INTERNATIONAL ROADMAP FOR DEVICES AND SYSTEMS

INTERNATIONAL ROADMAP FOR DEVICES AND SYSTEMS

2020 EDITION

APPLICATION BENCHMARKING

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

Thanks goes to the members of the team, who are:

Name	Representing
Geoffrey Burr	IBM
Tom Conte [chair]	Georgia Institute of Technology, USA
Paolo Gargini	Chair, IRDS
Vladimir Getov	University of Westminster, UK
Yoshihiro Hayashi	Keio University, JAPAN [SDRJ representative]
Takeshi Iwashita*	Hokkaido University, JAPAN [SDRJ representative]
Vijay Janapa Reddi*	Harvard University, USA
Masaaki Kondo*	University of Tokyo, JAPAN [SDRJ representative]
Tushar Krishna*	Georgia Institute of Technology, USA
Peter M. Kogge	University of Notre Dame, USA
Anoop Niar*	Flextronics, Inc.
Dam Sunwoo	Arm Research
Josep Torrellas	University of Illinois at Urbana-Champaign, USA
Peter Torelli	Chair, EEMBC

(*new members for 2020 edition)

APPLICATION BENCHMARKING

1. INTRODUCTION

The mission of the Applications Benchmarking[1] (AB) International Focus Team (IFT) is to identify key application drivers, and to track and roadmap the performance of these applications for the next 15 years. Given a list of market drivers from the Systems and Architectures International Focus Team (SA IFT), AB generates a cross matrix map showing which application(s) are important or critical (gating) for each market.

Historically, applications drive much of the nanoelectronics industry. For example, 15 years ago the PC industry put pressure on semiconductor manufacturers to advance to the next node in the roadmap. Today, as applications shift to the mobile market, it is again manufacturers that are applying pressure for new technology. The market for *Internet of Things edge devices* (IoT-e) has its own set of pressures and needs, including low cost and low energy consumption. Most of this is discussed in the Systems and Architectures (SA) chapter elsewhere in this roadmap. It is the function of this chapter to step back from the current markets and their needs, and to consider the current and near-future application needs in each of these markets. For this reason, AB was created as part of the International Roadmap for Devices and Systems (IRDS).

The application areas that the AB IFT tracks are summarized below in Table AB-1.

Table AB-1 Application Areas

Application Area	*Description*
Big Data Analytics	Data mining to identify nodes in a large graph that satisfy a given feature/features
Artificial Intelligence	Modern artificial intelligence applications with emphasis on machine learning approaches. Graphical dynamic moving image (movie) recognition of a class of targets (e.g., face, car). This can include neuromorphic / deep learning approaches such as DNNs.
Discrete Event Simulation	Large discrete event simulation of a discretized-time system. (e.g., large computer system simulation) Generally used to model engineered systems. Computation is integer-based.
Physical System Simulation	Simulation of physical real-world phenomena. Typically, finite-element based. Examples include fluid flow, weather prediction, thermo-evolution, and medical and drug discovery. Computation is floating-point-based.
Optimization	Integer NP-hard optimization problems, often solved with near-optimal approximation techniques.
Graphics/VR/AR	Large scale, real-time photorealistic rendering driven by physical world models. Examples include interactive gaming, Augmented Reality, Virtual Reality.
Cryptographic codec	Cryptographic encoding and decoding, including specialized hardware acceleration

DNN—deep neural network, NP—the non-polynomial complexity class of algorithms, VR—virtual reality, AR—augmented reality

In order to track these areas, the AB IFT relies upon existing standard benchmarks where available. These benchmarks should fulfill two criteria:

1. **Benchmark Availability**: There are several benchmark sets available that cover each application area. However, many of these benchmarks either cover only a portion of an application area or cover more than one application area.

[1] *Note that in the computer industry, as opposed to the larger semiconductor industry, "benchmarking" refers to using test programs that serve as proxies for user applications in order to estimate the performance of a computer system on a given application domain.*

2. **Benchmark Results Availability**: In order for benchmarks to be useful for projecting a trend in performance vs. time, there must be a sufficiently-long history of benchmark scores. At a minimum, AB IFT believes at least 4 years prior to the current day of scores should be available.

Some of the application areas studied also have the following requirements:

3. **Metrics vs. Precision and Accuracy**: For some application areas (e.g., feature recognition, optimization), the precision of the result is a parameter for the benchmark performance (i.e., recognizing 81% of faces vs. 85% of faces). Related to this is the accuracy of the result is often also parameterized (i.e., finding a near optimal result that is within 5% of the optimal result).
4. **Power/Energy Scoring**: With a few exceptions, the majority of available benchmarks track metrics that are, unfortunately, disconnected from power dissipation and total energy consumption.

