Chapter in 'Contemporary HPC Architectures'

Contents

Chapter 1

HPC Challenge: Design, History, and Implementation Highlights

Jack Dongarra

University of Tennesse

Piotr Luszczek

University of Tennesse

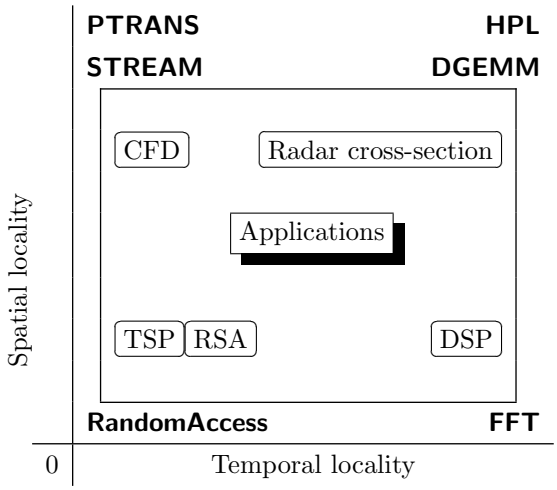


FIGURE 1.1: The application areas targeted by the HPCS Program are bound by the HPCC tests in the memory access locality space.

1.1 Introduction

The HPC Challenge (HPCC)[1] benchmark suite was initially developed for the DARPA's HPCS program [Kep04] to provide a set of standardized hardware probes based on commonly occurring computational software kernels. The HPCS program has initiated a fundamental reassessment of how we define and measure performance, programmability, portability, robustness and, ultimately, productivity in the high-end domain. Consequently, the suite was aimed to both provide conceptual expression of the underlying computation as well as be applicable to a broad spectrum of computational science fields. Clearly, a number of compromises must have lead to the current form of the suite given such a broad scope of design requirements. HPCC was designed to approximately bound computations of high and low spatial and temporal locality (see Figure 1.1 which gives the conceptual design space for the HPCC component tests). In addition, because the HPCC tests consist of simple mathematical operations, this provides a unique opportunity to look at language

[1]This work was supported in part by the DARPA, NSF, and DOE through the DARPA HPCS program under grant FA8750-04-1-0219 and SCI-0527260.

HPCS Program

Memory Hierarchy	Benchmarks	Performance Targets	Required Improvement
Registers			
Operands ⇳	HPL	2 Pflop/s	800%
Cache			
Lines ⇳	STREAM	2 PB/s	4000%
Local Memory			
⇳	FFT	0.5 Pflop/s	20000%
Pages ⇳	RandomAccess	64000 GUPS	200000%
⇳	b_eff		
Remote Memory			
Buffers ⇳			
Disk			

FIGURE 1.2: HPCS program benchmarks and performance targets.

and parallel programming model issues. As such, the benchmark is to serve both the system user and designer communities [Kah97].

Finally, Figure 1.2 shows a generic memory subsystem and how each level of the hierarchy is tested by the HPCC software and what are the design goals of the future HPCS system – these are the projected target performance numbers that are to come out of the wining HPCS vendor designs.

1.2 The TOP500 Influence

Most commonly known ranking of supercomputer installations around the world is the TOP500 list [MSDS06]. It uses the equally famous LINPACK Benchmark [DLP03] as a single figure of merit to rank 500 of the world's most powerful supercomputers. The often raised issue of the relation between TOP500 and HPCC can simply be addressed by recognizing all the positive aspects of the former. In particular, the longevity of TOP500 gives an un-

Rank	Name	*Rmax*	HPL	PTRANS	STREAM	FFT	RandomAccess	Latency	Bandwidth
1	BlueGene/L	280.6	259.2	4665.9	160	2311	35.47	5.92	0.16
2	BlueGene W	91.3	83.9	171.5	50	1235	21.61	4.70	0.16
3	ASC Purple	75.8	57.9	553.0	44	842	1.03	5.11	3.22
4	Columbia	51.9	46.8	91.3	21	230	0.25	4.23	1.39
9	Red Storm	36.2	33.0	1813.1	44	1118	1.02	7.97	1.15

TABLE 1.1: All of the top-10 entries of the 27^{th} TOP500 list that have results in the HPCC database.

precedented view of the high-end arena across the turbulent times of Moore's law [Moo65] rule and the process of emerging of today's prevalent computing paradigms. The predictive power of TOP500 will have a lasting influence in the future as it did in the past. While building on the legacy information, HPCC extends it the context of the HPCS goals and can already serve as a valuable tool for performance analysis. Table 1.1 shows an example of how the data from the HPCC database can augment the TOP500 results.

1.3 Short History of the Benchmark

The first reference implementation of the code was released to the public in 2003. Year 2004 marked two important milestones for the benchmark: 1 Tflop/s was exceeded on HPCC's HPL test and the first submission with over 1000 processors was recorded in the public submission database. The first optimized submission came in April 2004 from Cray using then recent X1 installation at the Oak Ridge National Laboratory. Ever since then Cray has championed the list of optimized submissions. By the time the first HPCC birds-of-feather at the Supercomputing conference in 2004 in Pittsburgh, the public database of results already featured major supercomputer makers – a sign that vendors noticed the benchmark. At the same time, a bit behind the scenes, the code was also tried by government and private institutions for procurement and marketing purposes.

