A new sustained system performance metric for scientific performance evaluation

Miwako Tsuji¹ · William T. C. Kramer² · Jean-Christophe Weill³ · Jean-Philippe Nominé³ · Mitsuhsa Sato¹

Accepted: 27 November 2020 / Published online: 2 January 2021
© The Author(s) 2021

Abstract

Because of the increasing complexities of systems and applications, the performance of many traditional HPC benchmarks, such as HPL or HPCG, no longer correlates strongly with the actual performance of real applications. To address the discrepancy between simple benchmarks and real applications, and to better understand the application performance of systems, some metrics use a set of either real applications or mini applications. In particular, the Sustained System Performance (SSP) metric Kramer et al. (The NERSC sustained system performance (SSP) metric. Tech Rep LBNL-58868, 2005), which indicates the expected throughput of different applications executing with different datasets, is widely used. Whereas such a metric should lead to direct insights on the actual performance of real applications, sometimes more effort is necessary to port and evaluate complex applications. In this study, to obtain the approximate performance of SSP representing real applications, without running real applications, we propose a metric called the Simplified Sustained System Performance (SSSP) metric, which is computed based on several benchmark scores and their respective weighting factors, and we construct a method evaluating the SSSP metric of a system. The weighting factors are obtained by minimizing the gap between the SSP and SSSP scores based on a small set of reference systems. We evaluated the applicability of the SSSP method using eight systems and demonstrated that our proposed SSSP metrics produce appropriate performance projections of the SSP metrics of these systems, even when we adopted a simple method for computing the weighting factors. Additionally, the robustness of our SSSP metric was confirmed via computation of the weighting factors based on a smaller set of reference systems and computation of the SSSP metrics of other systems.

Keywords Benchmarks · Performance evaluation · Key performance indicators · Sustained System Performance (SSP)

✉ Miwako Tsuji
miwako.tsuji@riken.jp

Extended author information available on the last page of the article

1 Introduction

Benchmarks play important roles in various stages of system life cycles, including design, development, implementation and maintenance, where they are used to estimate, evaluate, and ensure the effectiveness of a system. The High-Performance Linpack (HPL) kernel benchmark [14, 29] is one of the most widely used benchmarks in the world of high-performance computing. Linpack solves a dense linear system of equations, such as $y = Ax$. Because Linpack uses lower–upper (LU) factorization, its dominant computations are double-precision general matrix multiplications, which are relatively easy to tune to achieve the theoretical peak performance of processors. However, because of the increasing complexities of modern systems and applications, HPL performance no longer correlates strongly with the actual performance of real applications [13]. To address the discrepancy between HPL and real applications at performance levels, new benchmarks such as the High-Performance Conjugate Gradient (HPCG) [13] have been proposed. HPCG performance is known to be strongly influenced by memory bandwidth. Meanwhile, some systems with relatively few bytes per flop (B/F) also work efficiently for real applications.

Another approach to understanding system performance is to use a set of real applications, or real applications represented by mini applications. Several sets of mini applications, most of which are combinations of applications with different characteristics from different scientific fields [9, 17, 18, 23, 25, 32, 38], have been proposed. These sets of mini applications have not been provided with a single metric for system performance, to enable easy comparison between these systems. In response, Kramer et al. proposed the Sustained System Performance (SSP) metric, which is a system performance metric based on the time to solution and is able to accommodate any number of application areas [22]. In contrast to simple kernel benchmarks, these application metrics can lead to direct insights on the performance of systems in terms of their real workloads. However, execution of mini or real applications requires more time and effort than those required for the execution of simple kernel benchmarks. Furthermore, whereas methods for optimizing simple kernel benchmarks are well known, optimization of an application must be determined almost completely, without reliance on pre-existing methods.

In this paper, we propose a new system performance metric for application executions that is characterized by the ease of use of simple benchmarks, and the practicality of application sets. The proposed metric, which we refer to as the Simplified Sustained System Performance (SSSP) metric, is an extension of the SSP metric. Whereas the SSP metric focuses on the performance of systems based on actual executions of target applications, the SSSP metric focuses on the performance of applications based on the execution of simple benchmarks. SSSP uses a suite of simplified kernel benchmarks to produce performance projections for a suite of real applications. For the projection, we calculate a set of weighting factors that allow us to approximate the SSP metrics, which are computed over a set of applications using a set of simple benchmarks.

We performed experiments to investigate the applicability of the proposed metric. First, we executed a set of applications and a set of benchmarks on eight

different systems. In this case, we considered relatively small-scale applications and benchmarks. From the results of the benchmarks, the SSSP metric for each system was computed and compared with the corresponding SSP metric. The experiments demonstrated that the SSSP metric produced appropriate performance projections of the SSP metric, especially when we introduced weighting factors into the benchmarks in the approximation of the SSP. Therefore, if we applied both the applications and benchmarks on a few reference systems to obtain the weighting factors, then we could approximate the SSP metric using the SSSP metric in other systems, i.e., we could obtain the SSP metric by executing only benchmarks and calculating the weighted sum of the benchmark scores. Secondly, we executed the set of applications and set of benchmarks on two large systems. Whereas using benchmark scores from a large-scale system and weighting factors from small-scale experiments could be used to approximate the SSP metric of the large system, the result was less precise than for the same-scale cases.

We summarize the contributions of this paper as follows:

- A new performance metric called the SSSP metric, which can lead to appropriate insights on the application performance of systems without the need for actual execution of full scale, long running real applications,
- A method for constructing SSSP metrics based on weighting factors on simple kernel benchmarks,
- Experiments on eight different systems reveal that most well-known benchmarks cannot provide an appropriate performance metric on the application performance of systems.

The rest of this paper is structured as follows. In Sect. 2, we describe the Sustained System Performance (SSP) metric, which can be approximated using the proposed metric. Section 3 presents the Simplified System Performance (SSSP) metric, and Sect. 4 evaluates, via numerical experiments, how the SSSP metric can be used to approximate the SSP metric through numerical experiments. Sections 5 and 6 discuss the study, specifically its results, and examine related works. Lastly, Sect. 7 presents the conclusion of the study and discusses possible avenues for future research.

2 Background: Sustained System Performance (SSP)

In this section, we describe the SSP metric [22], which can be approximated using the proposed metric.

2.1 Overview of SSP metric

To evaluate high-performance computing (HPC) systems and compare different systems in terms of system effectiveness on sets of applications intended for use, the US National Energy Research Scientific Computing Center (NERSC)

proposed a metric called the Sustained System Performance (SSP) metric[22]. NERSC has used SSP for their procurement process and on-going system evaluations since 1998. Aside from system comparison, the purpose of the SSP method was to provide the following guidelines throughout the lifetime of a system and to facilitate the design of future systems [20]:

- Evaluating and/or selecting a system from among its competitors,
- Validating the selected system once it is built and deployed,
- Assuring the system performance remains as expected throughout the lifetime of the system,
- Helping guide future system designs.

Moreover, the SSP metric takes the following attributes of good metrics, as described in [24, 35] into account:

- proportionality
- reliability
- consistency
- independence
- ease of use
- repeatability
- avoid misleading conclusions

The objective of this paper is to propose a metric that leads to appropriate insights on the application performance of systems without the need for actual execution of real applications while maintaining all the attributes of good metrics. We construct our metric by approximating the SSP metric using a set of simplified benchmarks. We adapt the SSP metric as the approximation target of the proposed method, because the SSP metric is clearly defined, easy to apply, and is already being used for the procurement of several systems.

2.2 Definition of SSP metric

Here, we briefly describe how to compute the SSP metric. For more detail, please refer to Kramer et. al [21, 22].

Table 1 summarizes the notation used in defining the SSP metric.

