchmarks considered here. Gustafson and Todi [8] performed seminal work relating "mini-application" performance to that of full applications, but they did not extend their ideas to large-scale systems and applications, as this paper does. McCalpin [5] showed improved correlation between simple benchmarks and application performance, but did not extend the results to parallel applications. Marin and Mellor-Crummey [18] show a clever scheme for combining and weighting the attributes of applications by the results of simple probes, similar to what is implemented here, but their application studies were mostly focused on "mini application" benchmarks, and were not extended to parallel applications and systems.

Methods for performance evaluations can be broken down into two areas [19]: structural models and functional/analytical models. Structural models use descriptions of individual system components and their interactions, similar to the process used for detailed simulations. Functional/analytical models, on the other hand, separate the performance factors of a system to create a mathematical model.

The use of detailed or cycle-accurate simulators in performance evaluation has been used by many researchers [20-24]. Detailed simulators are normally built by manufactures during the design stage of an architecture to aid in the design. For parallel machines, two simulators might be used, one for the processor and one for the network. These simulators have the advantage of automating performance prediction from the user's standpoint. The disadvantage is that these simulators are proprietary and often not available to HPC users and Centers. Also, because they capture all the behavior of the processors, simulations can take on an upwards of 1,000,000 times longer, than the real runtime of the application [25]. This means, to simulate 1 hour of an application it could take approximately 114 years of CPU time. Direct execution methods are commonly used to accelerate architectural simulations [26] but they still can have large slowdowns. To avoid these large computational costs, cycle-accurate simulators are usually only used to simulate a few seconds of an application. This causes a modeling dilemma, for most scientific applications the complete behavior cannot be captured in a few seconds of a production run. Applications rarely spend all their time in one routine and their behavior may change as the application progresses through its simulation (in some cases the actual physics of the problem being solved changes).

Cycle-accurate simulators are limited to only work in modeling the behavior of the processor for which they were developed, so they are not applicable to other architectures. In addition, the accuracy of cycle-accurate simulation can be questionable. Gibson et al [27] showed that simulators that model many architectural features have many possible sources for error, resulting in complex simulators that produce greater than 50% error. This work suggested that simple simulators are sometimes more accurate than complex ones.

In the second area of performance evaluation, functional and analytical models, the performance of an application on the target machine can be described by a complex mathematical equation. When the equation is fed with the proper input values to describe the target machine, the calculation yields a wall clock time for that application on the target machine. Various flavors of these methods for developing these models have been researched. Below is a

brief summary of some of this work but due to space limitations it is not meant to be inclusive of all.

Saavedra [28-30] proposed applications modeling as a collection of independent Abstract FORTRAN Machine tasks. Each abstract task was measured on the target machine and then a linear model was used to predict execution time. In order to include the effects of memory system, they measured miss penalties and miss rates to include in the total overhead. These simple models worked well on the simpler processors and shallower memory-hierarchies of the mid 90's. The models now need to be improved to account for increases in the complexity of parallel architectures including processors, memory subsystems, and interconnects.

For parallel system predictions, Mendes [20-21] proposed a cross platform approach. Traces were used to record the explicit communications among nodes and to build a directed graph based on the trace. Sub-graph isomorphism was then used to study trace stability and to transform the trace for different machine specifications. This approach has merit and needs to be integrated into a full system for applications tracing and modeling of deep memory hierarchies in order to be practically useful today.

Simon [22] proposed to use a Concurrent Task Graph to model applications. A Concurrent Task Graph is a directed acyclic graph whose edges represent the dependence relationship between nodes. In order to predict the execution time, it was proposed to have different models to compute the communication overhead, (FCFS queue for SMP and Bandwidth Latency model for MPI) with models for performance between communications events. As above, these simple models worked better in the mid 1990's than today.

Crovella and LeBlanc [23] proposed complete, orthogonal and meaningful methods to classify all the possible overheads in parallel computation environments and to predict the algorithm performance based on the overhead analysis. Our work adopts their useful nomenclature.