At the time, Jack Dongarra described the goals of the HPC Challenge Benchmarks: "The HPC Challenge Benchmarks will examine the performance of HPC architectures using kernels with more challenging memory access patterns than just the High Performance LINPACK (HPL) benchmark used in the

TOP500 list. The HPC Challenge Benchmarks are being designed to augment the TOP500 list, provide benchmarks that bound the performance of many real applications as a function of memory access characteristics - e.g., spatial and temporal locality, and provide a framework for including additional benchmarks." HPCC is already up to par with the TOP500 in terms of HPL performance and it also offers a far richer view of today's High End Computing (HEC) landscape as well as giving an unprecedented array of performance metrics for various analyses and comparison studies.

The FFT test was introduced in version 0.6 in May 2004 and the first submission with the new test was recorded in July the same year. As of early October 2005, the fastest system in the database obtained nearly 1 Tflop/s in the Global FFT test (three orders of magnitude increase over time). At the same time, the fastest (in terms of HPL) system was listed at position 11 on June's edition of TOP500 list, but the result recorded in the HPCC database was four percentage points higher in terms of efficiency. Today all of these achievements have been superseded by submissions from TOP500's highest ranking machines including the number one entry.

Another highlight of 2005 was announcement of a contest: the HPCC Awards. The two complementary categories of the competition emphasized performance and productivity – the very goals of the sponsoring HPCS program. The performance-emphasizing Class 1 award draw attention of the biggest players in the supercomputing industry which resulted in populating the HPCC database with most of the top-10 entries of TOP500 (some of which even exceeding performance reported on TOP500 – a tribute to HPCC's continuous results' update policy). The contestants competed to achieve highest raw performance in one of the four tests: HPL, STREAM, RandomAccess, and FFT. The Class 2 award by solely focusing on productivity introduced subjectivity factor to the judging but also to the submitter criteria of what is appropriate for the contest. As a result a wide range of solutions were submitted spanning various programming languages (interpreted and compiled) and paradigms (with explicit and implicit parallelism). It featured openly available as well as proprietary technologies some of which were arguably confined to niche markets and some that are widely used. The financial incentives for entering turned out to be all but needed as the HPCC seemed to have enjoyed enough recognition among the high-end community. Nevertheless, HPCwire kindly provided both: press coverage as well as cash rewards for four winning contestants of Class 1 and the winner of Class 2. At the HPCC's second birds-of-feather session during the SC|05 conference in Seattle, the former class was dominated by IBM's BlueGene/L from Lawrence Livermore National Lab while the latter was split among MTA pragma-decorated C and UPC codes from Cray and IBM, respectively.

Over the years, HPCC has received exposure in numerous news outlets including Business Wire, Cnet, eWeek, HPCwire, and Yahoo!. The website often receives over 100,000 hits per month and the source code download rates exceed 1,000 downloads per year. A different kind of publicity comes

from the acquisition procedures as supercomputer centers around the world choose HPCC for their required performance testing from bidding vendors.

June 2010 marked the release of version 1.4.1 of the benchmark code. And in 2011, the HPCC Awards competition continued with two classes of submissions; Class 1: Best Performance and Class 2: Most Productivity. While the former still invites submissions from large HPC installations around the globe and awards four winners in four categories (HPL, STREAM, FFT, RandomAccess). The latter evolved over time to invite source code submissions of tests not included in HPCC and implemented in various languages. It stressed the productivity aspect of programming languages and HEC architectures. Usually more than one winner is awarded. The competition results are customarily announced during a BOF session at SC conference series. Additional information about the awards competition can be found on the HPCC Awards website: `http://www.hpcchallenge.org/`.

Development of the HPC Challenge Benchmarks is being funded by the Defense Advanced Research Projects Agency (DARPA) High Productivity Computing Systems (HPCS) Program. Dr. Charles Holland is the current HPCS program manager. According to him: "The HPCS program is interested in both improved performance and ease of programming. Combining these two will give us the productivity that the national security community needs. For performance, the HPC Challenge benchmarks augment LINPACK with benchmarks that use more challenging memory access patterns, providing a more accurate evaluation of HPC systems."

1.4 The Benchmark Tests' Details

Extensive discussion and various implementations of the HPCC tests were given elsewhere [DL05, LD07, TK06, PBV+06, GS06]. However, for the sake of completeness, this section lists the most important facts pertaining to the HPCC tests' definitions.

All calculations use *double precision* floating-point numbers as described by the IEEE 754 standard [75485] and no mixed precision calculations [LLL+06] are allowed. All the tests are designed so that they will run on an arbitrary number of processors (usually denoted as p). Figure 1.3 shows a more detailed definition of each of the seven tests included in HPCC. In addition, it is possible to run the tests in one of three testing scenarios to stress various hardware components of the system. The scenarios are shown in Figure 1.4.