Let I be a set of target applications i , and J_i be the data set for application i , where $1 \leq i \leq |I|$. Let the total reference operation count of application i executing data set j be $f_{i,j}$. The reference operation count is determined once for each application–data combination. The determination can be performed either via static analysis of source code or using performance counters on a reference system. Let $m_{i,j}$ be the concurrency of application i executing data set j . The amount of work for each processor for application i working on data set j , $m_{i,j}$, is then calculated as follows:

Table 1 Notations for SSP metric

Notation	Explanation
i	An application
I	A set of target applications, $i \in I$
j	A data set
J_i	A set of data sets for each application i , $j \in J_i$
$f_{i,j}$	The total reference operation count of application i executing data set j . In this paper, $f_{i,j}$ is the number of floating point operations.
$m_{i,j}$	The concurrency of application i executing data set j
$t_{i,j}$	the time-to-completion for application i executing data set j
$p_{i,j}$	The per-processor performance (PPP) of application i executing data set j $p_{i,j} = f_{i,j} / (m_{i,j} \times t_{i,j})$
N	The number of processors in a system

$$\frac{f_{i,j}}{m_{i,j}}. \quad (1)$$

If the time to completion of application i executed on data set j is $t_{i,j}$, then the per-processor performance (PPP) $p_{i,j}$ is calculated as follows

$$p_{i,j} = \frac{f_{i,j}}{m_{i,j} \times t_{i,j}} \quad (2)$$

Note that, sometimes, we use $p_{i,j}^s$ to indicate the per-processor performance of (i, j) on a system s .

The SSP metric for system s is obtained via multiplication of the sum of the PPPs over all applications for all data sets by the number of processors N in the system. Therefore, the SSP metric is calculated as follows:

$$\text{SSP}(s) = N \sum_{i \in I} \sum_{j \in J_i} p_{i,j} \quad (3)$$

$$= N \sum_{i \in I} \sum_{j \in J_i} \frac{f_{i,j}}{m_{i,j} \times t_{i,j}} \quad (4)$$

As shown in Eq. 4, the SSP metric is defined as the expected throughput of different applications, multiple data sets, and various concurrences.

For more details on the SSP metric please refer [7, 20, 27]. As an instance of the SSP metric, the Sustained Petascale Performance (SPP) metric and the set of applications used in the SPP metric have been presented at [39].

3 Simplified Sustained System Performance (SSSP) Metric

In this section, we present a new metric called the Simplified Sustained System Performance (SSSP) metric, which we designed to produce performance projections for applications without the need to evaluate the full-scale applications based on actual execution. The proposed SSSP metric is a correlated simplification of the already existing SSP metric. Whereas the SSP metric is computed over a set of applications, the SSSP metric, for ease of use, is applied to a set of simplified benchmarks. To approximate the SSP metric accurately, weighting factors for the benchmarks are computed to in such a way as minimize the differences between the SSP and SSSP metrics. Firstly, both the applications and benchmarks must be executed on several systems to optimize the weighting factors. After we obtain the weighting factors, we can approximate the SSP metric using simple benchmark results and the weighting factors without having to run any applications. Therefore, the advantage of the SSSP metric compared to the SSP metric is that the SSSP metric is able to produce an approximated score to the SSP metric without requiring the actual execution of applications, unlike when the SSP metric itself is used.

3.1 Concept of the SSSP metric

As described in Sect. 2, the SSP metric assigns a score to each system, indicating system performance with regard to its execution of applications. Because it is based on a wide variety of applications, the SSP metric can directly show the performance and time to solution of real workloads. However, we have to expend significantly much effort to port and optimize full applications to evolving architectures to compute SSP metric. Thus, initially, we have to better understand the working of the applications, prepare appropriate versions of libraries and compilers, and identify and eliminate bottlenecks as much as possible. To compare two or more systems, we will have to repeat these steps multiple times.

In this study, we extend the SSP metric to overcome its complexity. The new metric, which we have called the Simplified Sustained System Performance (SSSP) metric, replaces the set of applications used in the SSP metric with a set of simple benchmarks. In addition, we introduce weighting factors to adjust the impact of different benchmarks and to approximate the SSP metric using the SSSP metric. Therefore, the SSSP metric takes advantage of the benefits of simple benchmarks, such as ease of use, while also producing a performance projection of the application metric. With the SSSP metric, we aim to calculate performance projections for real applications by appropriately combining various benchmarks as shown in Fig. 1. The SSP metric is computed based on sets of applications for various fields of interest, such as astrophysics and life-sciences, whereas the SSSP metric is computed based on sets of benchmarks such as HPL and conjugate gradient method. Use of weighting factors on benchmarks allows SSSP metric to produce performance projections for applications.

Fig. 1 Overview of the Simplified Sustained System Performance (SSSP) metric, which estimates the performance of benchmarks to produce performance projections for real applications. The SSP metric is computed based on sets of applications for various fields of interest, such as astrophysics and life-sciences, whereas the SSSP metric is computed based on sets of benchmarks such as HPL and conjugate gradient method. Use of weighting factors on benchmarks allows SSSP metric to produce performance projections for applications

3.2 Definition of the SSSP metric

The definition of the SSSP metric is very simple and is almost the same as that of the SSP:

$$\text{SSSP}(s) = N \sum_{i \in I'} \sum_{j \in J_i} w_{i,j} p_{i,j} \quad (5)$$

where N is the number of nodes, I' is a set of benchmarks, J_i is a data set for each of the benchmarks, $p_{i,j}$ is the per-processor performance which has been defined in (2), and $w_{i,j}$ is the weighting factor, which will be defined to approximate $\text{SSP}(s)$ using $\text{SSSP}(s)$.

To ensure that the SSSP metric will be able to properly represent the performance of applications computed using the SSP metric, the SSSP metric must satisfy the following proportionality:

$$\forall s, s' \in S, \text{SSP}(s) < \text{SSP}(s') \Rightarrow \text{SSSP}(s) < \text{SSSP}(s') \quad (6)$$

where S is a set of different systems. This indicates that if a system s is better than another system s' based on the SSSP metric, then system s must also be better than system s' on the basis of the SSP metric.

Additionally, by assigning appropriate weighting factors w , we aim to produce an accurate approximation of the SSP metric using the SSSP metric:

$$\underset{\mathbf{w}}{\text{minimize}} \sum_{s \in S'} |\text{SSSP}(s) - \text{SSP}(s)|. \quad (7)$$

The weighting factors should be provided determined on the SSP and SSSP metrics of a set of reference systems.

4 Experiments

In this section, we investigate the applicability of the proposed metric.

4.1 Experimental procedure and environment

To validate the proposed metric, for the sake of simplicity, we focus on the per-processor performance (PPP) of the systems, rather than the SSP and SSSP metrics themselves:

$$P_{\text{ssp}}(s) = \sum_{i \in I} \sum_{j \in J_i} p_{i,j}, \quad (8)$$

$$P_{\text{sssp}}(s) = \sum_{i \in I'} \sum_{j \in J_i} w_{i,j} p_{i,j}, \quad (9)$$

where I is a set of applications, and $P_{\text{ssp}}(s)$ is the PPP metric of a system s based on I , I' is a set of benchmarks, and $P_{\text{sssp}}(s)$ is the PPP metric of a system s based on I' .

The weighting factors $w_{i,j} \in \mathbf{w}$ are introduced for the approximation of $P_{\text{ssp}}(s)$ based on $P_{\text{sssp}}(s)$; In some cases, we do not consider the weighting factors by assuming $w_{i,j} = 1$ for all pairs of (i, j) . The weighting factors are computed to minimize the difference between the $P_{\text{ssp}}(s)$ and $P_{\text{sssp}}(s)$ scores of a subset of systems;

$$\underset{\mathbf{w}}{\text{minimize}} \sum_{s \in S'} |P_{\text{SSSP}}(s) - P_{\text{SSP}}(s)| \quad (10)$$

while $w_{i,j} > 0$ for all pairs of (i, j) , where S' is a partial set of systems. With the obtained weighting factors, $P_{\text{sssp}}(s)$ are computed for the rest of the systems, to be compared with $P_{\text{ssp}}(s)$. If the gap between $P_{\text{sssp}}(s)$ and $P_{\text{ssp}}(s)$ is small, this indicates that we can approximate the performance of applications, $P_{\text{ssp}}(s)$, based only on the benchmark tests.

