HPC Benchmark Assessment with Statistical Analysis

Fei Xing¹, Haihang You², and Charngda Lu³

¹ Mathematica Policy Research Inc., USA
fxing@mathematica-mpr.com

² Institute of Computing Technology, Chinese Academy of Sciences, China
northshore.you@gmail.com

³ wayStorm, Inc, Taiwan
charngdalu@yahoo.com

Abstract
High-performance computing (HPC) benchmarks are widely used to evaluate and rank system performance. This paper introduces a benchmark assessment tool equipped with a rigorous statistical method to evaluate HPC benchmarks against a set of scientific applications. The method is based on the combination of Variable Clustering (VarCluster) and Principal Component Analysis (PCA). We built the tool upon HPC Challenge (HPCC) benchmark suite and six popular scientific applications of Kraken, a petaflop supercomputer. Experimental results show that HPCC’s Fast Fourier Transform (FFT) kernel, rather than the High-Performance Linpack (HPL) on which Top500 is based, is more representative of the HPC workloads on Kraken.

Keywords: statistical method; performance analysis; principal component analysis; variable clustering

1 Introduction

Seeking widely accepted HPC benchmarks and metrics which are representative of real-world workloads has been a popular research topic [9]. The most cited supercomputer ranking is the Top500 [15]. Starting from 1993, every six months the Top500 website releases a ranking of the most powerful computer systems based on the High-Performance Linpack (HPL) benchmark result. Another ranking, Graph 500 [16], evaluates computers by a benchmark tailored to mimic data-intensive HPC workloads using graph traversal operations. Besides, Green 500 list [10] judges computers by their HPL performance-energy efficiency. Conceptually these rankings attempt to condense the complexity of any machine from a smartphone to a leadership-class supercomputer into a single number that people can easily grasp and categorize. The downside of this methodology is also obvious: it usually only covers one dimension, i.e., the outcome of a particular benchmark, of the overall performance.

In light of the benchmark debates in the HPC community, in this paper we propose a methodology to evaluate a benchmark’s adequacy and representativeness of real-world HPC workloads. First, both the
benchmark and scientific applications of choice are profiled and measured. Then, a statistical algorithm is applied to reduce the high-dimensional performance data into a few numbers. Lastly, a performance scoring system is built in which a few (tunable, usually 3) easily interpretable Performance Indices (PIs) are enough to characterize the original high-dimensional performance metrics. Intuitively, the PIs measure the consumption of hardware resources, be it CPU (computation), network (communication), I/O, or memory, and generally, the higher the PIs, the more HPC resources are utilized by the application.

In practice, our method starts by fixing the choices of performance tools and baseline kernels (the benchmarks). For the latter, we chose the HPC Challenge (HPCC) [14]. Performance of the baseline kernels are then measured by the chosen performance tools and the resulting multi-dimensional performance datasets are fed into our algorithm. A few (usually 3) linear formulae to calculate the PIs will be generated by the algorithm. Then for any application instrumented by the same performance tools, we apply the linear formulae to condense numerous performance metrics into a few easy-to-understand PIs. The affinity between baseline kernels and applications is the distance of their PIs. Our experiments show that FFT, rather than HPL, is a better representative of the HPC workloads we tested.

The remainder of the paper is organized as follows. Section 2 explains how PCA and VarCluster are used in our statistical performance analysis strategy and the evaluation of benchmark affinity with real-world applications. In Section 3 we describe the HPCC benchmark suite and profiling tools used, and in the subsequent Section 5 we discuss the experiments conducted on the Kraken supercomputer. The paper continues with related work in Section 6 and concludes with future directions in Section 7.

2 Methodology

The evaluation process consists three components: Performance Data Preprocessing, Performance Space Construction, and Evaluation. The procedures are illustrated in Fig. 1. First, with a set of performance tools, we obtain performance measurements of HPC benchmarks (HPCC in this paper) per MPI rank. Values of different performance metrics are normalized within a comparable scale; Second, statistical methods are applied to creates a set of linear formulae which can be used to compute Performance Indices (PI). The performance values of benchmarks are mapped into the performance space established by PIs; Lastly, the first step is repeated for real scientific applications. performance data are acquired, normalized and fed into the linear formulae to generate their PIs; Then, within the performance space, the proximity of each benchmark is computed by comparing with real applications. Below we describe procedures used in our methodology. We assume there are \( n \) HPC benchmarks and the performance tools yield \( m \) metrics, so the performance measurement of baseline kernels generates an \( n \times m \) data matrix \( X \).

