2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)

A Metric for Evaluating Supercomputer Performance in the Era of Extreme Heterogeneity

Brian Austin, Chris Daley, Douglas Doerfler, Jack Deslippe, Brandon Cook,
Brian Friesen, Thorsten Kurth, Charlene Yang, Nicholas J. Wright
Lawrence Berkeley National Laboratory
Berkeley, CA USA
baustin, csdaley, dwdoerf, jrdeslippe, bgcook, bfriesen, tkurth, cjyang, njwright@lbl.gov

***Abstract*—When acquiring a supercomputer it is desirable to specify its performance using a single number. For many procurements, this is usually stated as a performance increase over a current generation platform, for example machine A provides 10 times greater performance than machine B. The determination of such a single number is not necessarily a simple process; there is no universal agreement on how this calculation is performed and each facility usually uses their own method. In the future, the landscape will be further complicated because systems will contain a heterogeneous mix of node types, and, by design, every application will not run on every node type. For example, at the National Energy Research Scientific Computing Center (NERSC) the Cori supercomputer contains two node types, nodes based on dual-socket Intel Xeon (Haswell) processors and nodes based on Intel Xeon Phi (Knights Landing) processors. However, NERSC evaluated these two partitions separately, without utilizing a single, combined performance metric. NERSC will be deploying its next-generation machine, NERSC-9, in the year 2020 and anticipates that it too will be a heterogeneous mix of node types. The purpose of this paper is to describe a single performance metric for a heterogeneous system.**

I. Introduction

Performance metrics have great value for evaluating and comparing supercomputers. Such measurements are important for a number of reasons: measuring delivered performance against requirements collected from the scientific user community, satisfying stakeholder concerns on the appropriateness of a proposed or delivered system, setting relative job charge factors between systems at the center, etc. With the acquisition of each new system, the increase in capability delivered by the system must be quantified, and in the context of high performance computing (HPC) procurements, one might say that "you get what you measure" is as true as the adage "you get what you pay for."

While metrics that capture peak-performance in near ideal conditions, such as those obtained by the TOP500 list [1], provide a description of the capability of an HPC system, it is often hard to translate differences in measured LINPACK FLOPs between systems to performance differences in complex scientific simulation or data-analysis codes that may depend on a wide array of system features (e.g. cache and memory capacities, bandwidths and latencies, high speed network characteristics and I/O speeds and feeds). For this reason, it has become common to use an approach based on workload representative benchmark applications. The National Energy Research Scientific Computing center (NERSC) has historically used the Sustained System Performance (SSP) metric [2], which later evolved into the Sustained System Improvement (SSI) metric introduced in Section III. Both of these metrics characterize a system based on the average performance of a suite of representative benchmarks and implicitly assume a homogeneous compute architecture.

The slowing of Moore's Law has prompted many technologists to suggest that the largest gains in performance and efficiency are likely to be achieved by tuning processors for specific tasks rather than relying on incremental improvements to high-performance general purpose processors. In this vision of "extreme-heterogeneity", HPC platforms are comprised of integrated collections of diverse processing elements, each designed to accelerate different components of the workload [3]. To enable comparisons of heterogeneous (or homogeneous) architectures with a consistent value metric, and to quantify the value of hardware specialization, this paper introduces heterogeneous extensions to SSP and SSI.

Many HPC centers have already found utility in deploying platforms with heterogeneous node-types. For example, NERSC's current Cori system contains a mixture of node types, with either Intel Haswell or Intel Knights Landing processors, and the Blue Waters system at NCSA includes Cray XC-6 with AMD CPUs and XK-7 nodes accelerated with NVIDIA Kepler GPUs. The predominant accelerators used in HPC today (as exemplified by the Summit system at Oak Ridge Leadership Computing Facility) are graphics processing units (GPUs). Section V illustrates the use of SSI to determine the optimal fraction of nodes with and without GPUs, subject to total budget constraints. In this scenario, it becomes important to balance the performance benefits of each accelerator against its cost. Although this example considers only two node types, hetero-SSI can be applied, without modification, to arbitrarily many accelerators.

II. Scalable System Improvement (SSI) metric

A. Related work and historical background

The SSI metric was developed to evaluate supercomputer performance and influence architectural direction. It combines aspects of the Sustained System Performance (SSP) and Capability Improvement (CI) metrics used in previous NERSC procurements.

978-1-7281-0182-8/18/$31.00 ©2018 IEEE
DOI 10.1109/PMBS.2018.00011

SSP: The SSP metric provides a measure of the sustained floating point performance of a suite of applications [2]. It is defined as the *geometric* mean of the performance in FLOP/s/node of each application in the suite multiplied by the total number of nodes in the system:

$$\mathrm{SSP} = N \left\langle \frac{F_i/t_i}{n_i} \right\rangle \tag{1}$$

where the angle brackets denote a mean operation over all applications, i, N is the total number of compute nodes in the platform, and F_i, t_i and n_i are the total number of FLOPs, runtime, and nodes used to run benchmark i. The final metric has units of FLOP/s and represents the average rate of computational work on a supercomputer running the computing center workload.

CI: The CI metric provides a measure of the capability improvement over a reference system. It is motivated by the desire to execute larger problems more efficiently on a new supercomputing platform. The CI of each application benchmark is defined as the increase in problem size multiplied by the run time performance speedup relative to a reference platform [4].

$$\mathrm{CI} = \left\langle c_i \, \frac{t_i^{\mathrm{ref}}}{t_i} \right\rangle \sim \left\langle \frac{F_i/t_i}{F_i^{\mathrm{ref}}/t_i^{\mathrm{ref}}} \right\rangle \tag{2}$$

Historically, the CI was calculated using the *arithmetic* mean of the application benchmarks CIs. The similarity on the RHS of Equation 2 emphasizes that t_i and t_i^{ref} correspond to different, weak-scaled inputs to the same application; the problem size factor, c_i, is analogous to F_i/F_i^{ref}, though there may not be an exact equivalence. The c_i factor enabled the inclusion of capability-scale benchmarks that could not be performed on the reference platform.

