Introduction to the HPC Challenge Benchmark Suite

Piotr Luszczek[1], Jack J. Dongarra[1,2], David Koester[3], Rolf Rabenseifner[4,5], Bob Lucas[6,7], Jeremy Kepner[8], John McCalpin[9], David Bailey[10], and Daisuke Takahashi[11]

[1]University of Tennessee Knoxville
[2]Oak Ridge National Laboratory
[3]MITRE
[4]High Performance Computing Center (HLRS)
[5]University Stuttgart, `www.hlrs.de/people/rabenseifner`
[6]Information Sciences Institute
[7]University of Southern California
[8]MIT Lincoln Lab
[9]IBM Austin
[10]Lawrence Berkeley National Laboratory
[11]University of Tsukuba

March 2005

Abstract

The HPC Challenge[1] benchmark suite has been released by the DARPA HPCS program to help define the performance boundaries of future Petascale computing systems. HPC Challenge is a suite of tests that examine the performance of HPC architectures using kernels with memory access patterns more challenging than those of the High Performance Linpack (HPL) benchmark used in the Top500 list. Thus, the suite is designed to augment the Top500 list, providing benchmarks that bound the performance of many real applications as a function of memory access characteristics e.g., spatial and temporal locality, and providing a framework for including additional tests. In particular, the suite is composed of several well known computational kernels (STREAM, HPL, matrix multiply – DGEMM, parallel matrix transpose – PTRANS, FFT, RandomAccess, and bandwidth/latency tests – b_eff) that attempt to span high and low spatial and temporal locality space. By design, the HPC Challenge tests are scalable with the size of data sets being a function of the largest HPL matrix for the tested system.

[1]This work was supported in part by the DARPA, NSF, and DOE through the DARPA HPCS program under grant FA8750-04-1-0219.

1 High Productivity Computing Systems

The DARPA High Productivity Computing Systems (HPCS) [1] is focused on providing a new generation of economically viable high productivity computing systems for national security and for the industrial user community. HPCS program researchers have initiated a fundamental reassessment of how we define and measure performance, programmability, portability, robustness and ultimately, productivity in the High End Computing (HEC) domain.

The HPCS program seeks to create trans-petaflops systems of significant value to the Government HPC community. Such value will be determined by assessing many additional factors beyond just theoretical peak flops (floating-point operations). Ultimately, the goal is to decrease the time-to-solution, which means decreasing both the execution time and development time of an application on a particular system. Evaluating the capabilities of a system with respect to these goals requires a different assessment process. The goal of the HPCS assessment activity is to prototype and baseline a process that can be transitioned to the acquisition community for 2010 procurements. As part of this effort we are developing a scalable benchmark for the HPCS systems.

The basic goal of performance modeling is to measure, predict, and understand the performance of a computer program or set of programs on a computer system. The applications of performance modeling are numerous, including evaluation of algorithms, optimization of code implementations, parallel library development, and comparison of system architectures, parallel system design, and procurement of new systems.

This paper is organized as follows: sections 2 and 3 give motivation and overview of HPC Challenge while sections 4 and 5 provide more detailed description of the HPC Challenge tests and section 6 talks briefly about the scalibility of the tests; sections 7, 8, and 9 describe the rules, the software installation process and some of the current results, respectively. Finally, section 10 concludes the paper.

2 Motivation

The DARPA High Productivity Computing Systems (HPCS) program has initiated a fundamental reassessment of how we define and measure performance, programmability, portability, robustness and, ultimately, productivity in the HPC domain. With this in mind, a set of computational kernels was needed to test and rate a system. The HPC Challenge suite of benchmarks consists of four local (matrix-matrix multiply, STREAM, RandomAccess and FFT) and four global (High Performance Linpack – HPL, parallel matrix transpose – PTRANS, RandomAccess and FFT) kernel benchmarks. HPC Challenge is designed to approximately bound computations of high and low spatial and temporal locality (see Figure 1). In addition, because HPC Challenge kernels consist of simple mathematical operations, this provides a unique opportunity to look at language and parallel programming model issues. In the end, the benchmark is to serve both the system user and designer communities [2].

3 The Benchmark Tests

This first phase of the project have developed, hardened, and reported on a number of benchmarks. The collection of tests includes tests on a single processor (local) and tests over the complete system (global). In particular, to characterize the architecture of the system we consider three testing scenarios:

1. Local – only a single processor is performing computations.

2. Embarrassingly Parallel – each processor in the entire system is performing computations but they do not communicate with each other explicitly.

3. Global – all processors in the system are performing computations and they explicitly communicate with each other.

The HPC Challenge benchmark consists at this time of 7 performance tests: HPL [3], STREAM [4], RandomAccess, PTRANS, FFT (implemented using FFTE [5]), DGEMM [6, 7] and b_eff (MPI latency/bandwidth test) [8, 9, 10]. HPL is the Linpack TPP (toward peak performance) benchmark. The test stresses the floating point performance of a system. STREAM is a benchmark that measures sustainable memory bandwidth (in Gbyte/s), RandomAccess measures the rate of random updates of memory. PTRANS measures the rate of transfer for large arrays of data from multiprocessor's memory. Latency/Bandwidth measures (as the name suggests) latency and bandwidth of communication patterns of increasing complexity between as many nodes as is time-wise feasible.

Many of the aforementioned tests were widely used before HPC Challenge was created. At first, this may seemingly make our benchmark merely a packaging effort. However, almost all components of HPC Challenge were augmented from their original form to provide consistent verification and reporting scheme. We should also stress the importance of running these very tests on a single machine and have the results available at once. The tests were useful separately for the HPC community before and with the unified HPC Challenge framework they create an unprecedented view of performance characterization of a system – a comprehensive view that captures the data under the same conditions and allows for variety of analysis depending on end user needs.