Xu, Zhang, and Sun [24**]** proposed a semi-empirical multiprocessor performance prediction scheme. For a given application and machine specification, the application first is instantiated to thread graphs which reveal all the possible communications (implicit or explicit) during the computation. They then measured the delay of all the possible communication on the target machine to compute the elapsed time of communication in the thread graph. For the execution time, of each segment in the thread graph between communications, they use partial measurement and loop iteration estimation to predict the execution time. The general idea of prediction from partial measurement is adopted here.

Abandah and Davidson, [25] and Boyd et al [26] proposed hierarchical modeling methods for parallel machines that is kindred in spirit to our work, and was effective on machines in the early and mid 90's.

A group of expert performance modelers at Los Alamos have been perfecting the analytical model of applications important to their workload for years [38-41]. These models are quite accurate in their predictions, although the methods for creating them are time consuming and not necessarily easily done by non-expert user [42].

9. Acknowledgements.

The applications benchmarking data used in this study was obtained by the following cast of many (listed alphabetically): Mr. Wendell Anderson (NRL), Mr. Robert W. Alter (ERDC-CSC), Dr. Paul M. Bennett (ERDC-CSC), Dr. Sam B. Cable (ERDC-CSC), Dr. John E. Cazes (TACC), Ms. Christine Cuicchi (NAVO), Ms. Carrie L. Leach (ERDC-CSC), Mr. Mitch Murphy (MHPCC), Dr. Thomas C. Oppe (ERDC-CSC), Mr. Daniel M. Pressel (ARL), Mr. Daniel S. Schornak (ASC-CSC), and Dr. William A. Ward, Jr. (ERDC-CSC). Application traces were gathered by (besides the authors) Mr. Mike Timmerman (Instrumental) and Ms. Cynthia Bailey Lee. Mr. Xiaofeng Gao wrote the binary analyzer used to determine data and control-flow dependencies. We would also like to acknowledge the European center for Parallelism of Barcelona, Technical University of Barcelona (CEPBA) for their continued support of their profiling and simulation tools. This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the ARL, ASC, ERDC, and NAVO Major Shared Resource Centers, the MHPCC Allocated Distributed Center, and the NRL Dedicated Distributed Center. This work was sponsored in part by the Department of Energy Office of Science through SciDAC award High-End Computer System Performance: Science and Engineering. Computer time was also provided by SDSC. Additional computer time was graciously provided by the Pittsburgh Supercomputer Center via an NRAC award.

10. References.

1. Top500, www.top500.org.

2. J. Dongarra, P. Luszczek, & A. Petitet, "The LINPACK benchmark: past, present and future", *Concurrency and*

Computation: Practice and Experience, vol. 15, pp. 1-18, 2003.

3. E. Joseph, C. G. Willard, M. Swenson, & D. Goldfarb, "A new HPC technical computing benchmark: the IDC balanced rating", *IDC Bulletin W*.

4. High Performance Computing Modernization Program, www.hpcmo.hpc.mil.

5. J. McCalpin, "Memory bandwidth and machine balance in current high performance computers", *IEEE Technical Committee on Computer Architecture Newsletter*.

6. HPC Challenge Benchmarks, http://icl.cs.utk.edu/hpcc/.

7. PMaC HPC Benchmark Suite, http://www.sdsc.edu/pmac/.

8. J. Gustafson & R. Todi, "Conventional benchmarks as a sample of the performance spectrum", *Hawaii International Conference on System Sciences*, 1998.

9. L. Carrington, A. Snavely, N. Wolter, & X. Gao, "A performance prediction framework for scientific applications", *Workshop on Performance Modeling and Analysis-ICCS*, Melbourne, 2003.

10. A. Snavely, X. Gao, C. Lee, N. Wolter, & J. Labarta, "Performance modeling of HPC applications", *Parallel Computing*, Dresden, 2003.

11. S. Browne, J. Dongarra, N. Garner, K. London, & P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters", *SC2000*, Dallas, 2000.

12. J. Hollingsworth, A. Snavely, & S. Sbaraglia, "EMPS: an environment for memory performance studies", *Proceedings of the 19th IEEE International Parallel and Distributed Systems (IPDPS)*, Washington D.C., 2005.

13. A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, & A. Purkayastha, "A framework for application performance modeling and prediction", *SC2002*, Baltimore, 2002.

14. R. Badia, G. Rodriguez, & J. Labarta, "Deriving analytical models from a limited number of runs", *Parallel Computing*, Dresden, 2003.