![Figure 1: Flow chart of the statistical performance analysis and benchmark evaluation](image-url)
2.1 Performance Data Preprocessing

The raw performance data are preprocessed. For event-type metrics, e.g. floating-point operations and cache misses, convert them to “rates” by dividing by the wall times of the respective application runs. For sample-type or unit-less metrics, e.g. memory usage or load imbalance level, do nothing.

In order to ensure the values of different performance metrics are within a comparable scale, we further normalize the data by dividing each value by the maximal value, i.e. for each column $x_i$ of $X$ ($1 \leq i \leq m$), obtain the normalized performance matrix $Z = (z_1, \ldots, z_m)$, where

$$z_i = \frac{x_{ij}}{\max_{1 \leq j \leq n} x_{ij}}$$

2.2 Performance Space Construction

This procedure reduces the original performance metric space into simple performance space, where each new PI is a hybrid combination of the original performance metrics. More importantly, the original performance metrics that contribute to one PI are not overlap with the others, which guarantees the independency of each new PI.

2.2.1 Variable Clustering

Variable clustering [7] partitions variables into a few homogeneous clusters in which the variables are strongly related to each other, and therein groups variables sharing similar information together. The homogeneity of a cluster $C$, which measures the adequacy between the variables in cluster $C$ and its central synthetic variable $y$, is essentially quantified by the sum of squared correlation between each variable and the centroid $y$. With the defined homogeneities, various cluster analysis algorithms are available to partition the variables into several sub-groups. In practice, the hierarchical clustering algorithm is applied to the preprocessed data matrix $Z$ because it provides a robust result in the sense that the algorithm starts with single-metric clusters, hence the result is not affected by the initialization. The entire hierarchical clustering process could be illustrated by a plot, called cluster dendrogram. From the cluster dendrogram, we choose a suitable cut with a small number $k$ of clusters $C_1, \cdots, C_k$ such that each cluster’s meaning can be easily interpreted.

2.2.2 PCA

Each of the clusters produced from the last step still have a few metrics, so in this step we apply PCA to each cluster and get a set of linear formulae. For each cluster, its PI calculation formula is the linear equation of the first principal component. So in total, $k$ linear formulae, each represents a cluster and its PI, are obtained.

After applying the above procedure to the preprocessed performance data of HPC benchmarks, we end up with $k$ linear formulae to compute PIs in later steps and a new $k$-dimensional performance space with each original performance data get a position in it.

2.3 Benchmark Evaluation

Given an application we can gather performance data measured by the same performance tools. After preprocessing procedure mentioned in Section 2.1, we obtain PIs for the application by plugging processed performance data into the $k$ formulae. Once select a list of widely used real-world applications, these applications can be viewed as representative cluster $A$ of scientific applications. We claim that a
benchmark $b^*$ among all the baseline kernels $B$ can best characterize real-world workload by following standard:

$$b^* = \arg \max_{b \in B} \text{Prox}(b, A),$$

where Prox is a well-defined criterion to measure the proximity.

### 3 HPC Benchmarks and Performance Tools

In this paper, we select HPC Challenge (HPCC) [14] benchmark suite, FPMPI and PAPI as the performance tools of choice.

#### 3.1 HPC Challenge Benchmarks

HPCC is a benchmark suite developed specifically to assess supercomputers in various categories such as floating point rate of execution, memory bandwidth, and communication bandwidth and latency. It is made up of seven common kernels, and one of which is the HPL used in the Top 500 ranking. See Table 1 for details.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Mode(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGEMM</td>
<td>1, *</td>
<td>Real-valued dense matrix multiplication in double-precision.</td>
</tr>
<tr>
<td>FFT</td>
<td>1, *, M</td>
<td>Discrete Fourier Transform of a complex-valued vector.</td>
</tr>
<tr>
<td>HPL</td>
<td>M</td>
<td>High-Performance Linpack, which solves a linear system of equations using Gaussian elimination with partial pivoting. It makes extensive use of matrix multiplications.</td>
</tr>
<tr>
<td>LatencyBdh</td>
<td>M</td>
<td>MPI microbenchmark that measures the latency and bandwidth of various communication patterns.</td>
</tr>
<tr>
<td>PTRANS</td>
<td>M</td>
<td>Parallel matrix transpose, which exercises a large volume of communication owing to pairs of processes transferring data with each other simultaneously.</td>
</tr>
<tr>
<td>RA</td>
<td>1, *, M, LCG, M_LCG</td>
<td>Random Access: Updates to random locations in a large global memory array. The (pseudo) random sequence is generated either by finite field arithmetic (the original algorithm) or by Linear Congruential Generator (LCG).</td>
</tr>
<tr>
<td>Stream</td>
<td>1, *</td>
<td>Measuring sustainable memory bandwidth and the concomitant computation rates for four vectorized operations.</td>
</tr>
</tbody>
</table>