Each of the included tests examines system performance for various points of the conceptual spatial and temporal locality space shown in Figure 1. The rationale for such selection of tests is to measure performance bounds on metrics important to HPC applications. The expected behavior of the applications is to go through various locality space points during runtime. Consequently, an application may be represented as a point in the locality space being an average (possibly time-weighed) of its various locality behaviors. Alternatively, a decomposition can be made into time-disjoint periods in which the application exhibits a single locality characteristic. The application's performance is then obtained by combining the partial results from each period.

Another aspect of performance assessment addressed by HPC Challenge is ability to optimize benchmark code. For that we allow two different runs to be reported:

- Base run done with provided reference implementation.
- Optimized run that uses architecture specific optimizations.

The base run, in a sense, represents behavior of legacy code because it is conservatively written using only widely available programming languages and libraries. It reflects a commonly used approach to parallel processing sometimes referred to as hierarchical parallelism that combines Message Passing Interface (MPI) with threading from OpenMP. At the same time we recognize the limitations of the base run and hence we allow (or even encourage) optimized runs to be made. The optimizations may include alternative implementations in different programming languages using parallel environments available specifically on the tested system. To stress the productivity aspect of the HPC Challenge benchmark, we require that the information about the changes made to the original code be submitted together with the benchmark results. While we understand that full disclosure of optimization techniques may sometimes be impossible to obtain (due to for example trade secrets) we ask at least for some guidance for the users that would like to use similar optimizations in their applications.

4 Benchmark Details

Almost all tests included in our suite operate on either matrices or vectors. The size of the former we will denote below as n and the latter as m. The following holds throughout the tests:

$$n^2 \simeq m \simeq \text{Available Memory}$$

Or in other words, the data for each test is scaled so that the matrices or vectors are large enough to fill almost all available memory.

HPL (High Performance Linpack) is an implementation of the Linpack TPP (Toward Peak Performance) variant of the original Linpack benchmark which measures the floating point rate of execution for solving a linear system of equations. HPL solves a linear system of equations of order n:

$$Ax = b; \quad A \in \mathbf{R}^{n\times n};\ x, b \in \mathbf{R}^n$$

by first computing LU factorization with row partial pivoting of the n by $n+1$ coefficient matrix:

$$P[A,b] = [[L,U],y].$$

Since the row pivoting (represented by the permutation matrix P) and the lower triangular factor L are applied to b as the factorization progresses, the solution x is obtained in one step by solving the upper triangular system:

$$Ux = y.$$

The lower triangular matrix L is left unpivoted and the array of pivots is not returned. The operation count for the factorization phase is $\frac{2}{3}n^3 - \frac{1}{2}n^2$ and $2n^2$ for the solve phase. Correctness of the solution is ascertained by calculating the scaled residuals:

$$\frac{\|Ax-b\|_\infty}{\varepsilon\|A\|_1 n}, \quad \frac{\|Ax-b\|_\infty}{\varepsilon\|A\|_1\|x\|_1}, \quad \text{and} \quad \frac{\|Ax-b\|_\infty}{\varepsilon\|A\|_\infty\|x\|_\infty},$$

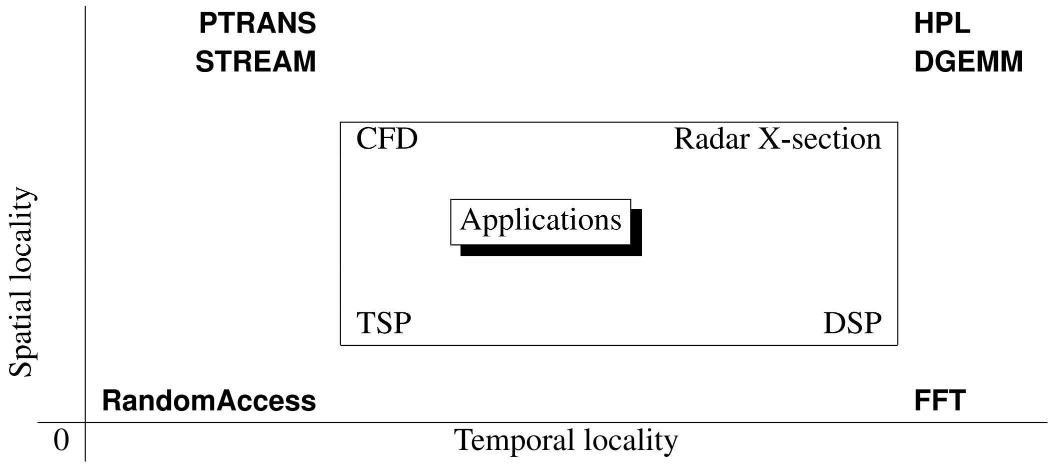


Figure 1: Targeted application areas in the memory access locality space.

where ε is machine precision for 64-bit floating-point values and n is the size of the problem.

DGEMM measures the floating point rate of execution of double precision real matrix-matrix multiplication. The exact operation performed is:

$$C \leftarrow \beta\, C + \alpha\, AB$$

where:

$$A, B, C \in \mathbf{R}^{n\times n}; \quad \alpha, \beta \in \mathbf{R}^{n}.$$

The operation count for the multiply is $2n^3$ and correctness of the operation is ascertained by calculating the scaled residual: $\|C - \hat{C}\|/(\varepsilon\, n \|C\|_F)$ ($\hat{C}$ is a result of reference implementation of the multiplication).