The choice of metric is important because it has an effect on the proposed architecture. Vendors will be influenced by optimizing the metric and may propose an architecture that is mismatched to the current needs or strategic direction of the computing center. For example, the SSP score is often improved by running application benchmarks on the fewest possible compute nodes to minimize scaling costs. However, this is unrealistic since a user has an expected turnaround time to complete the stages of their scientific workflow. It is also potentially harmful because it can lead to undesirable architectural trade-offs, e.g increasing memory capacity per node by sacrificing network performance. To address these issues, the SSI metric combines the most useful features of SSP and CI into a compact and consistent form.

SSI: SSI relates the average application throughput of one platform to that of a reference system.

$$\mathrm{SSI} = \left\langle N \, \frac{c_i/t_i}{n_i} \right\rangle \Bigg/ \left\langle N^{\mathrm{ref}} \, \frac{1/t_i^{\mathrm{ref}}}{n_i^{\mathrm{ref}}} \right\rangle \tag{3}$$

Along with the SSI definition, we propose two optional "knobs" that may influence the evaluation of an architecture. First, the use of a *weighted* mean allows the designer of the benchmark suite to prioritize applications according to their importance to the workload. Because different applications are likely to emphasize different aspects of the platform, weights have a direct impact on the optimal system balance.

The second knob is a requirement that $t_i \leq t_i^{\mathrm{ref}}$, which enforces users' natural expectations that each platform generation will improve the time-to-solution. This constraint is necessary because, for example, the unconstrained SSI score is often improved by running the benchmarks on the fewest number of nodes to avoid scaling inefficiencies. This case could lead to an undesirable architectural trade-off; sacrificing network performance for increased memory per node.

SSI provides multiple mechanisms to influence the proposed architecture and attach value to architectural features that are expected to benefit a workload in the coming years. The benchmark team specifies the weight of each application in the average, the capability scaling factor (c_i), and the node count (n_i^{ref}) used to obtain the reference figure of merit (FOM) such as t_i^{ref}. For example, the NERSC-9 benchmark team collected reference times at node counts when strong-scaling overheads are relatively high. The intended effect of selecting aggressive t_i^{ref} was to emphasize the need for network performance in order to obtain significant gains over the threshold speedup of 1.0.

In limiting cases, SSI simplifies to either of its SSP or CI predecessors. If the geometric mean is used, along with uniform weights, $c_i = 1$, and t_i is unconstrained, then Equation 3 can be rearranged easily to show that $\mathrm{SSI} = \mathrm{SSP} \,/\, \mathrm{SSP}^{\mathrm{ref}}$.

The connection to CI can be seen by interpreting SSI as an average of application CIs, each multiplied by a utilization factor, $U_i = (N/n_i)/(N^{\mathrm{ref}}/n_i^{\mathrm{ref}})$. The machine utilization is introduced to reward application throughput and machine size relative to a reference machine. (Utilization is accounted for in the SSP metric.) For example, if an application benchmark is run on 200 nodes of a proposed 1000 node machine and the reference run was performed on 160 nodes of a 500 node machine then the utilization term would be $\frac{1000}{200}/\frac{500}{160} = 1.6$. Defining a speedup factor, $S_i = t_i^{\mathrm{ref}}/t_i$ that compares the runtime (or other application-specific figure-of-merit such as time-per-iteration), leads to a particularly compact expression based on this interpretation:

$$\mathrm{SSI}^{*} = \langle c_i \, U_i \, S_i \rangle \tag{4}$$

B. Which mean to use

For SSI, NERSC uses the geometric mean. However, one could also use the arithmetic or the harmonic mean in the calculation. The choice of which mean to use depends on the use case. The use of the Pythagorean means in benchmarking has been extensively analyzed in [5] and [6]. Although there are some contradictions in their conclusions, a reasonable summary is: 1) the arithmetic mean should only be used when performance metrics are stated in time, 2) the harmonic mean should be used when performance is expressed as a rate, such as operations/second, and 3) if performance is normalized to a specific baseline, such as SSI does using speedup, the geometric mean is appropriate but the aggregate performance measure such as total time should be used before normalization. However, for SSI we do normalize each individual

result and then calculate the mean. The reason for this is each benchmark application's FOM is independent and the FOM varies due to what the benchmark author is trying to measure and can be total time, grind time, operations/second, updates/second, etc. Hence we cannot add results together to calculate total time, nor total work, nor total rate as is recommended for correctness when using the arithmetic and harmonic means.

For NERSC, another reason for using geometric mean is practicality. Historically we have used the geometric mean in the calculation of SSP and have compared performance improvement from one machine to the next by taking the ratios of respective machine SSPs. SSI is exactly this only more explicit and hence it maintains consistency and allows us to better understand improvements through machine generations.

III. Heterogeneous SSI formulation

In order to compare platforms consisting of architecturally distinct partitions with different performance properties, it is useful to generalize the preceding performance metrics to quantify the combined performance of the heterogeneous system. The most straightforward modification is to sum the contributions of each partition to the total throughput of each application, then compute their mean. This is analogous to a heterogeneous SSP in which subscript p has been added to the quantities in Equation 1 to indicate that they are measured independently for each partition:

$$\text{hetero-SSP} = \left\langle \sum_p N_p \frac{F_i/t_{i,p}}{n_{i,p}} \right\rangle \tag{5}$$

However, this simple approach of hetero-SSP does not account for the potential benefits of *specialization*. For example, if applications A and B can run on partitions P and Q, but only application B gets accelerated on Q, then the combined throughput can be increased if A cedes its Q-allocation to B and vice versa. The benefit of this trade-off is particularly dramatic when A has not been ported to Q.