We address these challenges in each of the sections below. For the Cryptographic Codec area, criteria (2) is missing (sufficient benchmark results available), thus we have included the analysis but without the quantitative data in the section below.

1.1. Current State of Technology

Application areas of interest for the IRDS are outlined in Table AB-1. These application areas were identified through dialogue between the members of both the SA IFT and AB IFT. Big data analytics is a focus of search giants. It is also vital to the business model of social network companies. Feature recognition is critical for human-computer interaction and for analysis of non-uniform, irregular data (such as used by the military and by search companies). Discrete event simulation is critical for industrial and systems engineering, computer systems engineering, telecommunications, transportation, etc. In contrast, physical system simulation is critical to product design in the automobile and aerospace industries. It is also of importance to the U.S. Department of Energy National Nuclear Security Administration. Optimization is used across engineering disciplines, for example in electronic design automation (EDA). It is also used in consumer products such as global positioning system (GPS) navigation systems. Graphics and media processing has a large impact on consumer electronics (e.g., gaming consoles), but also telecommunications and storage industries. It is also critical for search companies.

1.2. Drivers and Technology Targets

Overall Roadmap Technology Characteristics drivers for Systems (applicable systems or applications attributes) and Support Technologies (applicable device attributes) is summarized below in Section 3 and Table AB-2.

1.3. Vision of Future Technology

All of the application areas of Table AB-1 are expected to remain important over the coming 15 years. At this time, the AB IFT does not envision new application areas that are disjoint from or not covered by the application areas in Table AB-1. That said, suggestions from readers of this chapter for improvements are always welcome.

As outlined below in Section 3, most of the application areas can benefit from enhanced memory bandwidth. In some cases, benefits will also come from reduced memory access latency. The difference between these is whether the application critically requires data to compute future data (both bandwidth and access latency critical) or not (memory bandwidth critical). A second phenomenon is the use of application-specific accelerators to improve the performance of the application areas, while minimizing overall thermal power dissipation and reducing total energy consumption.

2. Scope of Report

The scope of this report is to characterize the Application Areas, predict their future performance, and to identify technology needs and architectural features. The period of projection is into the next 10 to 15 years. Since this projection is based on the overall evolution of published benchmarks with a long history, this projection takes into account both hardware and software evolution.

3. Summary and Key Points

A summary of the application area analysis is presented in Table AB-2. An entry in this table indicates that the application area is sensitive to the improvement paths shown in the columns. These improvement paths are:

Algorithmic improvement: In this situation, the AB team viewed the algorithm development as an active area that will result in a significant fraction of the performance growth over time.

Memory bandwidth and Memory latency: These two improvement areas are inter-related. There are trends today for improving memory bandwidth onto and off the processors. In addition, memory latency becomes important when traversal is a function of data (e.g., pointer chasing workloads). In general, when the processing being performed is very parallel, memory bandwidth dominates. When it is less parallel (e.g., discrete event simulation or sparse matrix calculations, with the latter included in physical system simulation), then latency is the more important memory attribute.

Network bandwidth: this improvement area relates to the interconnection between processing nodes ("cores"), and the interconnection between processing and memory. This network includes on-chip and off-chip networks.

Fixed-function acceleration: some workloads have highly-repetitive kernels that can be reduced to a specialized, fixed-function data path. For some application areas, this improvement area is expected to give significant benefits in the near future.

Table AB-2 Critical Technology Needs for Application Areas

	Improvement Paths				
Improvement Paths Application Area	**Algorithmic improvement**	**Memory bandwidth**	**Memory latency**	**Network bandwidth**	**Fixed-function acceleration**
Big Data Analytics		X	X	X	X
Artificial Intelligence	X	X			X
Discrete Event Simulation		X	X		
Physical System Simulation	X	X	X	X	
Optimization	X	X	X		X
Graphics/VR/AR	X	X			X
Cryptographic codec		X			X

4. Application Analysis

What follows is a detailed analysis of the six application areas presented in Table AB-1.

4.1. Big Data Analytics

This workload class involves a variety of algorithms that operate on a large graph to identify a certain property. The metric that is relevant is computational capability (speed). Ideally, speed would be measured as time to reach the solution of the problem. However, the Graph 500 initiative[1] accepts as a speed metric the traversed edges per second (TEPS) for the problems it considers. A second metric is the energy efficiency in doing these computations, measured in TEPS/watt. This second metric is the focus of the Green Graph 500 initiative[2].