1.4.1 General Guidelines

1. The use of high level languages is encouraged.

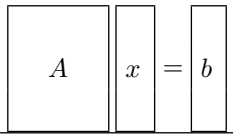
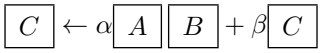
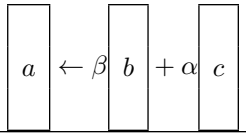
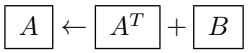
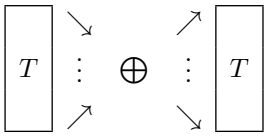
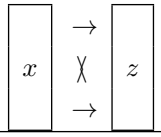
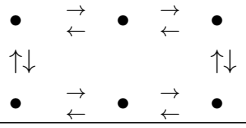
HPL	
	Compute x from the system of linear equations $Ax = b$.
DGEMM	
	Compute update to matrix C with a product of matrices A and B.
STREAM	
	Perform simple operations on vectors a, b, and c.
PTRANS	
	Compute update to matrix A with a sum of its transpose and another matrix B.
RandomAccess	
	Perform integer update of random vector T locations using pseudo-random sequence.
FFT	
	Compute vector z to be the Fast Fourier Transform (FFT) of vector x.
b_eff	
	Perform ping-pong and various communication ring exchanges.

FIGURE 1.3: Detail description of the HPCC component tests (A, B, C – matrices, a, b, c, x, z – vectors, α, β – scalars, T – array of 64-bit integers).

2. Calls to tuned library routines could be used in the submission but explicit and "elegant" coding of all aspects of the benchmark is preferred.

3. The entire benchmark could be expressed by using a few built-in operators of an hypothetical programming language. However, such submissions are strongly discouraged as they only show operator overloading and funciton call syntax and say nothing about the language. In partic-

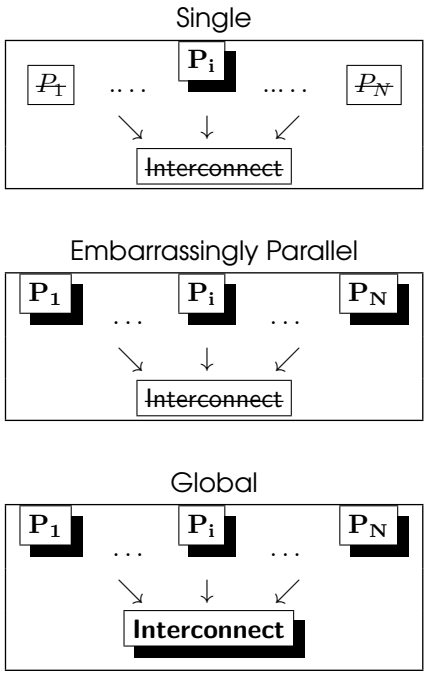


FIGURE 1.4: Testing scenarios of the HPCC components.

ular, how it deals with issues critical to HPC like expressing parallelism and hiding latency.

1.4.2 HPL

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.

1.4.2.1 Description

HPL solves a linear system of equations of order n:

$$Ax = b; \quad A \in \mathbf{R}^{n \times n}; \; x, b \in \mathbf{R}^n \tag{1.1}$$

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

$$P[A, b] = [[L, U], y]. \tag{1.2}$$

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:

$$Ux = y. \tag{1.3}$$

The lower triangular matrix L is left unpivoted and the array of pivots is not returned.

1.4.2.2 Data Size

A is n by n double precision (in IEEE 754 sense) matrix, b is n-element vector. The size of the A matrix ($8n^2$ bytes) should be at least half of the system memory.

1.4.2.3 Initialization

Both A and b should contain values produced by a reasonable pseudo-random generator with an expected mean of zero. "Reasonable" in this context means compact, fast, and producing independent and identically distributed elements.

1.4.2.4 Timed Region

The timed portion of the code performs steps given by equations (1.2) and (1.3) and does not include time to generate A and b.

1.4.2.5 Duration

Until solution to (1.1) is obtained.

1.4.2.6 Verification

Correctness of the solution is ascertained by calculating the following scaled residual:

$$r = \frac{\|Ax - b\|_\infty}{\epsilon(\|A\|_\infty \|x\|_\infty + \|b\|_\infty)n} \tag{1.4}$$

where ϵ is machine precision for 64-bit floating-point values and n is the size of the problem. The solution is valid if the following holds:

$$r < 16 \tag{1.5}$$

1.4.2.7 Performance

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 thus if the time to solution is t_S the formula for performance (in Gflop/s) is:

$$p_{\mathsf{HPL}} = \frac{\frac{2}{3}n^3 + \frac{3}{2}n^2}{t_S} 10^{-9}. \tag{1.6}$$

1.4.2.8 Alternative Implementations

If an alternative algorithm is chosen it should be able to deal with zeros on the diagonal (some sort of pivoting needs to be used) and the precision of the calculations needs to be preserved.