The heterogeneous SSI metric resembles a ratio of hetero-SSPs, but allows specialization by introducing $f_{i,p}$, which denotes the fraction of partition p devoted to benchmark i:

$$\text{hetero-SSI} = \frac{\left\langle \sum_p f_{i,p}\, N_p \frac{c_i/t_{i,p}}{n_{i,p}} \right\rangle}{\left\langle \sum_p^{\text{ref}} f_{i,p}^{\text{ref}}\, N_p^{\text{ref}} \frac{1/t_{i,p}^{\text{ref}}}{n_{i,p}^{\text{ref}}} \right\rangle} \tag{6}$$

The $f_{i,p}$ variables can be optimized to increase SSI subject to the constraints $\sum_i f_{i,p} \leq 1\, \forall p$, which prevents oversubscription of any partition. For applications that cannot run on a given partition, the optimal value of $f_{i,p}$ is zero.

Hetero-SSI can be rewritten particularly compactly when the reference platform consists of only one partition, shared equally among benchmarks:

$$\text{hetero-SSI}^* = B \times \left\langle \sum_p f_{i,p}\, c_i\, U_{i,p}\, S_{i,p} \right\rangle \tag{7}$$

TABLE I
GPU readiness categorization

GPU Status	Description	Fraction
Enabled	Most features are ported with good performance.	46%
Proxy	Kernels in related codes have been ported.	19%
Unlikely	A GPU port would require major effort.	11%
Unknown	GPU readiness cannot be assessed at this time.	24%

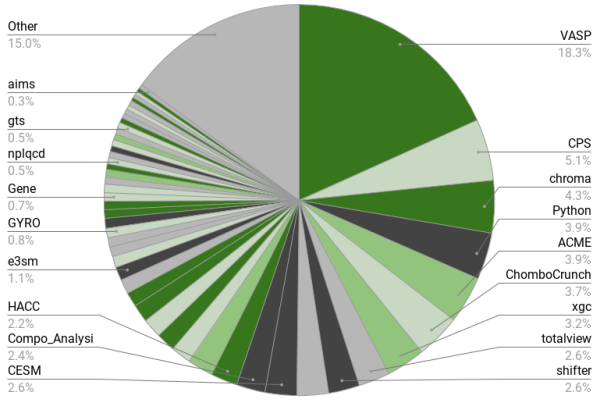


Fig. 1. NERSC workload by core hours used, July 2017 - August 2018. Colors correspond to the GPU readiness levels in Table I

The additional factor of B is equal to the number of benchmarks and is needed to normalize hetero-SSI with its non-heterogeneous form. In the following sections, we use the simplified form shown in Equation 7 to compute SSI.

IV. Benchmarking key NERSC applications on GPU based Supercomputers

A. Analysis of the NERSC workload and the GPU readiness of NERSC applications

Over 600+ different applications are regularly run on NERSC systems. In 2014 the top 13, 25 and 50 codes comprised 50%, 66% and 80% of the CPU time [7].

NERSC regularly conducts an analysis of the workload on its systems to understand application requirements and guide procurements. Historically this analysis has included science area, application code, algorithm, job size, thread usage, memory usage, library usage and I/O characteristics. The data is sourced from system monitoring tools and job accounting databases.

In the most recent analysis, an assessment of GPU readiness of the top NERSC applications was added. The top-50 cycle-consuming applications were binned into the GPU-readiness categories listed in Table I. The assessment was primarily based on a literature survey, but conversations with application developers and empirical analysis of some codes was also done. For a code to be considered "enabled" for a GPU, most features of the code must be ported in an officially recognized and available branch of the application. Figure 1 shows the breakdown of the NERSC workload weighted by net CPU time consumed during the period of July 2017 to August 2018.

B. Benchmark suite construction

Benchmark selection is critical to ensuring that SSI (or other composite application metrics) accurately estimate the throughput of the anticipated workload. The selected benchmarks must include all of the workload's key algorithms, and weights may be used to quantify their prevalence. In some cases, the predominant applications may not be suitable benchmarks due to licensing restrictions, exceedingly complex build procedures or export controls, making it necessary to represent those codes with mini-applications or other algorithmically similar codes. For example, Figure 1 indicates that the most heavily used application at NERSC is VASP, which is subject to license restrictions. In our benchmark suite, we use Quantum Espresso to represent VASP because it implements many of the same capabilities and algorithms, but is open source. Judicious benchmark selections may also include applications that are likely to become important during the lifetime of the system; DeepCAM does not contribute significantly to NERSC's current workload, but is included among our benchmarks to represent the rapidly growing space of deep learning algorithms.

The following paragraphs give brief descriptions of a suite of applications and their performance on various platforms. The purpose of this section is *not to be a formal analysis comparing the performance of a GPU based system to a CPU based system*, but to provide a reasonable basis for applying the heterogeneous SSI calculations. Some of the results were collected empirically using current generation platforms, while others were taken from published literature.

Quantum Espresso: Quantum ESPRESSO is an open-source Materials Science application that computes the electronic structure of materials via Density Functional Theory (DFT) and related approaches in a plane-wave basis set. The code is largely written in FORTRAN-90 with MPI+OpenMP support throughout. The GPU branch is currently under-development, utilizing CUDA FORTRAN and accelerated math libraries. The problem benchmarked was a SCF optimization.