15. Ad Emmen, "IDC reports latest supercomputer rankings based on the IDC balanced rating test", *Primeur Monthly*, May 16, 2002, http://www.hoise.com/ primeur/02/articles/monthly/AE-PR-06-02-45.html.

16. D. Bailey, J. Barton, T. Lasinski, H. Simon, "The NAS parallel benchmarks", *International Journal of Supercomputer Applications*, 1991.

17. SPEC, http://www.spec.org/.

18. G. Marin & J. Mellor-Crummey, "Cross-architecture performance predictions for scientific applications using parameterized models", *SIGMETRICS Performance 04*, 2004.

19. L. Svobodova, Computer System Performance Measurement and Evaluation Methods: Analysis and Applications (Elsevier, N.Y.1976).

20. R.S., Ballansc, J.A. Cocke, and H.G. Kolsky, The Lookahead Unit, Planning a Computer System, (McGraw-Hill, New York, 1962).

21. L.T. Boland, G.D. Granito, A.V. Marcotte, B.V. Messina, and J.W. Smith, "The IBM system 360/Model9:Storage System", *IBM J. Res. And Develop.*,vol. 11, pp. 54-79, 1967.

22. D. Burger, T.M. Austin, and S. Bennett, "Evaluating future microprocessors: The simplescalar tool set", *Tech. Rep. CS-TR-1996-1308*, University of Wisconsin-Madison , 1996.

23. J.O. Murphey and R.M. Wade, "The IBM 360/195", *Datamation*, vol. 16:4, pp. 72-79, 1970.

24. G.S. Tjaden and M.J. Flynn, "Detection and Parallel Execution of Independent Instructions", *IEEE Trans. Comptrs.*, vol. C-19 pp. 889-895, 1970.

25. J. Lo, S. Egger, J. Emer, H. Levy, R. Stamm, and D. Tullsen, "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading", *ACM Transactions on Computer Systems*, August, 1997.

26. B. Falsafi and D.A. Wood, "Modeling Cost/Performance of a Parallel Computer Simulator", *ACM Transactions on Modeling and Computer Simulation*, vol. 7:1, pp. 104-130, 1997.

27. J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich," FLASH vs. (Simulated) FLASH: Closing the Simulation Loop*", Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, November, pp. 49-58, 2000.

28. R.H. Saavedra and A.J. Smith, “Measuring Cache and TLB Performance and Their Effect on Benchmark Run Times”, *IEEE Transactions on Computers*, vol. 44:10 pp. 1223-1235, 1995.

29. R.H. Saavedra and A.J. Smith, “Analysis of Benchmark Characteristics and Benchmark Performance Prediction”, *TOCS14*, vol. 4, pp. 344-384, 1996.
30. R.H. Saavedra and A.J. Smith, “Performance Characterization of Optimizing Compilers”, *TSE21*, vol. 7, pp. 615-628, 1995.

31. C.L. Mendes and D.A. Reed, ”Integrated Compilation and Scalability Analysis for Parallel Systems”, *IEEE PACT*, 1998.

32. C.L. Mendes and D.A. Reed, “Performance Stability and Prediction”, *IEEE /USP International Workshop on High Performance Computing*, 1994.

33. J. Simon and J. Wierun, “Accurate Performance Prediction for Massively Parallel Systems and its Applications”, *Euro-Par*, vol. 2, pp. 675-688, 1996.
34. M.E. Crovella and T.J. LeBlanc, “Parallel Performance Prediction Using Lost Cycles Analysis”, *SuperComputing 1994*, pp. 600-609, 1994.

35. Z. Xu, X. Zhang, L. Sun, “Semi-empirical Multiprocessor Performance Predictions”, *JPDC*, vol. 39, pp. 14-28, 1996.

36. G. Abandah, E.S. Davidson, “Modeling the Communication Performance of the IBM SP2”, *Proceedings Int'l Parallel Processing Symposium*, April, pp. 249-257, 1996.

37. E.L. Boyd, W. Azeem, H.H. Lee, T.P. Shih, S.H. Hung, and E.S. Davidson, “A Hierarchical Approach to Modeling and Improving the Performance of Scientific Applications on the KSR1”, Proceedings of the 1994 *International Conference on Parallel Processing*, vol. 3, pp. 188-192, 1994.