While this domain is very popular, there are a large variety of frameworks and machine sizes. Some frameworks use a shared-memory model and hence run on relatively modest sized servers[3], while others use a distributed-memory model and target very large clusters[4]. Among the frameworks using the shared-memory model, there is no available cross-comparison. But, it is becoming clear that current benchmarks for these frameworks are not benchmarking the whole range of operations that appear in graph workloads. Specifically, current benchmarks do not appropriately cover operations on property-rich graph vertices/edges, and on time-changing graphs. We expect that such benchmarks will change with time. On the other hand, among the efforts that target large, distributed-memory machines, there is an agreed set of benchmarks, namely those in the Graph 500 website. Moreover, the Graph 500 competition has been going on since 2010, and hence there is historical data. For these reasons, we use the Graph 500 benchmark data to track the trends in big data analytics workload performance.

The Green Graph 500 initiative gives a different perspective, as it considers the power consumed in the computation. However, the machines that form the top of this list are small machines (32–60 cores). We think they are less representative of this type of workloads. For this reason, we do not consider the Green Graph 500 list.

Up until recently, the Graph 500 benchmark contained two kernels. However, in June 2017, V2.0 added a third one. Since we do not have historical data on the third kernel, we do not consider it. The first kernel constructs a weighted, undirected graph from the given input tuple list, and the second kernel operates on the graph. The first kernel constructs the graph in a format usable by the subsequent kernel. No subsequent modifications are permitted to benefit specific kernels. The second kernel performs a breadth-first search of the graph. Both kernels are timed.

Kernel 1: Graph Construction. The first kernel transforms an edge list to any data structures (held in internal or external memory) that are used for the next kernel. Each edge in the list is a tuple that contains an edge's endpoint vertex identifiers and its weight. Various internal memory representations are allowed, including, but not limited to, sparse matrices and multi-level linked lists.

Kernel 2: Breadth-first Search (BFS). A BFS of a graph starts with a single source vertex. Then, in phases, it finds and labels its neighbors, then the neighbors of its neighbors, and so on. This is a fundamental method on which many graph algorithms are based. The benchmark does not constrain the choice of BFS algorithm itself, as long as it produces a correct BFS tree as output. This benchmark's memory access pattern is data-dependent, with likely a small average prefetch depth. This benchmark measures the ability of the architecture to deliver high throughput while executing many concurrent threads, each with low memory-level parallelism and with a high density of memory accesses. It also measures resilience to hot-spotting when many of the memory references are to the same location. In addition, it measures efficiency when every thread's execution path depends on the asynchronous side-effects of others. Finally, it measures the ability to dynamically load-balance unpredictably-sized work units. The benchmark performs 64 BFS searches from different source vertices executed in sequence. Kernel problem classes are shown in Table AB-3.

Table AB-3 Kernel Problem Classes (the benchmark gives different problem classes, based on the approximate size of the graph. The classes are shown below)

Problem Class	*Size*
Toy (level 10)	17 GB or around 10^{10} bytes
Mini (level 11)	140 GB or around10^{11} bytes
Small (level 12)	1 TB or around10^{12} bytes
Medium (level 13)	17 TB or around 10^{13} bytes
Large (level 14)	140 TB or around 10^{14} bytes
Huge (level 15)	1.1 PB or around10^{15} bytes

4.1.1. PERFORMANCE TRENDS AND PREDICTION

The performance is given in TEPS for Kernel 2. Let *time* be the measured execution time for kernel 2. Let *m* be the number of edges traversed by the search (with some special accounting for some classes of edges called non-self-loop edges). Then, the TEPS (number of edge traversals per second) is given by *m/time*. The output results include additional information, such as the scale of the graph, the Kernel 1 time, various quartiles for the Kernel 2 times (min_time, firstquartile_time, median_time, thirdquartile_time, max_time), and standard deviation values.

Figure AB-1 plots the number of giga-TEPS (GTEPS) of the top six machines from November 2010 (which is when the Graph 500 competition started) to November 2019 (which is the latest measurement). The data is refreshed every 6 months. Nearly all of these runs since 2013 are with the "Large" problem size.