MILC: The MILC code is a widely used, computationally intense application designed to compute the interactions of quarks and gluons as described by the theory of quantum chromodynamics (QCD). The computational grid is a four-dimensional space-time grid (x, y, z, t) with quark fields, defined as three-dimensional complex vectors, at the grid points and gluon variables, defined as 3x3 unitary matrices, defined at the 'links' between grid points. The most computationally intense part is the conjugate gradient solver which determines how the motion of the quarks is affected by the gluons [8]. The baseline code has fine-grain parallelism implemented with OpenMP directives, mostly on loops over all grid points in the lattice [9]. There is a GPU optimized version using the QUDA library for lattice QCD calculations [10], and an Intel CPU optimized version using the QPhiX library [11]. These optimized versions are used for this study.

StarLord: StarLord [12] is a proxy application for the adaptive-mesh compressible radiation hydrodynamics code Castro [13]–[16]. It contains a subset of capabilities available on Castro, chiefly a solver for 3-D equations for compressible hydrodynamics, as well as an equation of state which is computed in each cell on the grid. The AMReX [17] framework on which StarLord is based is written primarily in C++, while the majority of the kernels which compute floating-point arithmetic in StarLord are written in Fortran. StarLord uses CUDA Fortran to launch kernels on GPUs. The most time-consuming functions in a typical StarLord simulation are the calculation of the equation of state in each cell, and the piecewise-parabolic method calculations for evolving the fluid flow. These functions and most others are performed in CUDA kernels; consequently, the majority of execution time in StarLord is spent in the GPU, with little data being copied between GPU and the host CPU.

DeepCAM: DeepCAM is a Google TensorFlow [18], [19] based application for segmenting out extreme weather phenomena in multivariate data obtained from climate simulations using the 0.25 degree resolution Community Atmospheric model (CAM5). DeepCAM is based on the recently introduced DeepLabv3+ [20] neural network architecture and performs a per-pixel classification task on the input data. It employs a convolutional encoder based on ResNet-50 [21]. In contrast to a decoder based on a combinations of bilinear extrapolations and conditional random fields, we employ a de-convolutional decoder which can reconstruct information with pixel-level resolution. Most of the time in a forward and backward pass is spent in these (de-)convolutional layers and the total number of FLOPs for each pass and layer can easily be computed. The performance in terms of FLOP/s can then be obtained by timing the forward and backward passes. We also compute a throughput number in terms of images/s, which reflects the number of samples (16-channel 768x1152 images) the code can ingest per second for a complete pair of forward and backward pass. In order to achieve high performance, TensorFlow makes use of the optimized libraries cuDNN [22] on NVIDIA GPUs and MKL-DNN [23] on contemporary Intel CPUs. The code can be executed in half (FP16) or single precision (FP32), where native FP16 support is only available for the NVIDIA Volta architecture.

GTC: GTC is a Particle In Cell (PIC) code used to study plasma microturbulence in magnetic confinement fusion devices known as tokamaks. GTC solves the 5D gyrokinetic Vlasov equation on a 3D toroidal grid. The stages in time evolution are: interpolate particle charge onto the grid, solve the gyrokinetic Poisson equation, compute an electric field, interpolate the electric field onto the particles, and move the particles. The particle to grid and grid to particle interpolations typically dominate run-time and are challenging to execute efficiently on GPUs because of synchronization requirements and irregular memory access pattern, respectively. The version of GTC used in this paper was developed by Princeton Plasma Physics Laboratory and is referred to as GTC-P [24]. It is parallelized with MPI and OpenMP and contains CUDA kernels for both of the interpolation stages and for moving

TABLE II
PLATFORMS USED TO COLLECT PERFORMANCE RESULTS

Name	CPU	GPU	Node CPU:GPU
NVIDIA PSG P100	Intel Haswell	NVIDIA P100	2:4
OLCF Summit	IBM Power-9	NVIDIA V100	2:6
OLCF Summit-dev	IBM Power-8	NVIDIA P100	2:4
CSCS Piz Daint	Intel Haswell	NVIDIA P100	1:1
NERSC Cori-P1	Intel Haswell	N/A	2:0
NERSC Cori-P2	Intel KNL	N/A	1:0

particles [25]. This version of GTC is different from the University of California-Irvine version of GTC, which is written in OpenACC and co-developed with NVIDIA [26], [27].

C. Application benchmark results

Our baseline CPU-only architecture consists of dual-socket Haswell nodes similar to the Cori-P1 system. To estimate the performance of a hypothetical accelerated node with two Haswell sockets and four NVIDIA V100 GPUs, we measured GPU speedups using a variety of platforms listed in Table II and applied the following transformations to our measured results:

- A 1.5x performance gain when going from a P100 GPU to a V100 GPU (based on memory bandwidth improvement)
- A 4x performance gain when going from 1 GPU to 4 GPUs (i.e. perfect weak scaling between GPUs)
- A 0.5x performance gain when going from a KNL CPU or Power 9 CPU to a Haswell CPU
- A 1x gain when going from a Power 8 CPU to a Haswell CPU

Although some of these assumptions may be unrealistically optimistic, our goal is to provide a data set that is sufficiently accurate for architectural exploration, not precise performance models for each benchmark.

The results are shown in Table III and the transformed values are in Table IV. From a practical perspective, it is likely that some applications cannot be accelerated or that there will not be sufficient resources to move the application to accelerators. To account for this component of the workload, Table IV also includes three applications that are not optimized for GPUs. For these entries, the GPU speedup is specified to be 1.0x, as a GPU node contains the same dual-socket CPU socket configuration and can be used without the GPU.

V. APPLYING HETEROGENEOUS SSI

In this section we apply the use of heterogeneous SSI on the results of the previous section and illustrate its use when all applications run on all partitions. We then show that for optimal performance it's necessary to use node specialization and placement by optimizing the $f_{i,p}$ factors.