The overall procedure of the experiments is as follows:

1. Execute applications I (shown in sect. 4.2) and benchmarks I' (shown in sect. 4.3) on the eight systems $s \in S$ listed in Table 2 to obtain $p_{i,j}^s$.
2. Compute the weighting factors $w_{i,j}$ based on eq. 10.
3. Check how $P_{\text{SSSP}}(s)$ is able to approximate $P_{\text{SSP}}(s)$ for 2 cases:

Table 2 Specifications of supercomputers used in our experiments

	K-computer (K)	FX10 (FX10)	FX100 (FX100)
CPU	Sparc64 VIIIfx	Sparc64 IXfx	Sparc64 XIfx
	2GHz	1.65GHz	1.975GHz
Performance	128 GFlops	211.2 GFlops	1011.2 GFlops
# of cores	8 cores	16 cores	32 + 2 cores
L1-inst	32KB/core	32KB/core	32KB/core
L1-data	32KB/core	32KB/core	32KB/core
L2	6MB/node	12MB/node	12MBx2/node
L3	—	—	—
Memory	DDR3, 16GB	DDR3, 32GB	HMC, 32GB
Peak flops			
ratio of K	1.0	1.65	7.90
Network	Tofu interconnect	Tofu interconnect	Tofu interconnect2
	5GiB/s x 2	5GiB/s x 2	12.5GiB/s x 2
	HA-PACS (HA)	Blue Waters (BW)	Oakforest PACS
	(base cluster)		(OF)
CPU	Intel E5 6276	AMD interlagos	Intel Xeon Phi
	2.6GHz	2.3GHz	1.4GHz
Performance	332.8 GFlops	313.6 GFlops	3046 GFlops
# of cores	8 cores x 2 sockets	16 Bulldozer cores x2	68 cores
L1-inst	32KB/core	64KB/2core	32KB/core
L1-data	32KB/core	16KB/core	32KB/core
L2	256KB/core	2MB/2core	2MB/2core
L3	20MB/node	8MB/4core	(16GB MCDRAM)
Memory	DDR3	DDR3	DDR4+MCDRAM
	128GB	64GB	96+16GB
Peak flops			
Ratio of K	2.60	2.25	23.8
Network	Fat-Tree	Cray Gemini torus	Intel Omni-Path
		Interconnect	
	4GiB/s x 2	9.6GiB/s Injection	100Gibps
	Inti SKL Partition	Inti TX2 Partition	
	(SKL)	(TX2)	
CPU	Intel Xeon Platinum	Marvell ThunderX2	
	8168 2.7GHz	2.2GHz	
Performance	4147.2 GFlops	1124.4 GFlops	
# of cores	24 cores x 2 sockets	32 cores x 2 sockets	
L1-inst	32KB/core	32KB/core	
L1-data	32KB/core	32KB/core	
L2	1MB/core	256KB/core	
L3	33MB/socket	2MB x 2 / 4core	
Memory	DDR4, 192GB	DDR4, 256GB	
Peak flops			

Table 2 (continued)

	K-computer (K)	FX10 (FX10)	FX100 (FX100)
Ratio of K	32.4	8.78	
Network	InfiniBand	InfiniBand	

- when $w_{i,j} = 1$, and
 - when the weighting factors obtained at (2) are used.
4. Check the robustness of the proposed method by computing the weighting factors over 8 subsets of the 8 systems. Each subset consists of 7 systems, one of the 8 systems is excluded.
- Check how $P_{SSSP}(s)$ with the aforementioned weighting factors can be used to approximate $P_{SSP}(s)$.
 - Compare the sets of weighting factors computed from different sets of systems.
5. To examine the scalability of the proposed method, execute applications I and benchmarks J on two large-scale systems.
 6. Compute the weighting factors $w_{i,j}$ from the results at (5).
 7. Check how $P_{SSSP}(s)$ can be used to approximate $P_{SSP}(s)$ for 2 cases:
 - if the weighting factors obtained at (2) were used, and
 - if the weighting factors obtained at (6) were used.

As listed previously, we consider two types of experiments: small-scale experiments and large-scale experiments. Table 2 outlines the systems used in our experiments. Whereas we use all 8 systems for the small-scale experiments, we consider 2 systems (K, OFP) for the large-scale experiments. As shown in the table, the systems used in our experiments are of a wide range of varieties. For example, from the viewpoint of processor architectures, there are 3 Intel processors, 3 SPARC processors, 1 AMD processor, and 1 Arm processor.

4.2 Fiber miniapp suite

The Fiber Miniapp Suite [25] is used to evaluate $P_{SSP}(s)$ for comparison with $P_{SSSP}(s)$. Table 3 lists the applications included in the Fiber Miniapp Suite. The Fiber Miniapp Suite is a set of mini applications that are extracted from real applications and retain the essential characteristics of their parent applications.

Table 3 Applications in fiber miniapp suite

Application	Scientific Category (characteristics)
CCS-QCD 1.2.1	Quantum chromodynamics (structured grid Monte Carlo)
FFVC-MINI 1.0.1	Thermo-fluid analysis (3-dimensional cavity flow)
NICAM-DC-MINI 1.0	Climate (structured grid, stencil)
mVMC-MINI 1.0	Materials science (many-variable variational Monte Carlo)
NGS-Analyzer-MINI 1.0.1	Analyzes output data generated by a next-generation Genome sequencer (workflow of multiple tasks, I/O-intensive)
NTChem-MINI 1.1	Quantum chemistry, molecular electronic (structure calculation, dense matrix calculation)
FFB-MINI 1.0.1	Thermo-fluid analysis (finite element method, unstructured grid)

From among the different sets of applications and mini applications designed to evaluate different systems, as discussed earlier, we selected the Fiber Miniapp Suite because this set covers a variety of social and scientific challenges and a wide range of computational characteristics, as shown in Table 3 [26]. Moreover, because the Fiber Miniapp Suite includes relatively “traditional” HPC workloads, we would be able to understand the experimental results and the relationship between applications and benchmarks easily.

CCS-QCD [8] is a lattice quantum chromodynamics (QCD) clover-fermion solver, which is used in quantum field theory. FFVC-MINI is a Navier-Stokes solver for the 3-dimensional unsteady thermal flow of incompressible fluid, and is used to assist practical design in industrial fields. NICAM-DC-MINI [37] is derived from the Non-hydrostatic ICosahedral Atmospheric Model (NICAM), which is used for weather forecasting. mVMC-MINI analyzes the physical characteristics of strongly correlated electron systems, by configuring variational wave functions that closely represent the ground state of such electron systems. NGS Analyzer-MINI analyzes data generated from a next-generation sequencer to identify genomic variation, mutation, etc. This application is regarded as an I/O-intensive application. NTChem-MINI calculates the molecular electronic structure based on an ab initio quantum chemistry method. NTChem performs dense–dense matrix multiplications. FrontFlow/blue (FFB)-MINI is based on a general-purpose thermal-flow simulation program.

4.3 Benchmarks

For a preliminary example of a set of benchmarks, we consider HPL, Fast Fourier Transform in the East (FFTE), HPCG, Stream-Triad, and NPB-BT-IO, which may cover different characteristics of systems, such as computation, memory, I/O, and networks. Please note that we used reference implementations without any modification in these experiments, and thus the benchmark efficiencies are different from those presented previously in Top 500 reports and other announcements.

HPL, which is one of the most widely used benchmarks, solves a dense linear system of equations via LU factorization. FFTE is a package for computing discrete Fourier transforms of 1-, 2-, or 3-dimensional sequences. The performance of FFTE is affected to a large degree by the memory and network bandwidth. HPCG solves a sparse linear system using the conjugate gradient method and is designed to implement computational and data access patterns that more closely match a different and broad set of important applications [3, 13]. The performance of HPCG is known to be influenced by memory system performance. Stream-Triad is one of the Stream benchmarks that evaluate memory performance, whereas NPB-BT-IO is from a NAS parallel benchmark (NPB) and repeats MPI-IO to evaluate network and IO performance.