STREAM is a simple benchmark program that measures sustainable memory bandwidth (in Gbyte/s) and the corresponding computation rate for four simple vector kernels:

$$\begin{aligned} \text{COPY:} \quad c &\leftarrow \quad a \\ \text{SCALE:} \quad b &\leftarrow \quad \alpha\, c \\ \text{ADD:} \quad c &\leftarrow \quad a + b \\ \text{TRIAD:} \quad a &\leftarrow \quad b + \alpha\, c \end{aligned}$$

where:

$$a, b, c \in \mathbf{R}^{m}; \quad \alpha \in \mathbf{R}.$$

As mentioned earlier, we try to operate on large data objects. The size of these objects is determined at runtime which contrasts with the original version of the STREAM benchmark which uses static storage (determined at compile time) and size. The original benchmark gives the compiler more information (and control) over data alignment, loop trip counts, etc. The benchmark measures Gbyte/s and the number of items transferred is either $2m$ or $3m$ depending on the operation. The norm of the difference between reference and computed vectors is used to verify the result: $\|x - \hat{x}\|$.

PTRANS (parallel matrix transpose) exercises the communications where pairs of processors exchange large messages simultaneously. It is a useful test of the total communications capacity of the system interconnect. The performed operation sets a random n by n matrix to a sum of its transpose with another random matrix:

$$A \leftarrow A^T + B$$

where:

$$A, B \in \mathbf{R}^{n\times n}.$$

The data transfer rate (in Gbyte/s) is calculated by dividing the size of n^2 matrix entries by the time it took to perform the transpose. The scaled residual of the form $\|A - \hat{A}\|/(\varepsilon\, n)$ verifies the calculation.

RandomAccess measures the rate of integer updates to random memory locations (GUPS). The operation being performed on an integer array of size m is:

$$x \leftarrow f(x)$$

$$f : x \mapsto (x \oplus a_i); \quad a_i\text{– pseudo-random sequence}$$

where:

$$f : \mathbf{Z}^m \to \mathbf{Z}^m; \quad x \in \mathbf{Z}^m.$$

The operation count is $4m$ and since all the operations are in integral values over GF(2) field they can be checked exactly with a reference implementation. The verification procedure allows 1% of the operations to be incorrect (skipped) which allows loosening concurrent memory update semantics on shared memory architectures.

FFT measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT) of size m:

$$Z_k \leftarrow \sum_j^m z_j e^{-2\pi i \frac{jk}{m}}; \quad 1 \le k \le m$$

where:

$$z, Z \in \mathbf{C}^m.$$

The operation count is taken to be $5m\log_2 m$ for the calculation of the computational rate (in Gflop/s). Verification is done with a residual $\|x-\hat{x}\|/(\varepsilon \log m)$ where $\hat{x}$ is the result of applying a reference implementation of inverse transform to the outcome of the benchmarked code (in infinite-precision arithmetic the residual should be zero).

Communication bandwidth and latency is a set of tests to measure latency and bandwidth of a number of simultaneous communication patterns. The patterns are based on b_eff (effective bandwidth benchmark) – they are slightly different from the original b_eff. The operation count is linearly dependent on the number of processors in the tested system and the time the tests take depends on the parameters of the tested network. The checks are built into the benchmark code by checking data after it has been received.

5 Latency and Bandwidth Benchmark

The latency and bandwidth benchmark measures two different communication patterns. First, it measures the single-process-pair latency and bandwidth, and second, it measures the parallel all-processes-in-a-ring latency and bandwidth.

For the first pattern, ping-pong communication is used on a pair of processes. Several different pairs of processes are used and the maximal latency and minimal bandwidth over all pairs is reported. While the ping-pong benchmark is executed on one process pair, all other processes are waiting in a blocking receive. To limit the total benchmark time used for this first pattern to 30 sec, only a subset of the set of possible pairs is used. The communication is implemented with MPI standard blocking send and receive.

In the second pattern, all processes are arranged in a ring topology and each process sends and receives a message from its left and its right neighbor in parallel. Two types of rings are reported: a naturally ordered ring (i.e., ordered by the process ranks in `MPI_COMM_WORLD`), and the geometric mean of the bandwidth of ten different randomly chosen process orderings in the ring. The communication is implemented (a) with MPI standard non-blocking receive and send, and (b) with two calls to MPI_Sendrecv for both directions in the ring. Always the fastest of both measurements are used. With this type of parallel communication, the bandwidth per process is defined as total amount of message data divided by the number of processes and the maximal time needed in all processes. This benchmark is based on patterns studied in the effective bandwidth communication benchmark [8, 9].

For benchmarking latency and bandwidth, 8 byte and 2,000,000 byte long messages are used. The major results reported by this benchmark are:

- maximal ping pong latency,
- average latency of parallel communication in randomly ordered rings,
- minimal ping pong bandwidth,
- bandwidth per process in the naturally ordered ring,
- average bandwidth per process in randomly ordered rings.

Additionally reported values are the latency of the naturally ordered ring, and the remaining values in the set of minimum, maximum, and average of the ping-pong latency and bandwidth.