**Table 1:** HPC Challenge (HPCC) Benchmark Suite. Mode 1 means Serial/Single Processor, * means Embarrassingly Parallel, and M means MPI Parallel (Global).

HPCC covers three parallelism regimes: Serial/Single Processor, Embarrassingly Parallel (all processors perform the same task as in the Serial case without any inter-processor communication and the slowest performance among all processors is reported), and MPI Parallel/Global (all processors compute and exchange data with each other). Collectively the kernels stress-test multiple facets of a computer system: Processor floating-point computation capability (DGEMM, FFT, HPL), memory bandwidth (PTRANS, Stream) and latency (FFT, RA), and interconnect bandwidth (FFT, HPL, LatencyBdh, PTRANS) and latency (FFT, LatencyBdh, RA).
3.2 Performance Tools

We use two popular tools to gather the performance attributes:

- **FPMPI**: Fast Profiling MPI [11] is a light-weight profiling library intended as a first step towards understanding the nature of the communication patterns and potential bottlenecks in a parallel application. For each metric, FPMPI reports the minimum/maximum/average values over all MPI processes. FPMPI also derives a value between 0 and 1 called “load imbalance factor,” defined as (max-min)/max, so 0 indicates perfect load balance while 1 suggests a huge imbalance.

- **PAPI**: Performance Application Programming Interface [6] is the de facto standard instrumentation library to access a processor’s hardware performance counters. It features a cross-platform API which allows application programmers and tool developers to easily and efficiently capture precise processor performance in a portable way. On supported processor architectures, PAPI can measure all levels of cache misses, translation lookaside buffer (TLB) misses, branch prediction failures, CPU cycles, counts of a variety of instructions such as memory accesses, floating-point arithmetic, integer operations.

4 Scientific Applications

Six real-world scientific applications and their corresponding input data, covering a total of four different scientific fields (molecular biosciences, atmospheric sciences, physics as well as astronomical sciences), are used in our experiments. According to a recent workload study of Kraken supercomputer [12], those above applications could serve as good representatives of real scientific applications running on Kraken. Meanwhile, selecting comparable number of applications within each scientific field grants the equal importance of each scientific field in our study.

- **ExaML** [19] (Exascale Maximum Likelihood) is a molecular sequencing tool for large-scale phylogenetic inference on supercomputers. It implements the RAxML (randomized accelarated maximum likelihood) algorithm. We used ExaML version 2.2 and “49 DNA sequences” test data found in its source code distribution.

- **GADGET** [1] is a cosmological $n$-body/SPH (smoothed particle hydrodynamics) simulation code which uses a hierarchical tree algorithm to compute gravitational forces. We used GADGET version 2.0.7 and the input simulation run is the collision of two disk galaxies (20,000 disks and 40,000 halo particles in total.)

- **HOMME** [17] (High-Order Method Modeling Environment) is a petascale, high-fidelity, global hydrostatic atmospheric modeling framework based on the spectral element and discontinuous Galerkin methods. We used the one-day simulation benchmark input problem.

- **MILC** [2] (MIMD Lattice Computation) package implements numerical simulations of lattice quantum chromodynamics (QCD), the theory of the strong interactions of subatomic physics. We used MILC version 7 with the benchmark input (problem size 256) from the NERSC website.

- **NAMD** [3] is a petascale MD code written using the object-oriented Charm++ parallel programming model. We used NAMD version 2.9. The input dataset is a one-million-atom large molecular system STMV (Satellite Tobacco Mosaic Virus.)