A. An iso-cost method to determine partition sizes

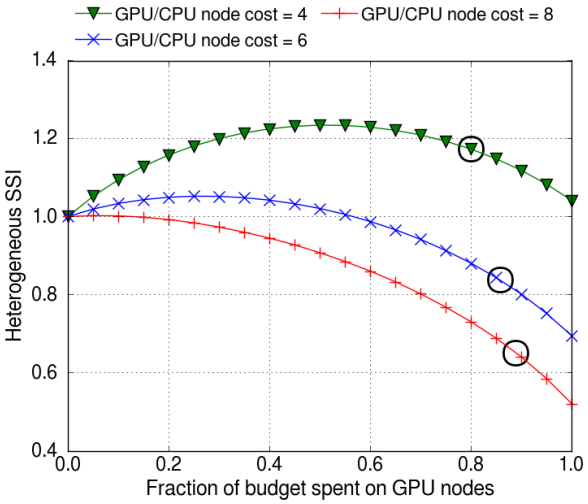
We apply heterogeneous SSI by doing an iso-cost sensitivity analysis for the workload in Table IV. It should be noted that the speedup values in Table IV are agnostic of node cost and assume both the CPU and GPU partitions are *full system size*. To perform an iso-cost comparison we must adjust the utilization values relative to the size of the respective partitions which depends on the CPU:GPU cost ratio. For example, in this analysis we assumed a compute node budget of $10,000$ units. If the cost of a CPU node is 1 unit and a GPU node is 8 units and if we set the fraction of the budget for GPU nodes at 0.4 then the number of GPU nodes in the system is $\frac{10{,}000 \times 0.4}{8} = 500$, the number of CPU nodes is $10,000 - 500 \times 8 = 6,000$ and the total system size is $6,500$ nodes. The utilization factor in the SSI metric takes into account the size of the system, so if the GPU partition goes from $10,000$ units to 500 units then we must adjust the utilization factor accordingly.

Prior to the iso-cost refactoring, we choose to set the initial $c_i\, U_{i,p} = 1.0$ to make this example easy to understand. This is valid because n_i is the same for both the CPU and GPU results contained in Table IV and hence before iso-cost refactoring $U_{i,p}$ is the same for both node types in a full 10,000 node configuration. We then refactor the SSI contribution of each application by adjusting $U_{i,p}$ accordingly based on the iso-cost node counts. For example, using the partition sizes calculated above, the adjusted $c_i\, U_{i,p}\, S_{i,p} = \frac{500}{10{,}000} = 0.05$ for the GPU partition and $= \frac{6{,}000}{10{,}000} = 0.6$ for the CPU partition. Applying this to the Quantum Espresso speedup in Table IV, its new contribution becomes $15.12 \times 0.05 = 0.756$ and $1.0 \times 0.6 = 0.6$ for the GPU and CPU partitions respectively. We then apply this adjustment to all the applications and calculate a heterogeneous SSI, illustrated in Figure 2 where the budget used for the GPU partition varies from 0% to 100%.

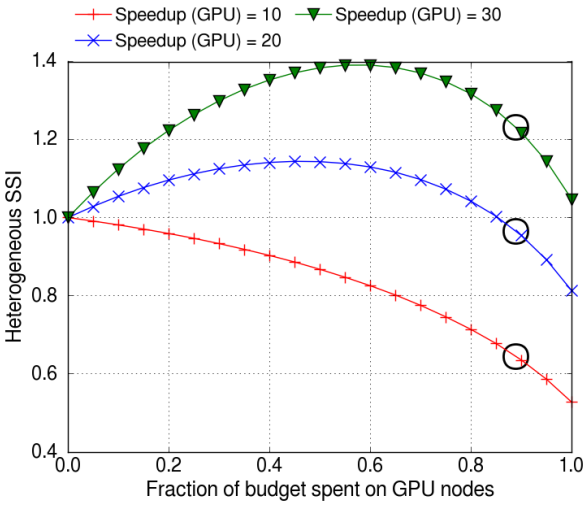
It should be noted that for a CPU only partition, the SSI value is 1.0. Hence, if $SSI{>}1.0$, then there is a benefit provided by the GPU partition. When the CPU:GPU cost ratio is 1:8, the added performance of the GPU nodes does not offset the cost premium for any fractional budget. As the cost ratio decreases, we see there is a benefit and for the case of the ratio being 1:4 the performance improvement is 1.23x for a fractional budget of 0.52. Figure 2 also illustrates the impact of improving the performance of the codes on the GPU partition. The nominal SSI of just the GPU accelerated applications is $\sim$10. As improvements are made to the codes the SSI will increase and hence change the threshold for which the GPU partitions performance offsets a cost premium. Here we show performance improves as the SSI of the GPU partition increases to 20 and then 30 where the improvement approaches 1.4x with a fractional budget of 58%. This illustrates the importance of activities such as facility centers of excellence in which the system and technology providers work closely with the application teams to ensure their codes run optimally.

TABLE III
APPLICATION SPEEDUP WHEN COMPARING A CPU+GPU PLATFORM TO A CPU PLATFORM. THE RESULTS ARE OBTAINED BY USING ALL CPUS AND GPUS IN A NODE UNLESS THE PLATFORM NAME IS SUFFIXED BY TEXT IN PARENTHESIS, I.E. (CPU COUNT USED : GPU COUNT USED).