Especially the ring based benchmarks try to model the communication behavior of multi-dimensional domain-decomposition applications. The natural ring is similar to the message transfer pattern of a regular grid based application, but only in the first dimension (adequate ranking of the processes is assumed). The

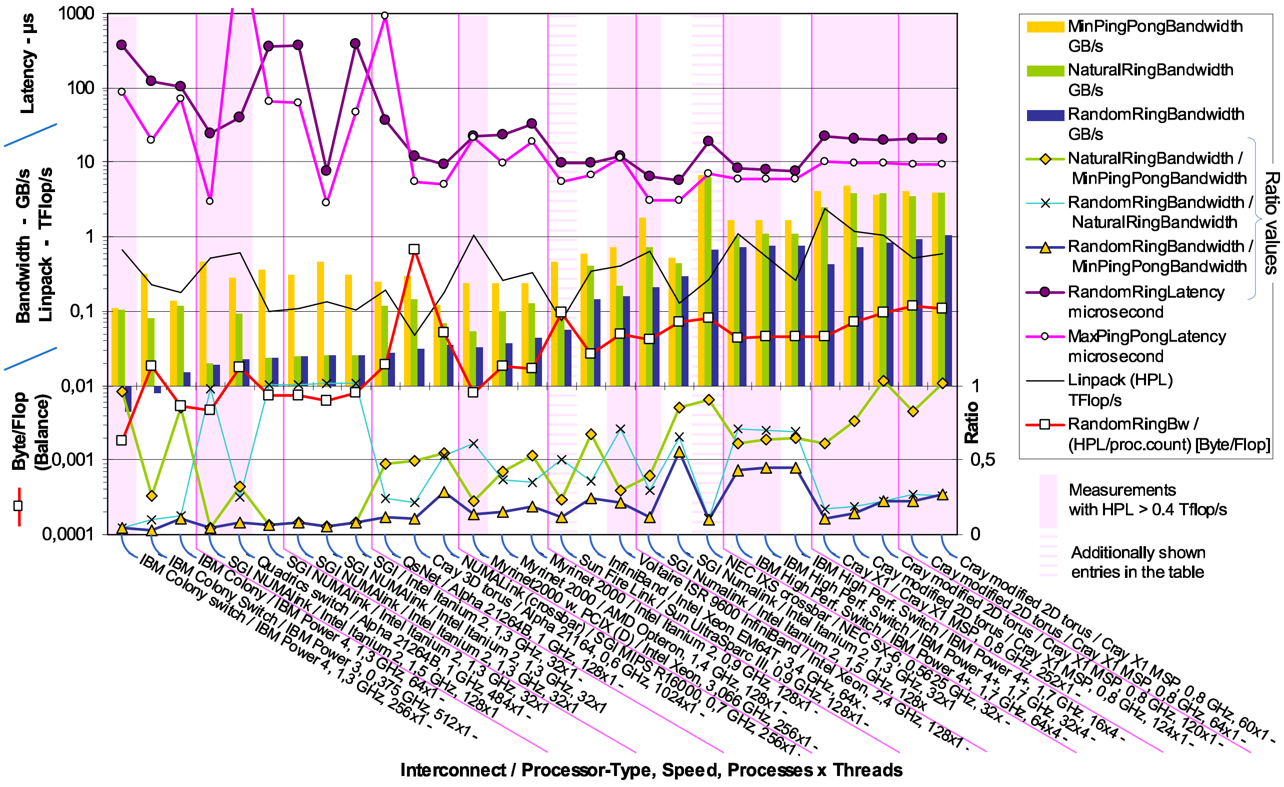


Figure 2: Base runs of the HPC Challenge bandwidth and latency benchmarks. Status Oct. 12, 2004.

random ring fits to the other dimensions and to the communication pattern of unstructured grid based applications. Therefore, the analysis in Fig. 2 is mainly focused on the random ring bandwidth. The measurements are sorted by this value, except that all Cray X1 and Myrinet measurements are kept together at the position of their best bandwidth.

The diagram consists of three bands: 1) the ping-pong and random ring latencies, 2) the minimal ping-pong, natural ring, and random ring bandwidth-bars together with a background curve showing the accumulated Linpack (HPL) performance, and 3) the ratios *natural ring* to *ping-pong*, *random ring* to *ping-pong*, and additionally *random ring* to *natural ring*.

Looking on the three interconnects with a random ring bandwidth larger than 500 Mbyte/s, one can see that all 3 systems are reporting a similar random ring bandwidth (except with largest CPU count on Cray X1), although the ping pong and natural ring bandwidth values are quite different. The ratio natural ring to ping-ping bandwidth varies between 0.6 and 1.0, random ring to ping-pong between 0.1 and 0.45, and random to natural ring between 0.1 and 0.7. With the IBM High Performance Switch (HPS), the reported bandwidth values (0.72-0.75 Gbyte/s) are nearly independent from the number of processes (64 to 256 [with 1.07 Tflop/s]), while the Cray X1 shows a degradation from 1.03 Gbyte/s with 60 MSPs (0.58 Tflop/s) to 0.43 Gbyte/s with 252 MSPs (2.38 Tflop/s). As of October 12, 2004, values for larger NEC systems are still missing.

If we are looking at the high-lighted systems with more than 0.4 Tflop/s accumulated Linpack performance, the random ring latency and performance is summarized in Table 1.

For the bandwidth values, the achievable percentage on the random ring from the ping-pong varies between 4 % and 45 %. For the latency values, the ratio ping-pong to random varies between 0.12 and 0.99.

These examples should show the communication performance of different network types, but also that the ping-pong values are not enough for a comparison. The ring based benchmark results are needed to analyze these interconnects.