1.4.3 RandomAccess

1.4.3.1 Description

Let $T[\cdot]$ be a table of size 2^n.
Let $\{a_i\}$ be a stream of 64-bit integers of length $N_U = 2^{n+2}$ generated by the primitive polynomial over GF(2)[2]:
$x^2 + x + 1$.
For each a_i, set

$$T[a_i\langle 63, 64-n\rangle] \leftarrow T[a_i\langle 63, 64-n\rangle] \oplus a_i \tag{1.7}$$

where:

- $\oplus$ denotes addition in GF(2) i.e. "exclusive or" (XOR)
- $a_i\langle l, k\rangle$ denotes the sequence of bits within a_i, e.g. $\langle 63, 64-n\rangle$ are the highest n bits.

1.4.3.2 Data Size

The parameter $m(= 2^n)$ is defined such that:
m is the largest power of 2 that is less than or equal to half of the system memory. Since the elements of the main table are 64-bit quantities, the table occupies $8m$ bytes of memory.

1.4.3.3 Initialization

Table elements are set such that:

$$\forall_{0 \le i < 2^n} T[i] \equiv i \tag{1.8}$$

1.4.3.4 Timed Region

The timed region consists of computation (1.7). The initialization (1.8) is not timed.

[2]Galois Field of order 2 – The elements of GF(2) can be represented using the integers 0 and 1, i.e., binary operands.

1.4.3.5 Duration

Ideally, 2^{n+2} updates should be performed to the main table ($N_U = 2^{n+2}$). However, the computation can be prematurely stopped after 25% of the time of the HPL run (but not shorter than 1 minute). Thus:

$$N_U \leq 2^{n+2} \tag{1.9}$$

1.4.3.6 Verification

The update defined by (1.7) should be repeated by an alternative method that is safe (does not generate errors resulting from, for example, race conditions in memory updates). If the benchmarked update was correct, the table should return to its initial state defined by (1.8). However, 1% of entries may have incorrect values, i.e. given a function:

$$f(i) = \begin{cases} 0 & \text{if } T[i] = i \\ 1 & \text{otherwise} \end{cases} \tag{1.10}$$

the following should hold:

$$\sum_{i=0}^{N_U} f(i) \leq 10^{-2} N_U \tag{1.11}$$

1.4.3.7 Performance

Let $t_{\mathsf{RandomAcccess}}$ be the time it took to finish the timed portion of the test (including N_U updates) then peroformance (in GUPS: Giga Updates Per Second) is defined as:

$$p_{\mathsf{RandomAcccess}} = \frac{N_U}{t_{\mathsf{RandomAcccess}}} 10^{-9}. \tag{1.12}$$

1.4.3.8 Alternative Implementations

Constraints on the look-ahead and storage before processing on distributed memory multi-processor systems is limited to 1024 per process (or processing element). The pseudo-random number generator that generates sequence $\{a_i\}$ has to be used.

1.4.4 Global EP-STREAM-Triad

1.4.4.1 Description

EP-STREAM-Triad is a simple benchmark program that measures sustainable memory bandwidth (in Gbyte/s) and the corresponding computation rate for a simple vector kernel operation that scales and adds two vectors:

$$a \leftarrow b + \alpha\, c \tag{1.13}$$

where:

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

The computation is peformed simultaneously on each computing element on its local data set.

1.4.4.2 Data Size

a, b, and c are m-element double precision vectors. The combined size of the vectors ($24m$ bytes) should be at least quarter of the system memory.

1.4.4.3 Initialization

Vectors b and c should contain values produced by a reasonable pseudo-random number generator.

1.4.4.4 Timed Region

The timed portion of the code should perform operation given by (1.13) at least 10 times.

1.4.4.5 Duration

The kernel operation should be repeated at least 10 times.

1.4.4.6 Verification

The norm of the difference between reference and computed vectors is used to verify the result: $\|a - \hat{a}\|$. The reference vector $\hat{a}$ is obtained by an alternative implementation.

1.4.4.7 Performance

The benchmark measures Gbyte/s and the number of items transferred is $3m$. The minimum time $t_{\min}$ is taken of all the repetitions of the kernel operation. Performance is thus defined as:

$$p_{\text{EP-STREAM-Triad}} = 24\frac{m}{t_{\min}}10^{-9} \tag{1.14}$$

1.4.4.8 Alternative Implementations

1.4.5 Global FFT

1.4.5.1 Description

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

$$Z_k \leftarrow \sum_j^m z_j e^{-2\pi i \frac{jk}{m}}; \quad 1 \le k \le m \tag{1.15}$$

where:

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

1.4.5.2 Data Size

Z and z are m-element double precision complex vectors. The combined size of the vectors ($32m$ bytes) should be at least quarter of the system memory. The size m of the vectors can be implementation-specific, e.g. be an integral power of 2.

1.4.5.3 Initialization

Vector z should contain values produced by a reasonable pseudo-random number generator. The real and imaginary parts of z should be generated independently. The layout of vectors z and Z should not be scrambled either before or after the computation.

1.4.5.4 Timed Region

The computation implied by (1.15) is timed together with the portion of code that unscrambles (if necessary) the resulting vector data. Timing for computation and unscrambling can be given separately for informational purposes but the combined time is used for calculating performance.

1.4.5.5 Duration

Until the transform defined by (1.15) is obtained.