4.4 How to obtain weighting factors

According to equations (9, 10), weighting factors are introduced to minimize the gap between SSP and SSSP metrics. These weighting factors result from the simple learning algorithm, shown in Fig. 2, which is almost the same as the learning algorithm used to compute weighting factors in single layer perceptron in machine learning. However, unlike in the perceptron algorithm, we adopt a simple constant value of d to update the weighting factors. Furthermore, we do not use a sigmoid function, which is generally used in many perceptron systems. The maximum number of iterations n_{iter} and the threshold t of the errors between $P_{\text{SSP}}(s)$ and $P_{\text{SSSP}}(s)$ must be assigned by user.

For the optimization of w_{ij} , it is possible to adopt more complicated algorithms, as discussed later.

4.5 Parameters

We performed two sets of experiments: (1) small-scale experiments using eight systems, and (2) large-scale experiments using two large systems. For the former, applications and benchmarks were executed on the eight systems listed in Table 2. For the latter, the K and OFP systems identified in Table 2 were used. Tables 4, 5, 6 show the numbers of nodes, MPI processes, and OpenMP threads per MPI process for the applications and benchmarks, respectively, for the small-scale (Tables 4, 5) and large-scale (Table 6) experiments.

Inputs: $P_{SSP}(s)$ for all systems s
 $\mathbf{p}^s$ where $p_{i,j}^s$ is the performance of benchmark i executing data set j
 for all systems s
 The maximum number of iterations, n_{iter}
 t : Threshold
 d : parameter to update $w_{i,j}$

Outputs: $w_{i,j}$ for all pairs of (i, j)

Initialization: $w_{i,j} = 1.0$ for all (i, j)

```

for  $n = 1 \cdots n_{\text{iter}}$ 
  for all systems  $s$ 
     $P_{SSSP}(s) = \sum_{i \in I} \sum_{j \in J_i} w_{i,j} p_{i,j}^s$ 
    /* compute  $P_{SSSP}$  for all systems using the current weighting factors */
     $e(s) = P_{SSP}(s) - P_{SSSP}(s)$ 
    /*compute the error*/
  end for
  if  $|e(s)| < t$  for all  $s$ 
    break
  end if
  for all systems  $s$ 
    for all pairs of  $(i, j)$ 
       $w_{i,j} += e(s) \cdot d \cdot p_{i,j}^s$ 
      /*Update the weighting factors based on the error*/
    end for
  end for
  if at least of of  $w_{i,j}^s \leq 0$ 
    break
  end if
end for

```

Fig. 2 Algorithm to obtain the weighting factors

For the parameters used for the algorithm shown in Sect. 4.4, we have adopted $d = 1.0e^{-26}$, $n_{\text{iter}} = 50000000$, and $t = 0$, i.e., small thresholds and the large number of iterations, because at least one of the weighting factors becomes zero before the convergence. Note that because each iteration of the algorithm finishes very quickly, the weighting factors are computed in less than 1 min.

4.6 Results for small systems

Tables 7 and 8 show the $p_{i,j}^s$ for each application/benchmark i and data set j running on a system s . From the $p_{i,j}^s$, we compute the $P_{SSP}(s)$ and $P_{SSSP}(s)$.

Figure 3 visualizes the $P_{SSP}(s)$, $P_{SSSP}(s)$, and the PPPs for benchmarks. The scores are averaged over data set. Note that in the figure, $P_{SSSP}(s)$ does not consider any distinct weighting factors, i.e., $w_{i,j} = 1$ for all pairs of (i, j) . Furthermore, note that the Y-axis is set to a log scale and does not start from 0.0, where as the X-axis is labeled with the names of systems arranged in order of the $P_{SSP}(s)$. According to the figure, the performance rankings of the systems in terms of HPCG, Stream,

Table 4 Numbers of nodes, processes, and threads (from left to right in each cell) used for Fiber Miniapp Suite

App and data	K	FX10	BlueWaters	HA-PACS
CCS-QCD Class1	1, 1, 8	1, 1, 16	1, 1, 32	1, 1, 8
Class2	64, 64, 8	64, 64, 16	16, 64, 8	16, 64, 8
FFVC 256 ³	1, 1, 8	1, 1, 16	1, 1, 16	1, 1, 16
1024 ³	64, 64, 8	64, 64, 16	16, 64, 8	32, 64, 8
NICAM-CD				
gl05rl00z80pe5	5, 5, 8	5, 5, 16	2, 5, 8	3, 5, 8
gl05rl00z40pe10	10, 10, 8	10, 10, 16	3, 10, 8	5, 10, 8
mVMC tiny	1, 1, 8	1, 1, 16	1, 1, 16	1, 1, 16
Middle	64, 64, 8	64, 64, 16	16, 128, 4	64, 128, 8
NGSAnalyzer				
Dummy-genome	6, 6, 1	6, 6, 1	6, 6, 1	6, 6, 1
NTChem h2o	2, 2, 8	2, 4, 8	1, 4, 6	1, 2, 8
Taxol	32, 32, 8	32, 64, 8	8, 32, 8	16, 32, 6
FFB test	8, 8, 8	8, 8, 16	1, 8, 4	4, 8, 8
App and data	FX100	TX2	OFP	SKL
CCS-QCD Class1	1, 1, 32	1, 1, 32	1, 1, 64	1, 1, 24
Class2	32, 64, 16	4, 64, 8	8, 64, 8	6, 64, 4
FFVC 256 ³	1, 1, 32	1, 16, 8	1, 8, 8	1, 8, 6
1024 ³	32, 64, 16	4, 64, 8	8, 64, 8	4, 64, 3
NICAM-CD				
gl05rl00z80pe5	3, 5, 16	3, 5, 64	1, 5, 8	3, 5, 24
gl05rl00z40pe10	5, 10, 16	5, 10, 32	1, 10, 4	5, 10, 24
mVMC tiny	1, 1, 32	1, 1, 32	1, 1, 16	1, 1, 16
Middle	32, 64, 16	4, 64, 8	8, 64, 8	4, 64, 6
NGSAnalyzer				
Dummy-genome	6, 6, 1	1, 6, 1	6, 6, 1	1, 6, 1
NTChem h2o	1, 4, 8	1, 16, 8	1, 8, 8	1, 12, 8
Taxol	32, 128, 8	4, 64, 8	4, 32, 8	4, 24, 8
FFB test	4, 8, 16	1, 8, 8	1, 8, 8	1, 8, 4

and BT-IO are shuffled two or three times. For example, whereas SKL is better than TX2 in terms of $P_{SSP}(s)$, TX2 is better than SKL in terms of HPCG, and although OFP is better than FX100 in terms of $P_{SSP}(s)$, FX100 is better than FX100 in terms of HPCG. On the other hand, the performance rankings of systems in terms of $P_{SSSP}(s)$, HPL, and FFTE are shuffled only once. The swap TX2↔SKL is observed for FFTE, and the swap OFP↔TX2 is observed for $P_{SSSP}(s)$ and HPL. Moreover, the PPP for HPL is fairly larger than the $P_{SSP}(s)$, and the PPP for FFTE is smaller than the $P_{SSP}(s)$. It is obvious that HPL overestimates the performance of the OFP system. Because $P_{SSSP}(s)$ averages several benchmark scores, including HPL, it

Table 5 # of nodes, processes and threads (from left to right in each cell) used for benchmarks