- **WRF** [4] (Weather Research and Forecasting) is a mesoscale numerical weather prediction system designed to serve both meteorological research and operational forecasting needs. We used WRF version 3.4.1 and “em_quarter_ss” benchmark input.
5 Results and Discussion

Kraken is a 1.17 peak petaFLOPS supercomputer managed by the National Institute for Computational Sciences at the Oak Ridge National Laboratory. It is a Cray XT5 system consisting of 9,408 compute nodes with 112,896 processor cores, 147 TB of memory, and 3.3 PB storage. Each compute node has twelve 2.6 GHz AMD “Istanbul” processor cores and 16 GB memory. The system sports a high-speed, low-latency network composed of Cray’s proprietary SeaStar2+ chips on board and a scalable 3D torus interconnect fabric. We ran HPCC benchmarks over a wide range of MPI ranks (12 to 12,288) to test the robustness of the algorithm. We try as many different modes (see Table 1) as possible. For clarity, we describe a kernel as Mode_KernelName (for example, M_RA), unless the kernel can only run in one mode, e.g. PTRANS. The kernels and the performance metrics are as follows:

- The 16 kernels \((n = 16)\) are: 1_DGEMM, *DGEMM, 1_FFT, *FFT, M_FFT, HPL, LatencyBdh, PTRANS, 1_RA, *_RA, M_RA, 1_LCG_RA, *_LCG_RA, M_LCG_RA, 1_Stream, *_Stream.

- The 11 metrics \((m = 11)\) are: MPI communication time (MPI_CT), MPI synchronization time (MPI_ST), MPI calls (MPI_Call), total MPI bytes (MPI_bytes), memory usage (Memory), floating point operations (Flops), total instructions (TOT_INS), level 2 data cache accesses (L2_DCA), level 1 data cache accesses (L1_DCA), communication imbalance (Cimbl), and synchronization imbalance (Simbl). The MPI related metrics are reported by FPMPI and the rest are PAPI hardware counter events.

5.1 Performance Data and Performance Space

The performance dataset contains 3,530 observations of the 11 performance metrics. The matrix sizes range evenly from 1,000 by 1,000 to 320,000 by 320,000 and the MPI ranks vary from 12 to 12,288. We apply ClustOfVar [7] to the dataset under R programming environment. The 11 metrics are grouped into 3 clusters, see Figure 2 and Table 2.

![Figure 2: Dendrogram of hierarchical clustering used by VarCluster](image)

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Performance Metrics</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1</td>
<td>MPI bytes, MPI calls, MPI synchronization time, MPI communication time, communication imbalance, communication imbalance</td>
<td>Communication</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>Memory</td>
<td>Memory</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>Total instructions, Flops, L1 &amp; L2 cache accesses</td>
<td>Computation</td>
</tr>
</tbody>
</table>

The first cluster captures the communication performance of the applications at the inter-node level. Cluster two contains only one metric – the memory usage, which indicates the total memory usage per process. As to cluster three, it includes hardware counter events, which reflects the CPU activity of the applications at the on-chip level. Next, we apply PCA to each cluster to get the PC1 in each cluster. After calculating the PC1 and calibrating the direction in each cluster as in §2.2.2, we obtain the the linear formulae for Performance Indices as follows:
\[
\text{PI 1} = -0.32 \times \text{MPI}\_CT + 0.82 \times \text{MPI}\_ST - 0.12 \times \text{MPI}\_Call \\
- 0.07 \times \text{MPI}\_bytes + 0.44 \times \text{Cimbl} + 0.14 \times \text{Simbl},
\]
\[
\text{PI 2} = \text{Memory},
\]
\[
\text{PI 3} = 0.46 \times \text{Flops} + 0.53 \times \text{TOT}\_INS + 0.48 \times \text{L2}\_DCA \\
+ 0.53 \times \text{L1}\_DCA.
\]

To interpret the PI 1 and PI 3, we scrutinize their formulae. The weights (i.e. PCA loadings) in the linear formulae for PI 3 are all positive and have comparable magnitudes, suggesting that PI 3 assesses the computational intensity and cache reuse impartially. The formula for PI 1, however, overweights the synchronization time and communication imbalance, and downplays the communication time and other MPI metrics. Therefore, a high PI 1 value signifies excessive synchronization and an imbalanced communication pattern.