1.4.5.6 Verification

Verification is done by acertainig the following bound on the residual:

$$\frac{\|z - \hat{z}\|_\infty}{\epsilon \ln m} < 16 \tag{1.16}$$

where $\hat{z}$ is the result of applying a reference implementation of the inverse transform to the outcome of the benchmarked code (in infinite-precision arithmetic the residual should be zero):

$$\hat{z}_k \leftarrow \sum_j^m Z_j e^{2\pi i \frac{jk}{m}}; \quad 1 \le k \le m \tag{1.17}$$

1.4.5.7 Performance

The operation count is taken to be $5m \log_2 m$ for the calculation of the computational rate (in Gflop/s) in time t:

$$p_{\mathsf{FFT}} = 5\frac{m \log_2 m}{t} 10^{-9} \tag{1.18}$$

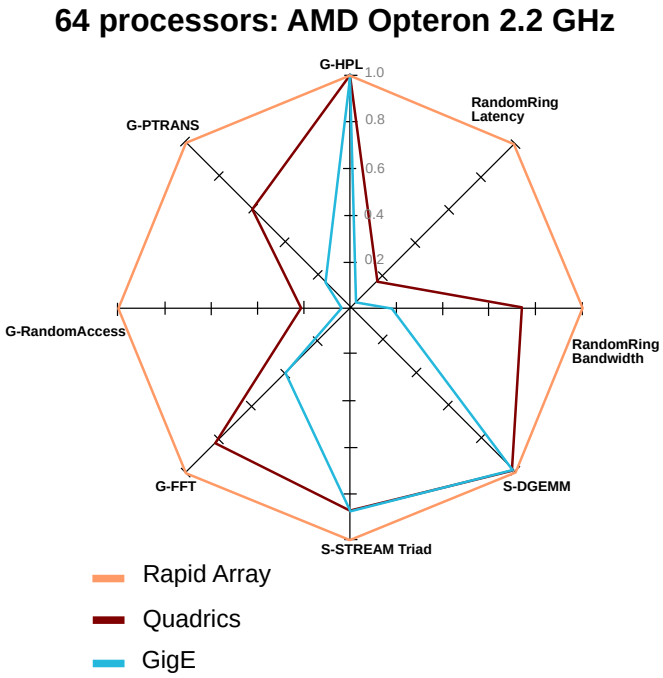


FIGURE 1.5: Sample kiviat diagram of results for three different interconnects that connect the same processors.

1.4.5.8 Alternative Implementations

The reference implementation splits the algorithm into computational and communication portions which do not overlap. Valid submsissions may choose other methods that take advantage of language and architectural features.

The number of processors may be implementation-specific, e.g. be an integral power of 2.

1.5 Benchmark Submission Procedures and Results

The reference implementation of the benchmark may be obtained free of charge at the benchmark's web site[3]. The reference implementation should be used for the base run: it is written in portable subset of ANSI C [KR78] using hybrid programming model that mixes OpenMP [Ope, CDK+01] threading

[3] `http://icl.cs.utk.edu/hpcc/`

with MPI [For94, For95, For97] messaging. 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. The HPCC rules allow only standard system compilers and libraries to be used through their supported and documented interface and the build procedure should be described at submission time. This ensures repeatability of the results and serves as educational tool for end users that wish to use the similar build process for their applications.

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

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. The baseline run should use the reference implementation of HPCC and in a sense it represents the scenario when an application requires use of legacy code – a code that can not be changed. The optimized run allows to perform more aggressive optimizations and use system-specific programming techniques (languages, messaging libraries, etc.) but at the same time still gives the verification process enjoyed by the base run.

All of the submitted results are publicly available after they have been confirmed by email. In addition to the various displays of results and raw data export the HPCC website also offers a kiviat chart display to visually compare systems using multiple performance numbers at once. A sample chart that uses actual HPCC results' data is shown in Figure 1.5.

Figure 1.6 show performance results of currently operating clusters and supercomputer installations. Most of the results come from the HPCC public database.

1.6 Performance Trends

HPCC Awards ever since their introduction in 2005 sparked interest in the HPCC Suite and contributed many new submissions, most importantly, from the largest supercomputer installations in the world. Figures 1.7, 1.8, 1.9, and 1.10 show performance trends for the first, second, and third places in the competition over the 7-year period. In 2011, the K computer from Japan has become an undisputed winner for all four tests tracked by the Performance Category of HPCC Awards. This had not been the case in the prior years. In 2009 and 2010, Jaguar supercomputer from Oak Ridge National Laboratory (ORNL) did not dominate all four tests.