Name	CPU Platform	CPU+GPU Platform	Nodes	Speedup	Notes
Quantum Espresso	Piz Daint (1:0)	Piz Daint	2	5.04	
MILC	Cori-P2	Piz Daint	256	1.69	Used a 128x128x128x128 lattice
Starlord	Cori-P1	Summit-dev	16	1.59	
DeepCam	Cori-P2	Summit (1:1)	1	15.35	Used 1 worker in FP32 precision
GTC-P	PSG P100 (2:0)	PSG P100	1	5.36	Used problem size A [25] with 4 toroidal domains



(a) Iso cost, CPU:GPU cost ratio of 1:4, 1:6 and 1:8



(b) Iso cost, SSI of the GPU is 10, 20 and 30

Fig. 2. An iso-cost heterogeneous SSI sensitivity analysis for the workload in Table IV. In a) the cost of a GPU node relative to a CPU node varies from 1:8 to 1:4. If a GPU node costs 8 times more than a CPU node then a GPU partition may not be justified, assuming the nominal Speedup (GPU) = 10. But as the price ratio drops the GPU partition becomes more desirable. In b) the CPU:GPU cost is fixed at 1:8 and the Speedup of the applications in the GPU partition varies from 10 to 30. This illustrates the impact of improving the performance of applications that use the GPU partition. The circle on each iso cost curve represents a system with 50% CPU nodes and 50% GPU nodes.

TABLE IV
ESTIMATED APPLICATION SPEEDUP WHEN COMPARING A DUAL-SOCKET HASWELL NODE WITH 4 NVIDIA V100 TO A DUAL-SOCKET HASWELL NODE.

Name	Nodes	CPU Speedup	GPU Speedup
CPU only App1	-	1.0	1.0
CPU only App2	-	1.0	1.0
CPU only App3	-	1.0	1.0
Quantum Espresso	1	1.0	15.12
MILC	256	1.0	10.16
Starlord	16	1.0	2.39
DeepCam	1	1.0	30.70
GTC-P	1	1.0	8.05

B. *Optimizing application placement to improve SSI*

In this section we consider the effect of specialization on the heterogeneous SSI score. The mapping of applications to partitions is specified using the $f_{i,p}$ variable in Equation 7. This becomes a constrained optimization problem in which $f_{i,p}$ can be varied in order to maximize SSI.

Figure 3 compares SSI scores with and without specialization on a system with GPU node cost = 8x CPU node cost. The data points marked with "Default" label are obtained using $f_{i,p}$=0.125 for all applications and partitions. The results match those in Figure 2a. The data points marked with "Specialized" label consider a sensible initial set of values for $f_{i,p}$ in which the 3 CPU applications are run on the CPU partition only ($f_{i,p}$=0.333) and the 5 GPU applications are run on the GPU partition only ($f_{i,p}$=0.2). Finally, the data points marked with "Specialized-optimized" are optimal values of $f_{i,p}$ for each machine configuration. We obtain the values using the Sequential Least Squares Programming (SLSQP) solver in SciPy's optimize.minimize function.

The results show that a naïve specialized mapping performs worse than a default mapping when less than 30% of the budget is spent on GPUs. This is because the utilization factors $U_{i,p}$ for the GPU applications are relatively small on the GPU partition given the small size of the GPU partition and are zero on the CPU partition. In contrast, an optimized specialized mapping improves upon the default mapping even when less than 30% of the budget is spent on GPUs. We find that at 5% of the budget, an optimal mapping involves running DeepCam on the GPU partition and all other applications on the CPU partition. The dramatic speedup of this machine

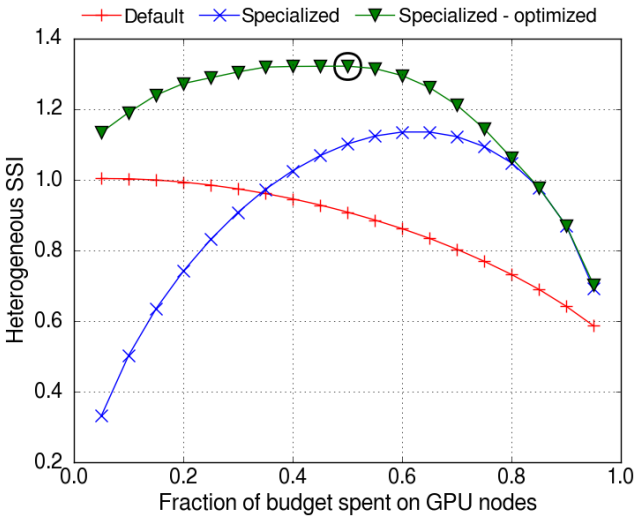


Fig. 3. The impact of specialization on the heterogeneous SSI metric for a system in which a GPU node costs 8x a CPU node. Default indicates that each application on each partition is assigned $f_{i,p}$=0.125. Specialized indicates that the CPU applications are assigned $f_{i,p}$=0.333 on the CPU partition and the GPU applications are assigned $f_{i,p}$=0.2 on the GPU partition. Finally, Specialized-optimized indicates the combination of $f_{i,p}$ leading to the highest SSI score. The circle on the Specialized-optimized iso cost curve indicates the system configuration giving the highest SSI score. This is a system with 50% of the budget spent on GPUs.

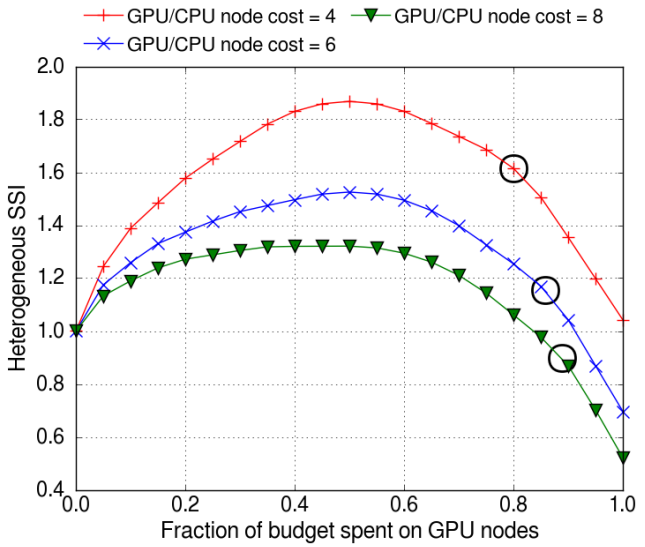


Fig. 4. The impact of specialization on the heterogeneous SSI score for a system in which a GPU node costs 4x, 6x and 8x a CPU node. The SSI score is obtained by using the optimal choice of $f_{i,p}$ for each machine configuration. The circle on each iso cost curve represents a system with 50% CPU nodes and 50% GPU nodes.