Switch	CPU	per SMP:	Proc.	Number	**Random**	Ping-	Rand.	Ping-	**HPL Linpack**		**Balance:**
		#netw.	Speed	of MPI	**Ring**	Pong	Ring	Pong	accumu-	**per**	**Communi./**
		adap's/		processes	**Bandw.**	Bandw.	Lat.	Lat.	lated	**process**	**Comput.**
		#CPUs	GHz	x threads	**Gbyte/s**	Gbyte/s	μs	μs	Gflop/s	**Gflop/s**	**byte/kflop**
IBM Colony	IBM Power 4		1.3	256x1	**0.0046**	0.108	374	87	654	**2.55**	**1.8**
Quadrics switch	Alpha 21264B		1.0	484x1	**0.023**	0.280	40		618	**1.28**	**17.8**
Myrinet 2000	Intel Xeon 3		3.066	256x1	**0.032**	0.241	22	22	1030	**4.02**	**8.1**
Sun Fire Link	Ultra Sparc III		0.9	128x1	**0.056**	0.468	9	5	75	**0.59**	**94.5**
Infiniband	Intel Xeon		2.46	128x1	**0.156**	0.738	12	12	413	**3.23**	**48.2**
SGI Numalink	Intel Itanium 2		1.56	128x1	**0.211**	1.8	6	3	639	**4.99**	**42.2**
NEC SX-6 IXS	NEC SX-6	1/8	0.5625	32x1	**0.672**	6.8	19	7	268	**8.37**	**80.3**
NEC SX-6 IXS^{+})	NEC SX-6	1/8	0.5625	4x8	**6.759**	7.0	8	6	(268)	**(66.96)**	**(100.9)**
IBM HPS	IBM Power 4+		1.7	64x4	**0.724**	1.7	8	6	1074	**16.79**	**43.1**
IBM HPS	IBM Power 4+		1.7	32x4	**0.747**	1.7	8	6	532	**16.62**	**45.0**
Cray X1	Cray X1 MSP		0.8	252x1	**0.429**	4.0	22	10	2385	**9.46**	**45.3**
Cray X1	Cray X1 MSP		0.8	124x1	**0.709**	4.9	20	10	1205	**9.72**	**72.9**
Cray X1	Cray X1 MSP		0.8	120x1	**0.830**	3.7	20	10	1061	**8.84**	**93.9**
Cray X1	Cray X1 MSP		0.8	64x1	**0.941**	4.2	20	9	522	**8.15**	**115.4**
Cray X1	Cray X1 MSP		0.8	60x1	**1.033**	3.9	21	9	578	**9.63**	**107.3**

Table 1: Comparison of bandwidth and latency on HPCC entries with more than 0.4 Tflop/s with two exceptions: For SGI Numalink, only MPT 1.10 values are shown, the older MPT 1.8-1 values are omitted, and for Sun Fire and NEC SX-6, smaller systems are reported because on larger systems, HPCC results are not yet available. Note, that each thread is running on a *CPU*, but the communication and the second HPL value are measured with *MPI processes*.
$^{+}$) This row is based on an additional measurement with the communication benchmark software. The HPL value of this row is taken from the previous row because there isn't a benchmark value available and significant differences between single- and multi-threaded HPL execution are not expected. The last two columns are based on this HPL value.

5.1 Balance of Communication to Computation

For multi-purpose HPC systems, the balance of processor speed, and memory, communication, and I/O bandwidth is important. In this section, we analyze the ratio of the inter-node communication bandwidth to the computational speed. To characterize the communication bandwidth between SMP nodes, we use the random ring bandwidth, because for a large number of SMP nodes, most MPI processes will communicate with MPI processes on other SMP nodes. This means, with 8 or more SMP nodes, the random ring bandwidth reports the available inter-node communication bandwidth per MPI process. To characterize the computational speed, we use the HPL Linpack benchmark value divided by the number of MPI processes, because HPL can achieve nearly peak on cache-based and on vector systems, and with single- and multi-threaded execution. The ratio of the random ring bandwidth to the HPL divided by the MPI process count expresses the communication-computation balance in byte/flop (see Figure 2) or byte/kflop (used in Table 1).

Although the balance is calculated based on MPI processes, its value should be in principle independent of the programming model, i.e. whether each SMP node is used with several single-threaded MPI processes, or some (or one) multi-threaded MPI processes, as long as the number of MPI processes on each SMP node is large enough that they altogether are able to saturate the inter-node network [10].

Table 1 shows that the balance is quite different. Currently, the HPCC table lacks the information, how many network adapters are used on each of the SMP nodes, i.e., the balance may be different if a system is measured with the exactly same interconnect and processors but a smaller or larger amount of network adapters per SMP node.

On the reported installations, the balance values start with 1.8 / 8.1 / 17.8 byte/kflop on IBM Colony, Myrinet 2000 and Quadrics. SGI Numalink, IBM High Performance Switch, Infiniband, and the largest Cray X1 configuration have a balance between 40 and 50 byte/kflop. Highest balance values are available on Sun Fire Link (but only with 0.59 Gflops per MPI process), on NEC SX-6 and on Cray X1, here with up to 115 byte/kflop.

For NEC SX-6, the two different programming models *single-* and *multi-threaded* execution were used. With the single-threaded execution, 25 % of the random ring connections involve only intra-node communications. Therefore only 0.504 Gbyte/s (75 % from 0.672 Gbyte/s) represent the inter-node communication bandwidth per CPU. The inter-node bandwidth per node (with 8 CPUs) is therefore 4.02 Gbyte/s respectively. The balance of inter-node communication to computation is characterized by the reduced value 60.2 byte/kflop. With multi-threaded execution, all communication is done by the master-threads and is inter-node communication. Therefore, the random ring bandwidth is measured per node. It is significantly better with the multi-threaded application programming scheme (6.759 Gbyte/s) than with single-threaded (4.02 Gbyte/s). Implications on optimal programming models are discussed in [10].