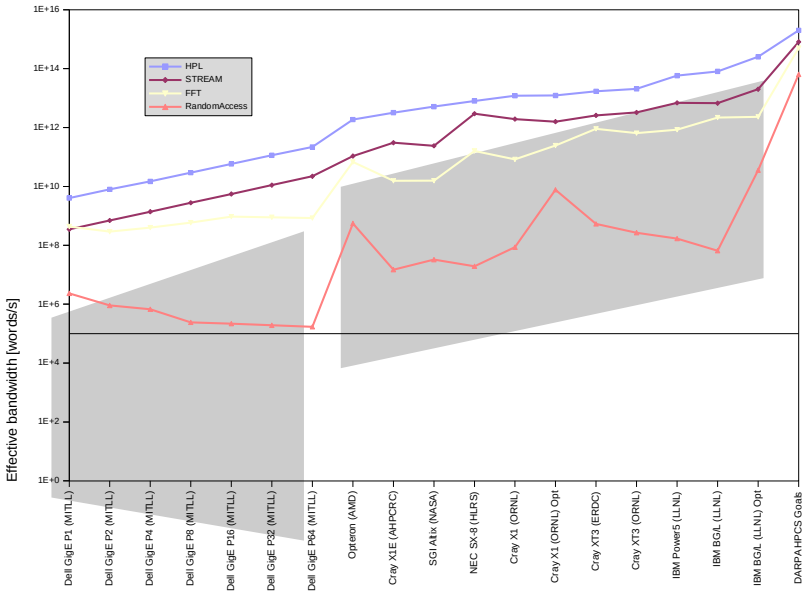


FIGURE 1.6: Sample interpretation of the HPCC results.

One of the important observations in Figure 1.7 is the fact that the HPL performance from the HPCC Suite is usually lower than the number recorded by TOP500. This is no accident as the code used for HPCC submission usually does not match exactly what was used for TOP500. The former has to satisfy the HPCC submission rules including data layout software interfaces. On the contrary, the TOP500 code is only to fulfill "paper-and-pencil" description of the High Performance LINPACK. But importantly, optimization of HPCC has to focus on all components of the suite which might not necessarily benefit the HPL component. Finally, the competive pressure from all the TOP500 submissions is much greater incentive to maximize the HPL score. This may be most visible for the K Computer and its 2011 HPCC result of 2118 Tflop/s. This is nearly 80% lower result than the 10510 Tflop/s reported on the TOP500 list. The easiest explanation is the lack of competition – the second fastest system is ORNL's Jaguar at 1534 Tflop/s and so the result reported by the K Computer is sufficent to put it as the winner for the HPL test. Clearly, if K Computer reported only 1600 Tflop/s it still would have been a winner. The reason for going all the way to over 2000 Tflop/s was the RandomAccess test. The second place contender for RandomAccess is a Blue Gene/P system at Lawrence Livermore National Laboratory at 117 GUPS. For the K Computer the reported value was 121 GUPS, only about 3% better result but a sufficient one to win. From our analysis we conclude that large enough partition of the K Computer was used to give it a winning score in all four tests tracked by

Name	Generation	Computation	Communication	Verification	Per-processor data
HPL	n^2	n^3	n^2	n^2	p^{-1}
DGEMM	n^2	n^3	n^2	1	p^{-1}
STREAM	m	m	1	m	p^{-1}
PTRANS	n^2	n^2	n^2	n^2	p^{-1}
RandomAccess	m	m	m	m	p^{-1}
FFT	m	$m \log_2 m$	m	$m \log_2 m$	p^{-1}
b_eff	1	1	p^2	1	1

TABLE 1.2: Time complexity formulas for various phases of the HPCC tests (m and n correspond to the appropriate vector and matrix sizes, p is the number of processors.)

the HPCC Awards in the Performance Category. The partition used for the winning submission consisted of 18432 nodes with 8 cores per node and 147456 cores total. This is far less (almost 80% less) than the 705024 cores used for the TOP500 submission.

1.7 Scalability Considerations

There are a number of issues to be considered for benchmarks such as HPCC that have scalable input data to allow for arbitrary sized system to be properly stressed by the benchmark run. Time to run the entire suite is a major concern for institutions with limited resource allocation budgets. Each component of HPCC has been analyzed from the scalability standpoint and Table 1.2 shows the major time complexity results. In the following, it is assumed that:

- M is the total size of memory,
- m is the size of the test vector,
- n is the size of the test matrix,
- p is the number of processors,

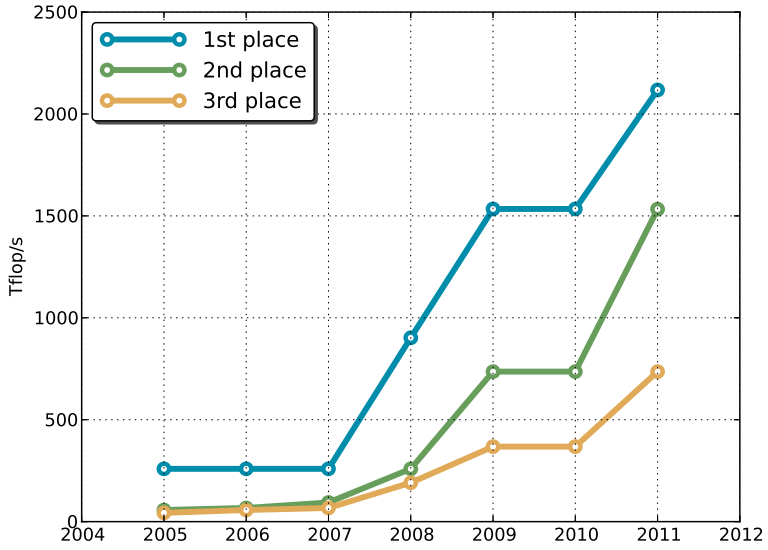


FIGURE 1.7: Historical trends for winners of the performance category of the HPCC Awards for Global HPL.