Bench and data	K	FX10	BlueWaters	HA-PACS
HPL2.2 $N_s = 80000$	4, 32, 1	2, 32, 1	1, 32, 1	1, 16, 1
$N_s = 160000$	16, 128, 1	8, 128, 1	4, 128, 1	4, 32, 1
FFTE 6.0 512 ³	8, 8, 8	8, 8, 16	2, 8, 8	4, 8, 16
1024 ³	64, 64, 8	64, 64, 16	16, 64, 8	32, 64, 8
HPCG 3.0 256 ³	64, 512, 1	32, 512, 1	2, 64, 1	4, 64, 1
512 ³	64, 512, 1	32, 512, 1	2, 64, 1	4, 64, 1
Stream-Triad 2 ¹⁵	1, 1, 8	1, 1, 16	1, 1, 1	1, 1, 16
2 ²⁹	1, 1, 8	1, 1, 16	1, 1, 2	1, 1, 16
NPB BT-IO C	2, 16, 1	1, 16, 1	1, 16, 1	1, 16, 1
D	16, 64, 1	4, 64, 1	1, 64, 1	4, 64, 1
Bench and data	FX100	TX2	OFP	SKL
HPL2.2 $N_s = 80000$	2, 64, 1	1, 64, 2	1, 64, 1	1, 48, 1
$N_s = 160000$	8, 256, 1	4, 256, 2	4, 256, 1	4, 192, 1
FFTE 6.0 512 ³	4, 8, 16	1, 8, 8	1, 8, 8	2, 4, 12
1024 ³	32, 64, 16	4, 64, 8	4, 64, 8	6, 64, 4
HPCG 3.0 256 ³	16, 512, 1	1, 64, 2	4, 256, 1	2, 64, 1
512 ³	16, 512, 1	1, 64, 2	4, 256, 1	2, 64, 1
Stream-Triad 2 ¹⁵	1, 1, 32	1, 1, 2	1, 1, 32	1, 1, 48
2 ²⁹	1, 1, 32	1, 1, 128	1, 1, 16	1, 1, 48
NPB BT-IO C	1, 16, 1	1, 64, 1	1, 64, 1	1, 16, 1
D	2, 64, 1	1, 64, 1	1, 64, 1	2, 64, 1

seems reasonable for a swap between OFP and TX2 for both $P_{SSSP}(s)$ and HPL. Gaps between $P_{SSP}(s)$ and $P_{SSSP}(s)$ without weighting factors are also observed, i.e., $P_{SSSP}(s)$ without weighting factors always overestimate system performance. The reason for the gap is that the simple sum of benchmarks scores is increased by HPL, which generally results in high theoretical peak performance ratio.

Figure 4 shows the $P_{SSP}(s)$, $P_{SSSP}(s)$ without weighting factors, and $P_{SSSP}(s)$ using weighting factors. Note that the Y-axis is set to log scale and does not start from 0.0. In the figure, it is obvious that the $P_{SSSP}(s)$ using weighting factors was able to result in an appropriate performance projection of the $P_{SSP}(s)$. Although we still observe the swap between OFP and TX2 for $P_{SSSP}(s)$ with weighting factors, the difference between the $P_{SSP}(s)$ and $P_{SSSP}(s)$ with weighting factors is smaller than that between $P_{SSP}(s)$ and $P_{SSSP}(s)$ without weighting factors.

Table 9 lists the weighting factors for the benchmarks and data sets. This result shows that if we assign a lower weight to HPL, a higher weight to BT-IO, and weights close to unity in the remaining cases, the $P_{SSSP}(s)$ provides a more precise approximation of $P_{SSP}(s)$. The result also suggests that we do not have to evaluate all the benchmarks for the approximation of the SSP metric, and that we would be able to omit HPL from the calculation. However, if the requirements for

Table 6 # of nodes, processes and threads (from left to right in each cell) used for applications and benchmarks (I/O-intensive benchmarks and applications are omitted)

App and data	K	OFP
CCS-QCD Class3	512, 512, 8	64, 512, 8
Class4	8000, 8000, 8	1000, 8000, 8
FFVC 1024 ³	512, 512, 8	64, 512, 8
4096 ³	4096, 4096, 8	512, 4096, 8
NICAM-CD gl08rl03z80pe640	640, 640, 8	80, 640, 8
gl09rl04z80pe2560	2560, 2560, 8	320, 2560, 8
mVMC 1024	1024, 1024, 8	128, 1024, 8
8192	8192, 8192, 8	1024, 8192, 8
NTChem c60atc60h28	128, 128, 8	16, 128, 8
c54h18x2_cc-pvdz	4, 4, 8	1, 4, 8
FFB 69	512, 512, 8	64, 512, 8
115	4096, 4096, 8	512, 4096, 8
Bench and data	K	OFP
HPL2.2 $N_s = 480000$	1024, 8192, 1	128, 8192, 1
$N_s = 960000$	8192, 65536, 1	1024, 65536, 1
FFTE 6.0 4096 ³	1024, 1024, 8	128, 1024, 8
4096 ³	8192, 8102, 8	1024, 8192, 8
HPCG 3.0 512 ³	512, 4096, 1	64, 4096, 1
1024 ³	512, 4096, 1	64, 4096, 1
Stream-Triad 2 ¹⁵	1, 1, 8	1, 1, 32
2 ²⁹	1, 1, 8	1, 1, 16

a target set of applications for a system are unclear, it is recommended to consider a set of benchmarks that cover the different characteristics of systems to construct the SSSP metric.

To validate the robustness of the SSSP metric with weighting factors, we also compute the weighting factors in such a way as to minimize equation (10) for seven systems by excluding one of the eight systems from the computation of weighting factors. As an example of the robustness checks for our proposed method, Figure 5 compares the $P_{SSSP}(s)$ using weighting factors obtained from all eight systems and the $P_{SSSP}(s)$ using weighting factors obtained from seven systems, where the TX2 cluster has been omitted. As shown in the figure, the $P_{SSSP}(s)$ based on the seven systems also approximated the $P_{SSP}(s)$ for the TX2 system, which had been excluded in the computation of the weighting factors. Moreover, the $P_{SSSP}(s)$ scores for the weights of the seven systems are similar to those for the weights of the eight systems. This indicates the robustness of the proposed method against different systems.

Figure 6 shows the sum of the differences between $P_{SSP}(s)$ and $P_{SSSP}(s)$ over all systems $s \in S$, shown in equation (7). The X-axis is labeled with the sets

Table 7 PPP (GFlops) for applications

App and data	K	FX10	BlueWaters	HA-PACS
CCS-QCD Class1	18.4	24.7	17.8	29.9
Class2	11.0	11.0	15.9	19.7
FFVC 256 ³	14.0	21.5	11.2	35.2
1024 ³	12.7	17.4	52.8	67.3
NICAM-CD gl05rl00z80pe5	5.87	7.11	3.97	3.53
gl05rl00z40pe10	4.90	5.18	5.10	4.75
mVMC tiny	4.31	3.41	7.18	5.70
middle	19.1	25.2	29.8	23.6
NGSAnalyzer dummy-genome	0.011	0.009	0.0197	0.0467
NTChem h2o	10.9	12.2	43.1	76.8
taxol	61.2	67.3	142	199
FFB test	5.72	6.19	10.7	4.89
App and data	FX100	TX2	OFP	SKL
CCS-QCD Class1	23.1	55.7	25.4	96.9
Class2	33.7	67.8	38.9	60.3
FFVC 256 ³	39.7	154	226	242
1024 ³	34.8	133	228	190
NICAM-CD gl05rl00z80pe5	18.9	3.30	16.0	5.23
gl05rl00z40pe10	15.1	3.80	11.8	7.59
mVMC tiny	4.18	9.25	5.70	18.3
Middle	91.2	184	67.1	293
NGSAnalyzer dummy-genome	0.013	0.178	0.0129	0.348
NTChem h2o	57.3	103	64.3	364
taxol	199	715	531	1174
FFB test	21.1	11.8	5.82	24.0

of weighting factors used to compute $P_{SSSP}(s)$. “SSSP w/o weight” indicates $w_{ij} = 1$ for all pairs of benchmarks and data sets, whereas “SSSP w/ weight” indicates the use of the weighting factors obtained based on p_{ij} for applications and benchmarks over all systems. On the other hand, “SSSP w/o K” indicates the use of the weighting factors obtained based on p_{ij} from all systems except K; calculations of weighting factors excluding the other systems are identified in similar ways. As shown in the figure, it is obvious that using the weighting factors can reduce the gap between the SSP metric and the SSSP metric. It is also shown that the sum of the differences can be reduced successfully even if we omit a system in the computation of the weighting factors. Note that because we stopped the optimization of the weighting factors when at least one of the weighting factors becomes zero, $w_{ij} \leq 0$, the sum of the differences does not converge to the threshold.