6 On Scalability of Benchmarks with Scalable Input Data

6.1 Notation

- P – number of CPUs
- M – total size of system memory
- r – rate of execution (unit: Gflop/s, GUPS, etc.)
- t – time
- N – size of global matrix for HPL and PTRANS
- V – size of global vector for RandomAccess and FFT

6.2 Assumptions

Memory size per CPU is constant as the system grows. This assumption based on architectural design of almost all systems in existence. Hence, the total amount of memory available in the entire system is linearly proportional to the number of processors.

For HPL the dominant cost is CPU-related because computation has higher complexity order than communication: $\mathcal{O}(n^3)$ versus $\mathcal{O}(n^2)$.

6.3 Theory

Time complexity for HPL is $\mathcal{O}(n^3)$ (the hidden constant is $\frac{2}{3}$) so the time is:

$$t_{\mathsf{HPL}} \propto \frac{N^3}{r_{\mathsf{HPL}}} \tag{1}$$

Since N is a size of a square matrix then we need to take square root of available memory:

$$N \propto \sqrt{M} \propto \sqrt{P}$$

The rate of execution for HPL is determined by the number of processors since computations (rather than communication) dominates in terms of complexity:

$$r_{\mathsf{HPL}} \propto P$$

That leads to:

$$t_{\mathsf{HPL}} \propto \sqrt{P}$$

Time complexity for RandomAccess is $\mathcal{O}(n)$ so the time is:

$$t_{\mathsf{RandomAcccess}} \propto \frac{V}{r_{\mathsf{RandomAcccess}}} \tag{2}$$

The main table for RandomAccess should be as big as half of the total memory, so we get:

$$V \propto M \propto P$$

The rate of execution for RandomAccess can be argued to have various forms:

- If we assume that the interconnect scales with the number of processors then the rate would also scale:
$$r_{\mathsf{RandomAcccess}} \propto P$$
- Real-life experience tells that $r_{\mathsf{RandomAcccess}}$ is dependent on the interconnect and independent of the number of processors due to interconnect inefficiency:
$$r_{\mathsf{RandomAcccess}} \propto \mathbf{1}$$
Even worse, it is also conceivable that the rate is a decreasing function of processors:
$$r_{\mathsf{RandomAcccess}} \propto \frac{1}{P}$$

Conservatively assuming that $r_{\mathsf{RandomAcccess}} \propto \mathbf{1}$ it follows that:

$$t_{\mathsf{RandomAcccess}} \propto P$$

Manufacturer	Processor	Interconnect	GUPS [GUPS]	flops/cycle	frequency [GHz]	Mem/CPU [GiB]	SMP CPUs
IBM	Power 4	Federation	0.002628189	4	1.7	1	32
Cray	X1	2D torus	0.145440128	16	0.8	4	4
Atipa	Opteron	Myrinet 2000	0.003195448	2	1.4	1	2
SGI	Itanium 2	NUMAlink	0.003053927	4	1.5	8	4
Voltaire	Xeon	InfiniBand	0.000650427	2	2.4	1	2

Table 2: System used in tests. The RandomAccess number is not necessarily accurate.

6.4 Scalability Tests

Tested systems are in Table 2.

Table 3 shows estimates of time it takes to run HPL and RandomAccess. For a 256-CPU system RandomAccess takes much longer to run than HPL (except for Cray X1 which has large amount of memory per CPU which makes for longer HPL run and large GUPS number which makes for short RandomAccess run).

7 Rules for Running the Code

There must be one baseline run submitted for each computer system entered in the archive. There may also exist an optimized run for each computer system.

1. *Baseline Runs*
   Optimizations as described below are allowed.

   (a) *Compile and load options*
   Compiler or loader flags which are supported and documented by the supplier are allowed. These include porting, optimization, and preprocessor invocation.

   (b) *Libraries*
   Linking to optimized versions of the following libraries is allowed:

   - BLAS
   - MPI

   Acceptable use of such libraries is subject to the following rules:

   - All libraries used shall be disclosed with the results submission. Each library shall be identified by library name, revision, and source (supplier). Libraries which are not generally available are not permitted unless they are made available by the reporting organization within 6 months.
   - Calls to library subroutines should have equivalent functionality to that in the released benchmark code. Code modifications to accommodate various library call formats are not allowed.
   - Only complete benchmark output may be submitted – partial results will not be accepted.

2. *Optimized Runs*

   (a) *Code modification*
   Provided that the input and output specification is preserved, the following routines may be substituted:

   - In HPL: `HPL_pdgesv()`, `HPL_pdtrsv()` (factorization and substitution functions)
   - no changes are allowed in the DGEMM component
   - In PTRANS: `pdtrans()`
   - In STREAM: `tuned_STREAM_Copy()`, `tuned_STREAM_Scale()`, `tuned_STREAM_Add()`, `tuned_STREAM_Triad()`
   - In RandomAccess: `MPIRandomAccessUpdate()` and `RandomAccessUpdate()`
   - In FFT: `fftw_malloc()`, `fftw_free()`, `fftw_one()`, `fftw_mpi()`, `fftw_create_plan()`,

System	64 CPUs		256 CPUs	
	HPL	RandomAccess	HPL	RandomAccess
Power4	29.08	108.9	58.1	435.7
CrayX1	123.6	7.8	247.2	31.4
Opteron	70.6	89.6	141.2	358.4
Itanium 2	745.9	750.0	1491.9	3000.2
Xeon	41.2	440.2	82.4	1760.8

Table 3: Estimated time in minutes to perform full system test of HPL and RandomAccess.

`fftw_destroy_plan()`, `fftw_mpi_create_plan()`, `fftw_mpi_local_sizes()`, `fftw_mpi_destroy_plan()` (all of these functions are compatible with FFTW 2.1.5 [11, 12])

- In b_eff component alternative MPI routines might be used for communication. But only standard MPI calls are to be performed and only to the MPI library that is widely available on the tested system.