- t is the time to run the test.

Clearly any complexity formula that grows faster than linearly with respect to any of the system sizes is a cause of potential problem time scalability issue. Consequently, the following tests have to be addressed:

- HPL because it has computational complexity $\mathcal{O}(n^3)$.
- DGEMM because it has computational complexity $\mathcal{O}(n^3)$.
- b_eff because it has communication complexity $\mathcal{O}(p^2)$.

The computational complexity of HPL of order $\mathcal{O}(n^3)$ may cause excessive running time because the time will grow proportionately to a high power of total memory size:

$$t_{\text{HPL}} \sim n^3 = \left(n^2\right)^{3/2} \sim M^{3/2} = \sqrt{M^3} \tag{1.19}$$

To resolve this problem we have turned to the past TOP500 data and analyzed the ratio of *Rpeak* to the number of bytes for the factorized matrix for the first entry on all the lists. It turns out that there are on average 6 ± 3 Gflop/s for each matrix byte. We can thus conclude that performance rate of HPL

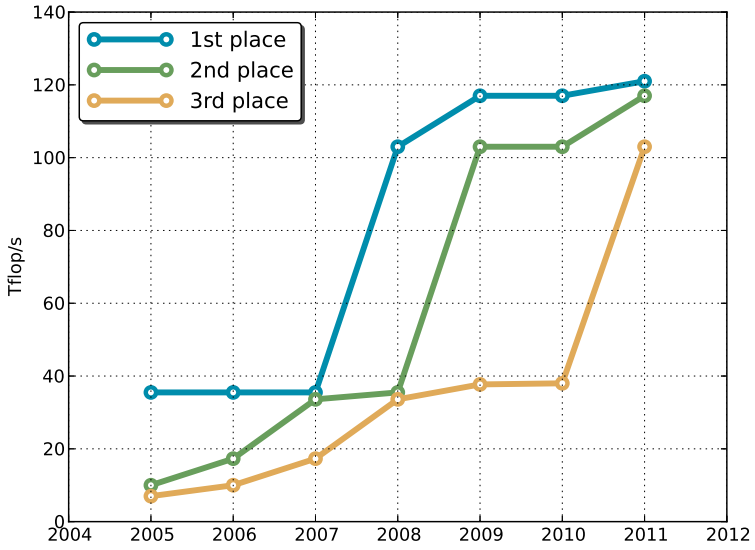


FIGURE 1.8: Historical trends for winners of the performance category of the HPCC Awards for Global RandomAccess.

remains constant over time ($r_{\text{HPL}} \sim M$) which leads to:

$$t_{\text{HPL}} \sim \frac{n^3}{r_{\text{HPL}}} \sim \frac{\sqrt{M^3}}{M} = \sqrt{M} \tag{1.20}$$

which is much better than (1.19).

There seems to be a similar problem with the DGEMM as it has the same computational complexity as HPL but fortunately, the n in the formula related to a single process memory size rather than the global one and thus there is no scaling problem.

Lastly, the b_eff test has a different type of problem: its communication complexity is $\mathcal{O}(p^2)$ which is already prohibitive today as the number of processes of the largest system in the HPCC database is almost 150 thousand. This complexity comes from the ping-pong component of b_eff that attempts to find the weakest link between all nodes and thus, theoretically, needs to look at the possible process pairs. The problem was remedied in the reference implementation by adapting the runtime of the test to the size of the system tested.

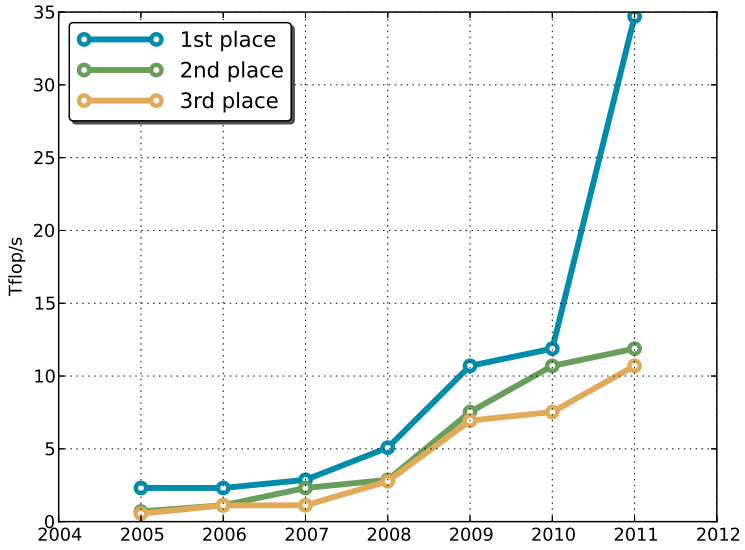


FIGURE 1.9: Historical trends for winners of the performance category of the HPCC Awards for Global FFT.