learning application on GPUs is enough to overcome the relatively small utilization factor on the small GPU partition. The optimized specialized mapping continues to change as more of the budget is spent on GPUs. In particular, we find that the solver places more of the GPU applications on the GPU partition in the order of highest GPU speedup first. The highest heterogeneous SSI score is obtained when 50% of the budget is spent on GPUs. At this machine configuration, the solver assigns factors of 0.25 for all CPU applications and Starlord on the CPU partition and factors of 0.25 for the remaining GPU applications on the GPU partition. This shows that there can be an overall throughput win for the aggregate workload by leaving applications with relatively small GPU speedups on the CPU partition. The naïve specialized mapping is the same as the optimal specialized mapping when 85% or more of the budget is spent on the GPU partition.

Figure 4 shows the heterogeneous SSI for different machine configurations when considering specialization. The earlier results in Figure 2a showed that the GPU nodes only delivered a significant benefit to the representative workload when a GPU node cost 4x a CPU node. The results in Figure 4 show that an optimized specialized mapping can improve the overall score even when a GPU node costs much more than 4x a CPU node. The results also show that devoting the entire budget to GPU nodes leads to a poor heterogeneous SSI score.

VI. Discussion

In an era of extreme-heterogeneity, the potential tension between capacity- and capability computing may become more pronounced. Although heterogeneity and specialization can clearly improve the throughput of a capacity workload (evident in the maxima depicted in Figures 2-4), as resources are divided across multiple partitions, the individual partitions become smaller and running a single capability job that uses all partitions effectively becomes more difficult.

The SSI metric described in this paper can be used to understand and quantify the impact of prioritizing one of these goals over the other. Consider the red curve in Figure 4, which corresponds to a 4x price difference between CPU-only and GPU-accelerated nodes. The highest throughput (i.e. capacity) occurs when 50% of the budget is spent on GPU nodes. This system has 5000 CPU nodes and 1250 GPU nodes. There is a capability trade-off because alternative iso cost configurations with either 10000 CPU-only nodes or 2500 GPU nodes would permit 2x larger CPU-only or GPU jobs.

There are also circumstances when heterogeneity is not beneficial. This could occur when the entire workload benefits from the same accelerator; if the benchmark suite consisted of only DeepCam and GTC-P, then an all-GPU system would be most cost effective. At the opposite end of the spectrum, it is not feasible to field specialized accelerators for each element of a highly diverse or rapidly evolving workload.

Workflow requirements also complicate the cost/benefit analysis of heterogeneous computing; if different stages of the same computational pipeline have substantially different performance characteristics, would their combined workflow be better served by general purpose processors or a mix of specialized accelerators (i.e. extreme heterogeneity)? Given the vast range of software diversity, particularly when workflows require tight coupling between accelerator types, it seems

nearly certain that HPC centers will continue to provide large deployments of general purpose processors for many years to come.

VII. Conclusion

Historically, the majority of large-scale supercomputers have had a homogeneous compute node and users of that machine only had to worry about porting to a single architecture. But there is currently much research and development of domain specific, or even application specific, integrated circuits that may be very attractive to deploy to accelerate key applications. In order to take advantage of these developments, more and more supercomputer centers are going to be deploying machines with more than a single node type, and in some cases this will most likely be more than two node types. NERSC has done this with its the current Cori supercomputer and plans at least two node types with the future NERSC-9, which is scheduled for delivery in 2020.

In this paper we have developed a formulation and methodology for quantifying the performance of a heterogeneous supercomputer using a single metric, heterogeneous SSI. We have applied heterogeneous SSI using data collected on current generation Intel Xeon® and NVIDIA GPU based node types and demonstrated how it can be used to help gauge how one would allocate their budget based on the accelerated application performance on the GPU nodes. The first example assumes that all applications can run on all platforms. We then demonstrated that node specialization and optimization of application placement on the different partitions can be used to significantly increase overall workload throughput, and in some scenarios the combination of accelerator nodes and optimized placement can nearly double the overall throughput, illustrating the importance of careful evaluation of workloads and their affinity for various accelerators.

Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. We would also like to acknowledge the Swiss National Supercomputing Center for their support and access to the Piz Daint supercomputer and to NVIDIA Corporation for access to their PSG cluster.

References

[1] "The Extreme Heterogeneity Virtual Workshop," Accessed August 8, 2018. [Online]. Available: {https://www.top500.org}

[2] W. Kramer, J. Shalf, and E. Strohmaier, "The NERSC Sustained System Performance (SSP) Metric," LBNL, Tech. Rep. 2005-09-18, Sep. 2005, https://escholarship.org/uc/item/4f5621q9; accessed 17 August 2018.

[3] "The TOP500 List," Accessed August 8, 2018. [Online]. Available: {https://www.orau.gov/ExHeterogeneity2018/}

[4] D. W. Doerfler, M. Rajan, C. Nuss, C. Wright, and T. Spelce, "Application-Driven Accpetance of Cielo an XE6 Petascale Capability Platform," in *CUG 2011: Cray Users Group*, May 2011.