Figure 3 and Figure 4 are 3d and 2d scatterplots of all benchmark observations projected within the performance space, where Figure 4 exhibits the combination of PIs of communication and computation. In the plot, we group all the Serial/Single Processor results together, M\_RA and M\_LCG\_RA together, *\_RA and *\_LCG\_RA together respectively because of their performance similarity. After this procedure, there are 10 types of benchmark kernels remaining: HPL, M\_RAs, *\_RAs, Serial regime kernels, PTRANS, *\_DGEMM, *\_Stream, M\_FFT, *\_FFT and LatencyBdh. In Figure 3, there are two distinct strata on the Memory dimension (PI2). We find that observations with relatively small PI 2 values are single-node runs while the rest are multi-node runs. This can be explained by a common practice of MPI libraries, which implement intra-node communication (single-node case) via shared memory, and inter-node communication via extra memory buffers. From the scatterplots Figure 3 and 4 we also notice that *\_DGEMM and HPL have similar patterns in PI 3 values (computation). It is not surprising that PI values of certain kernels are also dependent on problem sizes and MPI ranks, and usually the relationship is linear. *\_DGEMM and HPL are the two kernels having a wide spread along the PI 3 axis, implying a significant correlation between computational intensity and problem size. On the other hand, LatencyBdh, PTRANS and M\_RA exhibit large scattering along the PI 1 axis, because they are network-intensive microbenchmarks.
5.2 Performance of Scientific Applications

We collect performance data and calculate Performance Indices of selected real-world scientific applications: ExaML, GADGET, HOMME, MILC, NAMD, and WRF. The PI values are listed in Table 3.

![Figure 5: Scatterplot of PI 1 vs PI 3 for multi-node HPCC benchmark workload and real-world workload.](image)

**Table 3:** Performance Indices of test applications using multiple nodes.

<table>
<thead>
<tr>
<th>Application</th>
<th>PI 1 (Comm)</th>
<th>PI 2 (Mem)</th>
<th>PI 3 (Comp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExaML</td>
<td>0.51</td>
<td>0.40</td>
<td>0.64</td>
</tr>
<tr>
<td>GADGET</td>
<td>0.46</td>
<td>0.51</td>
<td>0.52</td>
</tr>
<tr>
<td>HOMME</td>
<td>0.58</td>
<td>1.29</td>
<td>0.48</td>
</tr>
<tr>
<td>MILC</td>
<td>0.43</td>
<td>0.64</td>
<td>0.50</td>
</tr>
<tr>
<td>NAMD</td>
<td>0.45</td>
<td>2.56</td>
<td>0.67</td>
</tr>
<tr>
<td>WRF</td>
<td>1.04</td>
<td>0.53</td>
<td>0.56</td>
</tr>
</tbody>
</table>

As Figure 5 illustrated, HPCC benchmarks and applications are positioned in the Performance Space of communication and computation. A prominent feature in Figure 5 is that the HPL and *DGEMM kernels distance themselves from others, which corroborates the fact they are computationally intensive. By comparison, none of the applications can attain even half of the PI values of computation of HPL and *DGEMM. Another notable peculiarity is Serial, which has a very large PI 1 (communication) value. The “Serial” in the scatterplot denotes the entirety of Serial regime kernels. Because the HPCC kernels, including all parallelism regimes, are packaged as one MPI program, when a serial kernel runs, all other MPI tasks are idle, waiting on a MPI_Bcast call to receive results from the sole running task. Therefore, over 90% of the CPU-hour is wasted on synchronization, and the more the MPI processes, the worse the situation. Since PI 1 is a measure of excessive synchronization and imbalanced communication, as discussed at the end of §5.1, this results in very high PI 1 values for Serial regime kernels. By contrast, the *(Embarrassingly Parallel)* kernels spend much less time in either synchronization or communication, so their PI 1 values position more closely to the test applications. The diagram gives us a better perspective of evaluating benchmark performance in terms of the application performance with PIs space.

5.3 Benchmark Evaluation

The three Performance Indices (PIs) generated earlier from the statistical approach equip us an evaluation system that can capture the proximities of a HPC benchmark with real-world HPC workloads, hence in return could be applied as a guideline of HPC benchmark assessment. According to a recent workload study on Kraken [21], the selected six real applications consist a wide range from molecular biosciences (ExaML and NAMD), atmospheric sciences (HOMME and WRF) to physics (MILC) as well as astronomical sciences (GADGET), which cover a large portion of the scientific fields running on Kraken.