1.8 Conclusions and Future Directions

No single test can accurately compare the performance of any of today's high-end system let alone any of those envisioned by the HPCS program in the future. Thusly, the HPCC suite stresses not only the processors, but the memory system and the interconnect. It is a better indicator of how a supercomputing system will perform across a spectrum of real-world applications. Now that the more comprehensive, HPCC suite is available, it could be used in preference to comparisons and rankings based on single tests. The real utility of the HPCC 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 parallel architectures. But the tests indicate that 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. The HPCC tests provide users with additional information to justify policy and purchasing decisions. We expect to expand and perhaps remove some existing benchmark components as we continue learning about the collection.

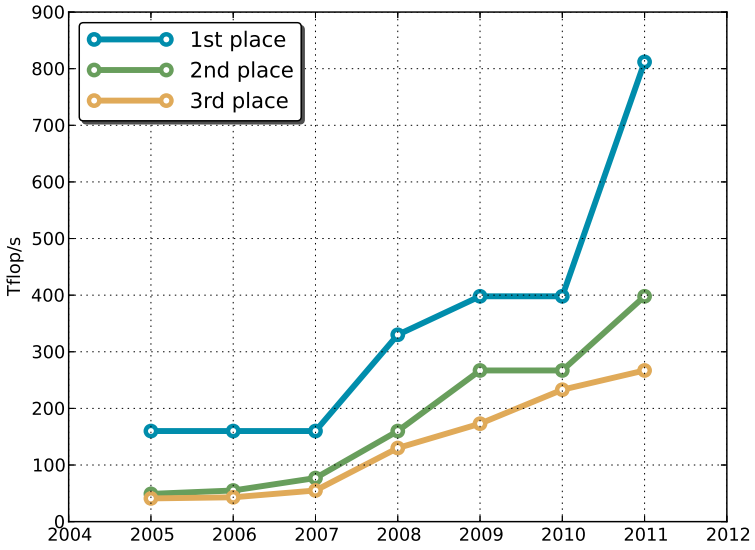


FIGURE 1.10: Historical trends for winners of the performance category of the HPCC Awards for Global EP-STREAM-Triad.

Looking forward into the High End Computing hardware trends, HPCC has a role to play in testing supercomputer installations that draw majority of their performance from hardware accelerators. The trend started with TOP500's first Peta-FLOP computer: Roadrunner based on IBM Cell processors. Currently, GPU-based computers have noticable presence at the presitiguous spots of TOP500. HPCC is well positioned to offer a rich view of such systems and their increased complexity. Hence, we are actively looking into extending HPCC availability for hardware-accelerated machines.

Bibliography

[75485] ANSI/IEEE Standard 754-1985. Standard for binary floating point arithmetic. Technical report, Institute of Electrical and Electronics Engineers, 1985.

[CDK+01] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

[DL05] Jack Dongarra and Piotr Luszczek. Introduction to the HPC Challenge benchmark suite. Technical Report UT-CS-05-544, University of Tennessee, 2005.

[DLP03] 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.

[For94] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8, 1994.

[For95] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard (version 1.1), 1995. Available at: `http://www.mpi-forum.org/`.

[For97] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface, 18 July 1997. Available at `http://www.mpi-forum.org/docs/mpi-20.ps`.

[GS06] Rahul Garg and Yogish Sabharwal. Software routing and aggregation of messages to optimize the performance of the HPCC Randomaccess benchmark. In *Proceedings of SC06*, Tampla, FL, November 11-17 2006.

[Kah97] 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.

[Kep04] Jeremy Kepner. HPC productivity: An overarching view. *International Journal of High Performance Computing Applications*, 18(4), November 2004.

[KR78] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, Upper Saddle River, New Jersey, 1978.

[LD07] Piotr Luszczek and Jack Dongarra. High performance development for high end computing with Python Language Wrapper (PLW). *International Journal of High Perfomance Computing Applications*, 21(2), Summer 2007.

[LLL+06] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy. In *Proceedings of SC06*, Tampa, Florida, Nomveber 11-17 2006. See `http://icl.cs.utk.edu/iter-ref`.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 19 1965.

[MSDS06] Hans W. Meuer, Erich Strohmaier, Jack J. Dongarra, and Horst D. Simon. *TOP500 Supercomputer Sites*, 28th edition, November 2006. (The report can be downloaded from `http://www.netlib.org/benchmark/top500.html`).

[Ope] OpenMP: Simple, portable, scalable SMP programming. `http://www.openmp.org/`.

[PBV+06] Steve J. Plimpton, R. Brightwell, Courtney Vaughan, K. Underwood, and M. Davis. A simple synchronous distributed-memory algorithm for the HPCC RandomAccess benchmark. In *Proceedings of Cluster 2006 – IEEE International Conference on Cluster Computing*, September 2006.

[TK06] Nadya Travinin and Jeremy Kepner. pMatlab parallel Matlab library. *International Journal of High Perfomance Computing Applications*, 2006. Submitted to Special Issue on High Productivity Languages and Models.