[5] P. J. Fleming and J. J. Wallace, "How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results," *Commun. ACM*, vol. 29, no. 3, pp. 218–221, Mar. 1986. [Online]. Available: http://doi.acm.org/10.1145/5666.5673

[6] J. E. Smith, "Characterizing Computer Performance with a Single Number," *Commun. ACM*, vol. 31, no. 10, pp. 1202–1206, Oct. 1988. [Online]. Available: http://doi.acm.org/10.1145/63039.63043

[7] NERSC, "2014 NERSC Workload Analysis," Accessed August 8, 2018. [Online]. Available: {http://portal.nersc.gov/project/mpccc/baustin/NERSC_2014_Workload_Analysis_v1.1.pdf}

[8] B. Bauer, S. Gottlieb, and T. Hoefler, "Performance modeling and comparative analysis of the MILC Lattice QCD application su3_rmd," in *Proc. CCGRID2012: IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2012.

[9] S. Gottlieb and S. Tamhankar, "Benchmarking MILC with OpenMP and MPI," *Nucl.Phys.Proc.Suppl.*, vol. 94, pp. 841–845, 2001.

[10] "The QUDA library for lattice QCD," Accessed August 8, 2018. [Online]. Available: {https://github.com/lattice/quda}

[11] "QCD for Intel Xeon Phi and Xeon processors," Accessed August 8, 2018. [Online]. Available: {https://github.com/JeffersonLab/qphix}

[12] StarLord: A Mini-App Version of Castro. https://github.com/AMReX-Astro/StarLord.

[13] A. S. Almgren, V. E. Beckner, J. B. Bell, M. S. Day, L. H. Howell, C. C. Joggerst, M. J. Lijewski, A. Nonaka, M. Singer, and M. Zingale, "CASTRO: A New Compressible Astrophysical Solver. I. Hydrodynamics and Self-gravity," *Astrophysical Journal*, vol. 715, pp. 1221–1238, Jun. 2010.

[14] W. Zhang, L. Howell, A. Almgren, A. Burrows, and J. Bell, "CASTRO: A New Compressible Astrophysical Solver. II. Gray Radiation Hydrodynamics," *Astrophysical Journal Supplement Series*, vol. 196, p. 20, Oct. 2011.

[15] W. Zhang, L. Howell, A. Almgren, A. Burrows, J. Dolence, and J. Bell, "CASTRO: A New Compressible Astrophysical Solver. III. Multigroup Radiation Hydrodynamics," *Astrophysical Journal Supplement Series*, vol. 204, p. 7, Jan. 2013.

[16] Castro: An Adaptive Mesh, Astrophysical Radiation Hydrodynamics Simulation Code. https://github.com/AMReX-Astro/StarLord.

[17] AMReX: Co-Design Center for Block-Structured AMR. https://www.exascaleproject.org/project/amrex-co-design-center-block-structured-amr/.

[18] TensorFlow website. [Online]. Available: https://tensorflow.org

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[20] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] cuDNN website. [Online]. Available: https://developer.nvidia.com/cudnn

[23] "Introducing DNN primitives in Intel® Math Kernel Library," https://software.intel.com/en-us/articles/introducing-dnn-primitives-in-intelr-mkl, 2017.

[24] Gyrokinetic Toroidal Code at Princeton (GTC-P). https://extremescaleglobalpic.princeton.edu/gtcp.

[25] B. Wang, S. Ethier, W. M. Tang, K. Z. Ibrahim, K. Madduri, S. Williams, and L. Oliker, "Modern Gyrokinetic Particle-In-Cell Simulation of Fusion Plasmas on Top Supercomputers," *CoRR*, vol. abs/1510.05546, 2015. [Online]. Available: http://arxiv.org/abs/1510.05546

[26] Particle Turbulence Simulations for Sustainable Fusion Reactions in ITER. https://www.olcf.ornl.gov/caar/gtc.

[27] D. Miles, "Accelerating HPC Applications on Nvidia GPUs with OpenACC," Presentation at Stanford HPC Advisory Council Conference and Workshop. [Online]. Available: http://www.hpcadvisorycouncil.com/events/2018/stanford-workshop/pdf/DayTwo_Wed21Feb2018/DMiles_PGI_OpenACC_DayTwo_21Feb2018.pdf, Feb. 2018.

APPENDIX A
ARTIFACT DESCRIPTION APPENDIX: [A METRIC FOR EVALUATING SUPERCOMPUTER PERFORMANCE IN THE ERA OF EXTREME HETEROGENEITY]

A. Abstract

The key contribution of this paper is the definition of a metric for measuring performance of a system composed of a heterogeneous mix of node types. All of the equations are provided in the body of the paper and the input data for the sample calculations are in Table IV. The Python script used to optimize the $f_{i,p}$ coefficients of the hetero-SSI formula and to generate the CPU:GPU cost tradeoff plots in Section V is publicly available as described below.

B. Description

1) Check-list (artifact meta information):

- **Program: optimize_ssi.py**
- **Run-time environment: Python**
- **Execution: python optimize_ssi.py**
- **Output: Graph files for Figures 2, 3 and 4**
- **Publicly available?: Yes**

2) How software can be obtained: The script can be downloaded from http://portal.nersc.gov/project/m888/reproducibility/optimize_ssi.py.

3) Software dependencies: We used Python version 2.7.13. The script depends on the following Python packages which may need to be installed separately (the versions we used are shown in parenthesis):

- NumPy (1.8.0)
- SciPy (0.16.0)
- Matplotlib (1.4.3)

4) Datasets: The speedup data from Table IV are embedded in the source code. Alternative data can be used by modifying lines 10 through 14.

C. Evaluation and expected result

The script will produce the following files:

- Figure 2a: hetero-ssi-cost.png
- Figure 2b: hetero-ssi-mean.png
- Figure 3: opt_v_noopt.png
- Figure 4: optimal_ssi.png