38. A. Hosie, L. Olaf, H. Wasserman, “Performance Analysis of Wavefront Algorithms on Very-Large Scale Distributed Systems”, *Springer’s “Lecture Notes in Control and Information Sciences”*, vol. 249, p. 171, 1999.

39. A. Hosie, L. Olaf, H. Wasserman, “Scalability Analysis of Multidimensional Wavefront Algorithms on Large-Scale SMP Clusters”, *Proceedings of Frontiers of Massively Parallel Computing ’99*, Annapolis, MD, February, 1999.

40. D.J. Kerbyson, A. Hoisie, and H.J. Wasserman, “Modeling the Performance of Large-Scale Systems”, *Keynote paper, UK Performance Engineering Workshop (UKPEW03),* July, 2003.

41. L. Yong, L.M. Olaf, H. Wasserman, “Development and Validation of a Hierarchical Memory Model Incorporating CPU- and Memory-Operation Overlap”, *Proceedings of the First International Workshop on Software and Performance*, Santa Fe, NM, pp. 152-163, 1996.

42. A. Spooner and D. Kerbyson, “Identification of Performance Characteristics from Multi-view Trace Analysis”, *Proc. Of Int. Conf. On Computational Science (ICCS*), part 3 2659, pp. 936-945, 2003.

11. Appendix.

Table 6. AVUS Standard observed times-to-solution.

Machine	Run times in seconds		
	32-CPUs	64-CPUs	128-CPUs
ERDC O3800	12737	5881	2733
MHPCC P3	15051	8354	3779
NAVO P3	18195	8601	3870
ASC SC45	6993	3334	1617
MHPCC 690 1.3	10286	4932	2368
ARL 690 1.7	8625	4466	1935
ARL Xeon	9115	4686	2422
ARL Altix	5872	2842	---
NAVO 655	6703	3115	1460
ARL Opteron	5527	2747	1401

Table 7. AVUS Large observed times-to-solution.

Machine	Run times in seconds		
	128-CPUs	256-CPUs	384-CPUs
ERDC O3800	18103	8577	5736
MHPCC P3	40177	12123	7706
NAVO P3	26362	12379	8042
ASC SC45	10412	5199	3394
MHPCC 690 1.3	14751	7591	---
ARL 690 1.7	12718	---	---
ARL Xeon	13654	6890	---
ARL Altix	---	---	---
NAVO 655	9844	4576	2949
ARL Opteron	8599	4273	2884

Table 8. HYCOM Standard observed times-to-solution.

Machine	Run times in seconds		
	59-CPUs	96-CPUs	124-CPUs
ERDC O3800	6619	4329	4449
MHPCC P3	10453	3912	2992
NAVO P3	7129	4420	3348
ASC SC45	3594	2469	1949
MHPCC 690 1.3	3532	2939	2661
ARL 690 1.7	2586	1675	1510
ARL Xeon	3705	2504	1991
ARL Altix	2263	1462	1176
NAVO 655	2010	1281	990
ARL Opteron	1936	1268	1031

Table 9. Overflow2 Standard observed times-to-solution.

Machine	Run times in seconds		
	32-CPUs	48-CPUs	64-CPUs
ERDC O3800	10875	8008	5497
MHPCC P3	14939	---	7371
NAVO P3	14939	---	7371
ASC SC45	6329	---	4109
MHPCC 690 1.3	9156	---	4701
ARL 690 1.7	---	---	---
ARL Xeon	---	---	---
ARL Altix	3143	2389	1730
NAVO 655	5454	4031	2908
ARL Opteron	---	---	---

Table 10. RF-CTH2 observed times-to-solution.

Machine	Run times in seconds		
	16-CPUs	32-CPUs	64-CPUs
ERDC O3800	6182	3268	1793
MHPCC P3	6557	3475	1869
NAVO P3	6557	3475	1869
ASC SC45	3134	2170	1005
MHPCC 690 1.3	2777	1813	1275
ARL 690 1.7	2154	1660	5156
ARL Xeon	4203	2308	1368
ARL Altix	---	1122	614
NAVO 655	1982	1075	607
ARL Opteron	1882	1072	671