(b) *Limitations of Optimization*

i. *Code with limited calculation accuracy*
The calculation should be carried out in full precision (64-bit or the equivalent). However the substitution of algorithms is allowed (see next).

ii. *Exchange of the used mathematical algorithm*
Any change of algorithms must be fully disclosed and is subject to review by the HPC Challenge Committee. Passing the verification test is a necessary condition for such an approval. The substituted algorithm must be as robust as the baseline algorithm. For the matrix multiply in the HPL benchmark, Strassen Algorithm may not be used as it changes the operation count of the algorithm.

iii. *Using the knowledge of the solution*
Any modification of the code or input data sets, which uses the knowledge of the solution or of the verification test, is not permitted.

iv. *Code to circumvent the actual computation*
Any modification of the code to circumvent the actual computation is not permitted.

8 Software Download, Installation, and Usage

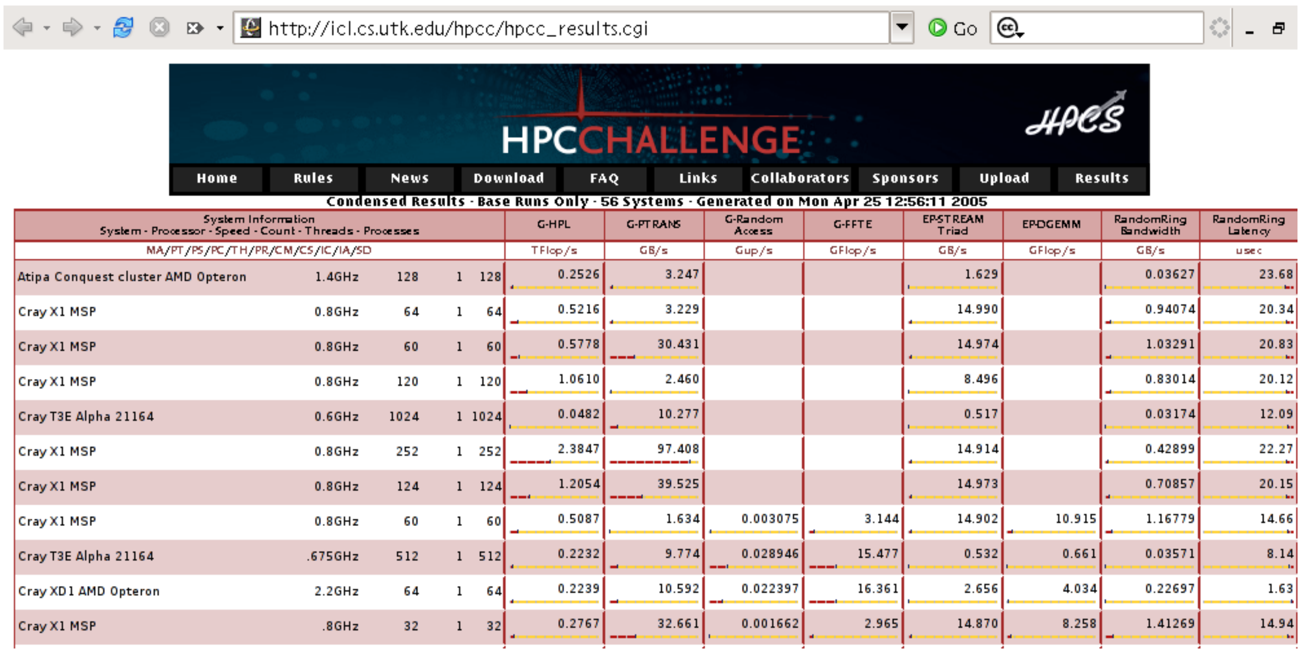
The reference implementation of the benchmark may be obtained free of charge at the benchmark's web site: `http://icl.cs.utk.edu/hpcc/`. The reference implementation should be used for the base run. The installation of the software requires creating a script file for Unix's `make(1)` utility. The distribution archive comes with script files for many common computer architectures. Usually, few changes to one of these files will produce the script file for a given platform.

After, a successful compilation the benchmark is ready to run. However, it is recommended that changes are made to the benchmark's input file such that the sizes of data to use during the run are appropriate for the tested system. The sizes should reflect the available memory on the system and number of processors available for computations.

We have collected a comprehensive set of notes on the HPC Challenge benchmark. They can be found at `http://icl.cs.utk.edu/hpcc/faq/`.

9 Example Results

Figure 3 shows a sample rendering of the results web page: `http://icl.cs.utk.edu/hpcc/hpcc_results.cgi`. It is impossible to show here all of the results for nearly 60 systems submitted so far to the



Condensed Results - Base Runs Only - 56 Systems - Generated on Mon Apr 25 12:56:11 2005