Table 8 PPP (GFlops) for benchmarks

Bench and data	K	FX10	BlueWaters	HA-PACS
HPL2.2 $N_s = 80000$	96.5	149	235	297
$N_s = 160000$	97.2	150	235	292
FFTE 6.0 512 ³	3.41	3.50	7.95	9.36
1024 ³	2.74	2.93	5.75	5.68
HPCG 3.0 256 ³	1.79	2.48	5.84	6.82
512 ³	1.52	2.01	5.76	6.90
Stream-Triad 2 ¹⁵	3.86	3.64	2.26	13.1
2 ²⁹	2.49	3.64	0.648	4.89
NPB BT-IO C	9.03	13.2	13.9	28.3
D	5.19	7.96	19.7	16.6
Bench and data	FX100	TX2	OFP	SKL
HPL2.2 $N_s = 80000$	545	905	1390	2110
$N_s = 160000$	547	875	1340	1990
FFTE 6.0 512 ³	15.0	28.4	14.4	24.7
1024 ³	13.4	28.3	24.2	18.4
HPCG 3.0 256 ³	10.6	19.2	7.09	13.0
512 ³	10.9	19.2	6.69	13.0
Stream-Triad 2 ¹⁵	13.1	1.06	10.9	27.5
2 ²⁹	8.52	17.9	5.36	14.7
NPB BT-IO C	18.4	78.7	32.0	43.6
D	23.2	101	31.7	84.5

Fig. 3 PPPs for SSP, SSSP, HPL, FFTE, HPCG, Stream, and BT-IO obtained from 8 systems, averaged over data set. Note that the $P_{SSSP}(s)$ does not consider weighting factors. The Y-axis is set as log scale, and it does not start from 0.0

Figure 7 shows the weighting factors for the benchmarks, computed via minimization of the equation (10) over all eight systems, and over each of the eight combinations of seven of the systems. As shown in the figure, we can obtain similar

Fig. 4 PPPs for SSP, SSSP without weighting factors and SSSP using weighting factors, computed from application and benchmark results for eight systems. Note that Y-axis is set to log scale and does not start from 0.0

Table 9 Weighting factors computed to minimize gap between SSP and SSSP, and weighting factors averaged over data set

Bench and data	Weight	Average
HPL2.2 $N_s = 80000$	0.63	0.315
$N_s = 160000$	0.00	
FFTE 6.0 512^3	0.862	0.992
1024^3	1.12	
HPCG 3.0 256^3	1.09	1.09
512^3	1.09	
Stream-Triad 2^{15}	1.05	1.09
2^{29}	1.13	
NPB BT-IO C	1.43	1.68
D	1.92	

weights for the same benchmarks even when any one of the systems is excluded from the computation. It also indicates the robustness and applicability of the SSSP metric. For BT-IO, the weighting factor becomes higher than those of the other benchmarks when we omit TX2 in the computation of the weighting factors because TX2 demonstrates its most salient performance in terms of BT-IO, as shown in Figure 3 and Table 8. Nonetheless, the $P_{SSSP}(s)$ obtained using the weighting factors computed by all systems except TX2 accord closely with the $P_{SSSP}(s)$ obtained using the weighting factors computed by all systems, as already shown in Fig. 5.

4.7 Results for large systems

In Sect. 4.6, we discussed the experimental results for relatively small numbers of nodes. Here, we consider the results for large systems of up to 8192 nodes. Unfortunately, we were able to perform large-scale experiments for only two systems

Fig. 5 PPPs for SSP, SSSP ($w_{ij} = 1$), SSSP using weights obtained from eight systems, and SSSP using weights obtained from seven systems (all except TX2 cluster). Y-axis is set to log scale and does not start from 0.0

Fig. 6 Sum of differences between $P_{SSP(s)}$ and $P_{SSSP(s)}$ over all systems $s \in S$. Sets of weighting factors for $P_{SSSP(s)}$ are computed using 8 systems (SSSP w/ weight), using all systems without K (SSSP w/o K), using all systems without FX10 (SSSP w/o FX10), etc. For reference, the sum of differences between $P_{SSP(s)}$ and $P_{SSSP(s)}$ without weighting factors (SSSP w/o weight) is also shown. Note that Y-axis is not log scale

because of the sizes of the available systems. These two large systems are K and OFP, which are also used in Sect. 4.6. In addition, we had to omit I/O-intensive benchmarks and applications such as BT-IO and NGSAnalyzer because of the disk quota limits in these systems.

Tables 10 and 11 outline the PPPs for applications and benchmarks, respectively. We compute

- (1) $P_{SSP(s)}$ using p_{ij} in Table 10,

Fig. 7 Weighting factors computed over all systems, and 7 systems in the absence of each system

Table 10 PPP p_{ij} (GFlops) for applications

App and data	K	OFP
CCS-QCD Class3	11.1	39.4
Class4	13.3	38.8
FFVC 1024 ³	10.6	65.2
4096 ³	12.5	60.2
NICAM-DC		
gl08r03z80pe640	5.10	6.79
gl09r04z80pe2560	5.13	6.73
mVMC 1024	19.1	61.1
8192	5.83	60.5
NTChem c60atc60h28	69.7	721
c54h18x2_cc-pvdz	10.1	86.5
FFB 115	6.30	5.45
69	5.83	5.28

- (2) $P_{SSSP}(s)$ using p_{ij} in Table 11,
- (3) $P_{SSSP}(s)$ using p_{ij} in Table 11 and weighting factors obtained by minimizing gap between (1) and (2),
- (4) $P_{SSSP}(s)$ using p_{ij} shown in Table 11 and weighting factors obtained using the 8 small systems (shown in Fig. 9).

Table 11 PPP p_{ij} (GFlops) for benchmarks

Bench and data	K	OFP
HPL $N_s = 480000$	83.3	874
$N_s = 960000$	68.3	665
FFTE 4096 ³	1.48	9.24
4096 ³	1.60	5.40
HPCG 512 ³	1.78	6.99
1024 ³	1.46	6.99
Stream-Triad 2 ¹⁵	3.86	10.9
2 ²⁹	2.49	5.36

Fig. 8 PPPs for SSP, SSSP (w/o weighting factors), SSSP (w/ weighting factors computed from small systems), SSSP (w/ weighting factors computed from large systems) in K and OFP (Left). The ratio of PPPs for SSSP (w/o weighting factors) and SSSP (w/ weighting factors computed from small systems), and the ratio of PPPs for SSSP (w/o weighting factors) and SSSP (w/ weighting factors computed from large systems) (Right)

Figure 8 (left) shows the $P_{SSP}(s)$ and $P_{SSSP}(s)$ for the K and OFP systems. As shown in the figure, the $P_{SSSP}(s)$ metrics computed using the weighting factors obtained based on the small systems are less precise than those using the weighting factors obtained based on the large systems, but still lead to appropriate insights on the application performance of the systems.