To quantify this proximity between applications and a benchmark, we calculate the distance of each baseline kernel against the cluster of test applications as follows. The MIN/MAX/AVG is the nearest/farthest/average Euclidean distance between a kernel and the test applications. A smaller Prox
value under each criteria implies a stronger similarity between the benchmark and the applications. The top three closest baseline kernels and their distances are shown Table 4. Clearly FFT, PTRANS, and LatencyBdh are the most realistic HPC benchmarks. Furthermore, we notice that M_FFT (Global FFT) has the most performance similarity to the six representative real-world scientific applications and “provide coverage of the major communication and computational patterns”, which is a key feature an HPC benchmark should include in Dongarra and Heroux’s position paper [9].

<table>
<thead>
<tr>
<th></th>
<th>MIN</th>
<th>MAX</th>
<th>Group Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>M_FFT</td>
<td>0.193</td>
<td>M_FFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.922</td>
<td>0.872</td>
</tr>
<tr>
<td>2nd</td>
<td>LatencyBdh</td>
<td>0.225</td>
<td>*_FFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.933</td>
<td>0.882</td>
</tr>
<tr>
<td>3rd</td>
<td>*_FFT</td>
<td>0.257</td>
<td>LatencyBdh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.939</td>
<td>0.914</td>
</tr>
</tbody>
</table>

Table 4: Closeness Ranking between HPCC benchmarks and real-world applications under 3 criteria.

6 Related Work

Over the years, statistical methods have been utilized for performance study in the HPC community. Ahn and Vetter’s [5] evaluated statistical techniques in the analysis of massive PAPI hardware performance counter events. For an application, data from seven PAPI counters were recorded for each MPI task in each instrumented code region, and then k-means clustering, PCA, and factor analysis were applied to characterize the performance data. Their approach can identify which tasks in the applications have similar performance counter metrics and hence reduce the data volume. Our work differs from theirs in the following aspects. First, our VarCluster uses Pearson correlation as the distance measure, while theirs uses the classical Euclidean distance. Secondly, PCA is an indispensable ingredient in VarCluster, but their use of PCA is merely a filter to clustering. Finally, we create an application scoring system which categorizes application based on synthetic and easily interpretable Performance Indices, while their research focuses on grouping application tasks with similar hardware performance behavior together. Hoste and Eeckhout [13] adopted genetic algorithm and PCA for data reduction. Vetter and Reed [20] used projection pursuit for the same purpose. Their works aim to pick important or interesting dimensions out of multidimensional performance data, whereas ours takes all dimensions into consideration in the derivation of new metrics. The use of HPCC in performance modeling and prediction has been studied before. Pfeiffer and Wright [18] utilized linear regression to fit the execution time of scientific applications using the HPCC kernels’ speeds and latencies. Chen et al. [8], on the other hand, examined what metrics are the best predictors for scientific application performance. They concluded that bandwidth of strided accesses to main memory, or bandwidth of random accesses to L1 cache, yields more accurate forecast than flops, on which Top 500 ranking is based.

7 Conclusion and Future Work

Performance evaluation and analysis have been a core topic in the HPC research, and High-Performance Linpack (HPL) as the long-established standard for measuring computing performance has been challenged in recent years. In this paper, we proposed a statistical approach which combines Variable Clustering (VarCluster) and Principal Component Analysis (PCA) to rigorously compare a benchmark’s adequacy and representativeness of real-world HPC workloads. Our scheme gives rise to an application scoring system in which three Performance Indices (PIs) are enough to capture the essence of the original high-dimensional performance space. Our experiments on the HPC Challenge (HPCC) benchmark suite and six scientific applications demonstrated that HPCC’s Fast Fourier Transform kernel, rather than the HPL on which Top500 is based, is more representative of the HPC workloads we tested.

In the future we plan to populate a comprehensive database with performance data of real scientific applications and benchmarks on a variety of supercomputers, so a better ranking and correlation of performance of applications and benchmarks can be easily studied and assessed.
Acknowledgment

This research used resources at the National Institute for Computational Sciences supported by the National Science Foundation. This research was also supported in part by the National Science Foundation under award number 1025159.

References