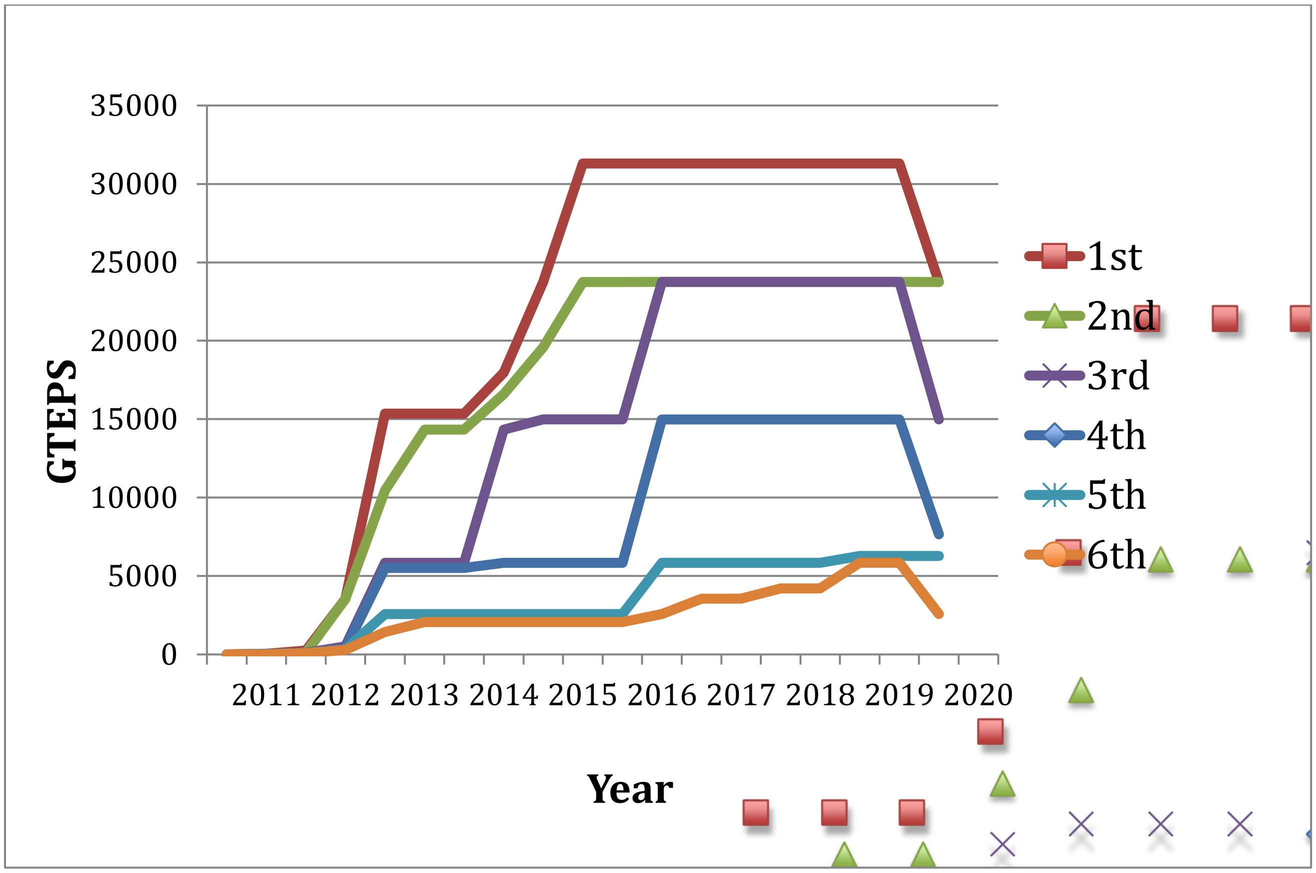


Figure AB-1 Number of GTEPS of the Top Six Machines Since the Inception of the Graph 500 Competition

We see that the GETPS rate has been steadily increasing, although it seems to have been moving in plateaus. As of June 2019, the top three were the K Computer (Japan), the Sunway TaihuLight (China), and the DOE/NNSA/LLNL Sequoia (USA). The top three machines are invariably custom-designed machines, with custom networks and, at this point, with 1.3 to1.5 Petabytes of memory. In the last sample (November 2019), the K Computer has disappeared from the ranking. As a result, the lines in the plot go down.

Figure AB-2 plots the number of cores used in the top six Graph 500 machines in the same period of time. If we look at the top three machines, we see that two of them use around 1 to 1.5 million cores, and the other uses 10 M cores. The former use more traditional HPC-class cores, while the latter uses simpler, custom cores.

Overall, we expect that both the TEPS and the number of cores in the top three machines will continue to go up in bursts in the near future. One difficulty is that the machines are quite expensive.