System Information System - Processor - Speed - Count - Threads - Processes					G-HPL	G-PTRANS	G-Random Access	G-FFTE	EP-STREAM Triad	EP-DGEMM	RandomRing Bandwidth	RandomRing Latency
MA/PT/PS/PC/TH/PR/CM/CS/IC/IA/SD					TFlop/s	GB/s	Gup/s	GFlop/s	GB/s	GFlop/s	GB/s	usec
Atipa Conquest cluster AMD Opteron	1.4GHz	128	1	128	0.2526	3.247			1.629		0.03627	23.68
Cray X1 MSP	0.8GHz	64	1	64	0.5216	3.229			14.990		0.94074	20.34
Cray X1 MSP	0.8GHz	60	1	60	0.5778	30.431			14.974		1.03291	20.83
Cray X1 MSP	0.8GHz	120	1	120	1.0610	2.460			8.496		0.83014	20.12
Cray T3E Alpha 21164	0.6GHz	1024	1	1024	0.0482	10.277			0.517		0.03174	12.09
Cray X1 MSP	0.8GHz	252	1	252	2.3847	97.408			14.914		0.42899	22.27
Cray X1 MSP	0.8GHz	124	1	124	1.2054	39.525			14.973		0.70857	20.15
Cray X1 MSP	0.8GHz	60	1	60	0.5087	1.634	0.003075	3.144	14.902	10.915	1.16779	14.66
Cray T3E Alpha 21164	.675GHz	512	1	512	0.2232	9.774	0.028946	15.477	0.532	0.661	0.03571	8.14
Cray XD1 AMD Opteron	2.2GHz	64	1	64	0.2239	10.592	0.022397	16.361	2.656	4.034	0.22697	1.63
Cray X1 MSP	.8GHz	32	1	32	0.2767	32.661	0.001662	2.965	14.870	8.258	1.41269	14.94

Figure 3: Sample results page.

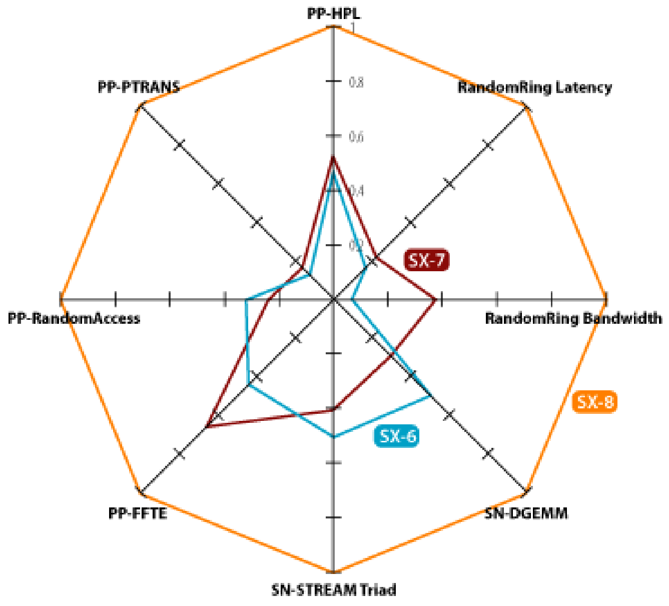


Figure 4: Sample kiviat diagram of results for three generations of hardware from the same vendor (NEC).

web site. The results database is publicly available at the aforementioned address and can be exported to Excel spreadsheet or an XML file. Figure 4 shows a sample kiviat diagram generated using the benchmark results. Kiviat diagrams can be generated at the website and allow easy comparitive analysis for multi-dimensional results from the HPC Challenge database.

10 Conclusions

No single test can accurately compare the performance of HPC systems. The HPC Challenge benchmark test suite stresses not only the processors, but the memory system and the interconnect. It is a better indicator of how an HPC system will perform across a spectrum of real-world applications. Now that the more comprehensive, informative HPC Challenge benchmark suite is available, it can be used in preference to comparisons and rankings based on single tests. The real utility of the HPC Challenge benchmarks are that architectures can be described with a wider range of metrics than just flop/s from HPL. When looking only at HPL performance and the Top500 List, inexpensive build-your-own clusters appear to be much more cost effective than more sophisticated HPC architectures. Even a small percentage of random memory accesses in real applications can significantly affect the overall performance of that application on architectures not designed to minimize or hide memory latency. HPC Challenge benchmarks provide users with additional information to justify policy and purchasing decisions. We expect to expand and perhaps remove some existing benchmark components as we learn more about the collection.

11 Acknowledgments

The authors would like to thank Gerrit Schulz and Michael Speck, student co-workers, who have implemented parts of the software.

References

[1] High Productivity Computer Systems. (`http://www.highproductivity.org/`).

[2] William Kahan. The baleful effect of computer benchmarks upon applied mathematics, physics and chemistry. The John von Neumann Lecture at the 45th Annual Meeting of SIAM, Stanford University, 1997.

[3] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15:1–18, 2003.

[4] John McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. (`http://www.cs.virginia.edu/stream/`).

[5] Daisuke Takahashi and Yasumasa Kanada. High-performance radix-2, 3 and 5 parallel 1-D complex FFT algorithms for distributed-memory parallel computers. *The Journal of Supercomputing*, 15(2):207–228, 2000.

[6] Jack J. Dongarra, J. Du Croz, Iain S. Duff, and S. Hammarling. Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, March 1990.

[7] Jack J. Dongarra, J. Du Croz, Iain S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:18–28, March 1990.

[8] Alice E. Koniges, Rolf Rabenseifner, and Karl Solchenbach. Benchmark design for characterization of balanced high-performance architectures. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01), Workshop on Massively Parallel Processing (WMPP)*, volume 3, San Francisco, CA, April 23-27 2001. In IEEE Computer Society Press.

[9] Rolf Rabenseifner and Alice E. Koniges. Effective communication and file-i/o bandwidth benchmarks. In *J. Dongarra and Yiannis Cotronis (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proceedings of the 8th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2001*, pages 24–35, Santorini, Greece, September 23-26 2001. LNCS 2131.

[10] Rolf Rabenseifner. Hybrid parallel programming on HPC platforms. In *Proceedings of the Fifth European Workshop on OpenMP, EWOMP '03*, pages 185–194, Aachen, Germany, September 22-26 2003.

[11] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.

[12] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2), 2005. special issue on "Program Generation, Optimization, and Adaptation".