Figure 8 (right) shows the ratios of $P_{SSSP}(s)$ (without weighting factors) and $P_{SSSP}(s)$ (with weighting factors computed based on small systems), and those of $P_{SSSP}(s)$ (without weighting factors) and $P_{SSSP}(s)$ (with weighting factors computed based on large systems), i.e.,

$$\frac{P_{SSSP}(s) \text{ without } w_{ij}}{P_{SSSP}(s) \text{ with } w_{ij} \text{ from small/large systems}}.$$

Fig. 10 Execution times of applications and benchmarks on K and OFP for large-scale applications and benchmarks. The execution times for HPCG are always 2000 due to its regulation. While the average execution time over applications is ranged from 841 (K) to 3563 (OFP) seconds, average execution time over benchmarks is ranged from 692 (OFP) to 741 (K) seconds

execution time over benchmarks can be reduced from 114 seconds to 756 seconds, which is comparable with that over applications. For the large-scale performance evaluations, whereas the average execution times over applications ranged from 841 (K) to 3563 (OFP) seconds, the average execution times over benchmarks ranged from 692 (OFP) to 741 (K) seconds even though the execution time for HPCG remains fixed to 2,000 seconds. On the other hand, some of the SSP metric instances require more than a few hours—for example, the NICAM-gl09r04z80pe2560 on OFP requires approximately 6 hours—the maximum execution time for the SSSP metric except the HPCG, is only 1054 seconds (18 minutes). Therefore, the SSSP metric is demonstrated to be able to provide its score more easily and quickly than the SSP metric is able to do for large-scale systems. Because the performance of simple benchmarks, such as Stream and HPL, can be expected based on system specifications, not all benchmarks will have to be executed.

5 Discussion

We have presented the SSSP metric and the method for constructing the metric. The SSSP metric is computed using the results of simple benchmarks and the weighting factors for these benchmarks. Because the weighting factors are obtained using the SSSP metric for the approximation of the SSP metric, the SSSP metric is guaranteed to be able to quantify the performance of systems at executing applications.

Therefore, the SSSP metric can accomplish the general characteristics of simple benchmarks, such as ease of use, while providing a more practical score than those of simple benchmarks.

Our experiments in Sect. 4 suggested that we should assign the smallest weighting factor to HPL in the approximation of the performance of systems at executing applications, because HPL is more likely to overestimate the performance of systems at executing applications and to mislead the expectations of application developers. By contrast, the SSSP metric can discount the HPL score based on the relationship between benchmarks and applications.

In our experiments, we did not include accelerator-based systems because some of the applications are not written for such systems. However, according to the results of our experiments, which were executed over a wide variety of different processor architectures, and the results of the robustness checks, we expect that the SSSP metric and the method developed for the computation of the SSSP metric can be applied to accelerator-based systems. Nonetheless, experiments on accelerator-based systems would be necessary to update the SSSP metric such that it would be able to approximate different sets of applications, including artificial intelligence applications, in the future.

During the design phase of a system, where we have to use performance estimation tools and several simulators, more time is required to estimate the performance of the system than to estimate using a real system. In that case, the SSSP metric has an advantage that complex applications will not have to be executed on the simulators. Sometimes, the calculated weighting factors for the benchmarks for a set of target applications may suggest the architectural characteristics that require enhancement. For example, if a memory-intensive benchmark has a high weighting factor, increasing the memory bandwidth should enhance the performance of the system at executing the target applications.

Another advantage of the SSSP metric over a simple set of applications or benchmarks is that the end result is very simple: a single value for a single system. Various methods have been developed for evaluating a system not only from a single viewpoint, such as the HPL score, but also from diversified standpoints. For example, the HPC Challenge benchmarks consist of seven different benchmarks, including HPL and Stream [2, 15]. Moreover, as discussed earlier, several sets of mini applications have been proposed for investigating the performance of systems at executing applications easily [9, 17, 18, 23, 25, 32, 38]. Whereas the results of these applications and benchmarks provide a radar chart for understanding a system, the SSSP metric provides a single score concentrating the radar chart to a single value.

In addition to the general purposes of benchmarks, such as selecting a system or confirming its performance, the simplicity and flexibility of the proposed method for constructing the SSSP allow us to better understand a set of applications or single application. For example, if a specific application is approximated based on a set of benchmarks and weighting factors, then the best system for the application from among different systems can be suggested based on the benchmark results and weighting factors. Moreover, since the SSSP metric can be approximated to the SSP metric instances of various workload sets, it can be used to evaluate various middleware, system software, virtual environments, and tools such as failure detection and fault tolerance in distributed systems [16], error correction in a virtual environment [19], and so on.

6 Related works

Although we have already discussed some of related works, such as HPL and HPCG, in the previous sections, we discuss more about the related works in greater detail in this section.

The HPC Challenge (HPCC) benchmark consists of seven different benchmarks including HPL and Stream [2]. The NAS parallel benchmarks [6, 28] are a set of benchmarks designed to help evaluate the performance of parallel supercomputers. The NAS parallel benchmarks are derived from computational fluid dynamics (CFD) applications and include kernels, pseudoapplications, OpenMP+MPI hybrid codes, and I/O tests. The Scalable Heterogeneous Computing (SHOC) benchmark suite, which is designed for evaluating the performance of scalable heterogeneous systems [11] is a set of kernels and mini applications implemented in OpenCL and CUDA. Unlike the SSSP metric, these collections of benchmarks do not provide a single value for a single system.

The Standard Performance Evaluation Corporation (SPEC) [12, 36] provides sets of benchmarks and suites of applications for evaluating different aspects of systems, such as CPU, communication, and storage. The score of a target system for a set is computed based on the geometric mean of normalized performance ratios of the reference system. Although some of the SPEC benchmarks, such as SPEC HPC 2002, focus on real-application sets, these real application sets are not as easy to run as simple benchmarks [5]. Additionally, some of the SPEC benchmarks, such as the SPECint, consist of simple kernel benchmarks, which are designed to evaluate a specific aspect of a system.

To evaluate the performance of systems at executing applications, there are several suites of mini-, proxy-, or real applications. ESIF-HPC-2 [10] is a benchmark suite that collects kernels and applications to measure computational and I/O performance for systems and parts of systems ranging from single nodes to full HPC systems. The ESIF-HPC-2 has been designed to cover the specific needs of the energy research community. The Exascale Computing Project (ECP) proxy applications [1, 31], which are developed as part of the ECP, are a collection of small, simplified codes that allow application developers to share important features of large applications without forcing collaborators to assimilate large and complex code bases. Most application suites do not provide their own method for computing a single value as a quantitative performance discriminator. Although we can compute the SSP metric over these application suites, tuning and porting applications remains inconvenient and costly to perform.

Although we have focused on the correspondence between simple kernel benchmarks and mini-applications, there are several works which consider the correspondence proxy applications and their parental real applications, and the correspondence between simple kernel benchmarks and mini-applications. Aaziz et. al. [4] proposed a method for characterizing the relationship between real and proxy applications in the ECP application using collection of runtime data such as instructions per cycle, and cache miss ratio and analyzing them. Sharkawi et al. [34] also used hardware performance counter data for the HPC applications and SPEC CFP 2006 benchmark sets from a single base system to model the behavior of the HPC applications as a function of the benchmarks. A genetic algorithm was used to identify the “best” group of benchmarks that explain each

HPC application. SWAPP [33], which was proposed by the same authors, applies the same scheme to project the performance of the MPI communication of HPC applications. Whereas they consider a single system in constructing the performance projection function, we constructed the performance projection function over several different systems. Additionally, unlike these methods, our method does not require any information from hardware counters.

Summarizing the above and benchmarks discussed in Sect. 1, most existing benchmarks, metrics, sets of benchmarks, and sets of applications have one or more of the following drawbacks:

- does not consider the application performance of systems;
- does involve complicated processes in the execution of real-world applications, such as porting and tuning;
- does not provide a single value metric for the easy understanding and comparison of systems; and/or
- requires more detailed information, such as hardware counters.

7 Conclusion

In this paper, we proposed a new performance metric, called the Simplified Sustained System Performance (SSSP) metric. The SSSP metric provides an easy approximation of the Sustained System Performance (SSP) metric, which indicates the performance of systems at executing applications. Whereas the SSP metric leads to direct insights on the performance of systems, it requires porting and evaluation of complex applications, which require significant amounts of effort. On the other hand, the SSSP metric can be computed via the performance evaluation of simple benchmarks and a set of weighting factors for the benchmarks. The weighting factors should be obtained by minimizing the differences between the SSP and SSSP metrics on a small set of reference systems. As demonstrated in the experiments, the SSSP metric was able to approximate the SSP metric more closely than any other single benchmark, such as HPL or HPCG. One of important future works is how exactly to approximate the SSP metric with respect to the SSSP metric. Although we have adopted a relatively simple learning algorithm in this paper, many algorithms have been developed for the computation of the weighting factors to minimize the equation (7). For example, we have confirmed that the CVXOPT [30], which is a software package for convex optimization, provides appropriate approximations. The Deep Learning/Machine Learning (DL/ML) algorithms have also been used in various optimization fields and should be strong candidates for our approximation.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is

not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. ECP proxy applications. <https://proxyapps.exascaleproject.org/>
2. HPC challenge. <http://www.hpcchallenge.org/>
3. Hpcg benchmark. <http://www.hpcg-benchmark.org/>
4. Aaziz O, Cook J, Cook J, Juedeman T, Richards D, Vaughn C (2018) A methodology for characterizing the correspondence between real and proxy applications. In: Proceedings of 2018 IEEE International Conference on Cluster Computing (CSTER). IEEE
5. Armstrong B, Bae H, Eigenmann R, Saied F, Sayeed M, Zheng Y (2006) Hpc benchmarking and performance evaluation with realistic applications. In: The SPEC Benchmark Workshop, pp. 1–11
6. Bailey D, Harris T, Saphir W, van der Wijngaart R, Woo A, Yarrow M (1995) The nas parallel benchmarks 2.0. Tech. Rep. NAS Technical Report NAS-95-020
7. Bauer G, Hoefer T, Kramer W, Fiedler R (2012) Analyses and modeling of applications used to demonstrate sustained petascale performance on blue waters. In: Proceedings of the annual meeting of the cray users groups CUG-2012
8. Boku T, Ishikawa K, Minami K, Nakamura Y, Shoji F, Takahasi D, Terai M, Ukawa A, Yosie T (2012) Multi-block/multi-core SSOR preconditioner for the QCD quark solver for K computer. In: Proceedings of The 30th International Symposium on Lattice Field Theory
9. Center for exascale simulation of combustion in turbulence: ExaCT. <http://exactcodesign.org/>
10. Change CH, Carpenter IL, Jones WB (2020) The ESIF-HPC-2 benchmark suite. In: Proceedings of the Principles and Practice of Parallel Programming (PPoPP) 2020
11. Danalis A, Marin G, McCurdy C, Meredith J, Roth P, Spafford K, Tipparaju V, Vetter J (2010) The scalable heterogeneous computing (shoc) benchmark suite. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems-ASPLOS. <https://doi.org/10.1145/1735688.1735702>
12. Dixit KM (1991) The spec benchmarks. *Parallel Computing* 17:1195–1209
13. Dongarra J, Heroux MA (2013) Toward a new metric for ranking high performance computing systems. Tech. Rep. SAND2013-4744, Sandia National Laboratories
14. Dongarra J, Bunch CMJ, Stewart GW (1979) LINPACK user's guide. SIAM, Philadelphia
15. Dongarra J, Luszczek P (2013) HPC challenge: design, history, and implementation highlights. In: Contemporary high performance computing: from petascale toward exascale. Taylor and Francis
16. Ghosh S (2014) Distributed systems: an algorithmic approach, 2nd edn. Chapman and Hall/CRC
17. Heroux MA, Doerfler DW, Crozier PS, Willenbring JM, Edwards HC, Williams A, Rajan M, Keiter ER, Thorn-quist HK, Numrich RW (2009) Improving performance viamini-applications. Tech Rep SAND2009-5574, Sandia National Laboratories
18. Karlin I, Bhatle A, Keasler J, Chamberlain BL, Cohen J, DeVito Z, Haque R, Laney D, Luke E, Wang F, Richards D, Schulz M, Still C (2013) Exploring traditional and emerging parallel programming models using a proxy application. In: 27th IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS 2013)
19. Kononenko K (2016) An approach to error correction in program code using dynamic optimization in a virtual execution environment. *J Supercomputing* 72:845–873. <https://doi.org/10.1007/s11227-015-1616-4>
20. Kramer WTC (2013) Measuring sustained performance on blue waters with the SPP metric. In: Proceedings of the Annual Meeting of the Cray Users Groups CUG-2013
21. Kramer WT (2008) Percu: a holistic method for evaluating high performance computing systems. Ph D thesis, EECS Department, University of California, Berkeley. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-143.html>
22. Kramer WT, Shalf J, Strohmaier E (2005) The NERSC sustained system performance (SSP) metric. Tech Rep LBNL-58868
23. Lawrence Livermore National Lab: LULESH. <https://codesign.llnl.gov/lulesh.php>
24. Lilja DJ (2000) Measuring computer performance a practitioner's guide. Cambridge University Press, New York

25. Maruyama N (2013) Mini-app effort in Japan. In: SC13 BoF: Library of mini-applications for exascale component-based performance modeling (Presentation)
26. Maruyama N (2014) Miniapps for enabling architecture-application co-design for exascale supercomputing. In: 19th Workshop on sustained simulation performance (Presentation)
27. Mendes CL, Bode B, Bauer GH, Enos J, Cristina Beldica WTK (2015) Deployment and testing of the sustained petascale blue waters system. *J Comput Sci* 10:327–337
28. NAS parallel benchmarks: <http://www.nas.nasa.gov/publications/npb.html>
29. Petitet A, Whaley RC, Dongarra J, Cleary A HPL-a portable implementation of the high-performance linpack benchmark for distributed-memory computers. <http://www.netlib.org/benchmark/hpl/>
30. Python Software for Convex Optimization. CVXOPT. <https://cvxopt.org/>
31. Richards D, Bhatele1 A, Aaziz O, Cook J, Finkel H, Homerding B, McCorquodale P, Mintz T, Moore S, Pavel R (2018) FY18 proxy app suite release, milestone report for the ecp proxy app project. Tech Rep LLNL-TR-760903, Exascale computing project
32. Sandia National Laboratories: Mantevo project. <https://mantevo.org/>
33. Sharkawi S, DeSota D, Panda R, Stevens S, Taylor V, Wu X (2012) SWAPP: a framework for performance projections of hpc applications using benchmarks. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, pp. 1722–1731. IEEE
34. Sharkawi S, DeSota D, Panda R, Indukuru R, Stevens S, Taylor V, Wu X (2009) Performance projection of hpc applications using spec cfp2006 benchmarks. In: Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2009), pp. 1–12. IEEE
35. Smith JE (1988) Characterizing computer performance with a single number. *Commun ACM* 31(10):1202–1206
36. Standard performance evaluation corporation: standard performance evaluation corporation. <https://www.spec.org/>
37. Terai M, Yashiro H, Sakamoto K, ichi Iga S, Tomita H, Satoh M, Minami K (2014) Performance optimization and evaluation of a global climate application using a 440m horizontal mesh on the K computer. In: Proceedings of 2014 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2014)
38. University of Illinois: Charm++:MiniApps. <http://charmplusplus.org/miniApps/>
39. William TC (2013) Kramer: blue waters and the future of @scale computing and analysis, 4th AICS Workshop

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Miwako Tsuji¹ · William T. C. Kramer² · Jean-Christophe Weill³ · Jean-Philippe Nominé³ · Mitsuhsa Sato¹

William T. C. Kramer
wtkramer@illinois.edu

Jean-Christophe Weill
Jean-Christophe.WEILL@cea.fr

Jean-Philippe Nominé
Jean-Philippe.NOMINE@cea.fr

Mitsuhsa Sato
msato@riken.jp

¹ RIKEN Center for Computational Science, Kobe, Japan

² University of Illinois at Urbana-Champaign, Champaign, USA

³ CEA, Paris, France