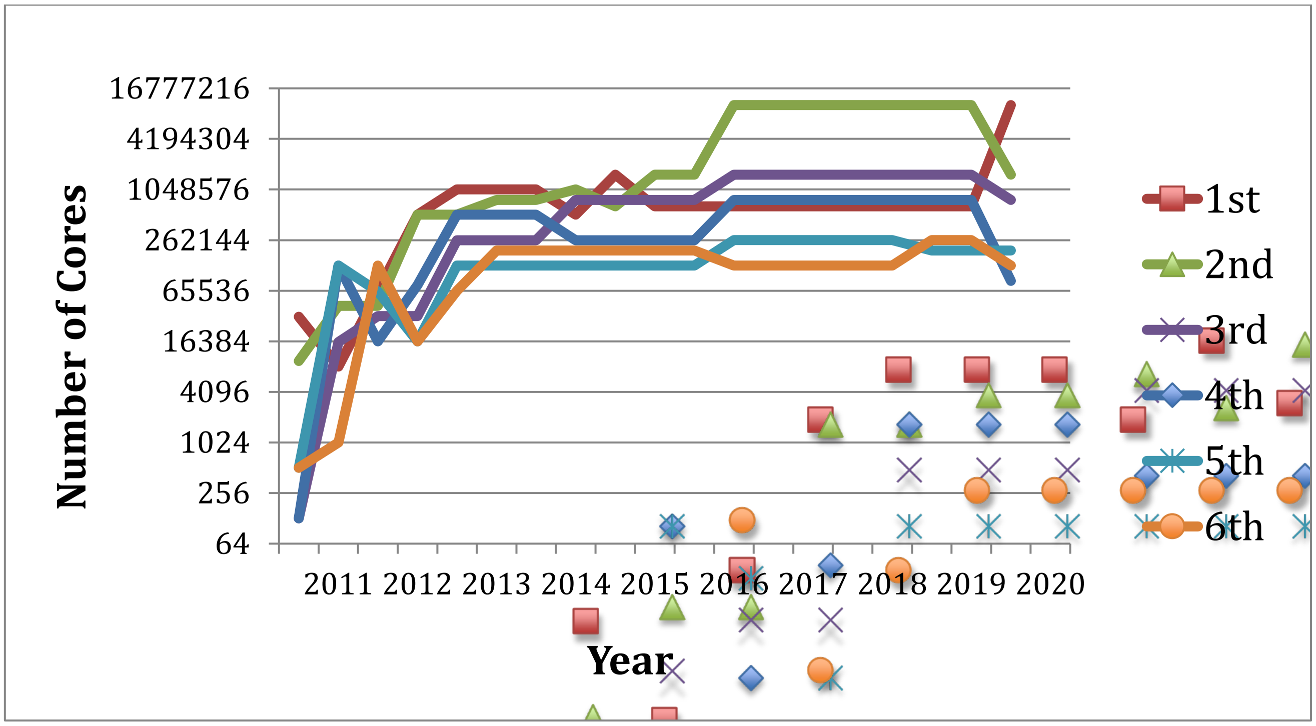


Figure AB-2 Number of Cores Used in the Top Six Graph 500 Machines Since the Inception of the Graph 500 Competition

4.1.2. Technology Needs

The gains in graph processing performance come from three main sources: improvements in algorithms, increase in memory bandwidth, and increase in processor/memory interconnection network bandwidth. It is important to note that *processor performance does not have a first-order impact.*

Some of the gains in TEPS over the years have been due to improvements in the algorithm used. While we expect this factor to continue to have an impact, the improvements are likely to provide diminishing returns.

The most critical resources at this point are the bandwidth and latency of the memory and the global network. Graph problems have a high ratio of communication to computation. Moreover, they rarely have much locality. As a result, there are frequent, non-local transfers of data between the memory and the cores, and among the cores. Such transfers follow an irregular pattern.

To increase performance, memory bandwidth needs to increase. The current top machines have an aggregate memory bandwidth of about 5 petabytes per second. We need to increase this number. We expect that the next generation machines will attain about 20 petabytes per second, thanks to in-package dynamic random access memories (DRAMs) like the high-bandwidth memory (HBM) standard. This will result in a noticeable performance improvement.

To further improve performance, global networks need to provide higher bandwidths and lower latencies. What is especially needed is a higher injection bandwidth into the network, as well as faster all-to-all communication support. The current top machines already have very expensive networks; we expect the cost of such networks to increase in the future, as they are upgraded. One possible source of improvements is to use optical networks, especially if they can provide aggregate injection rates into the network in the order of one terabit per second.

4.1.3. Systems and Architectures Impact

The trend we are observing is that the graph sizes continue to increase. Since graph data is typically confidential, we do not know the sizes of the databases used by large commercial data centers and government agencies. However, we expect the databases used to use hundreds of terabytes or petabytes soon. Moreover, these graphs are dynamic, meaning that they continuously change in time. To process these graphs, we need very large clusters.

In terms of algorithms, we expect the algorithms to keep changing. Programmers will attempt to exploit locality more, and to reduce the communication as much as possible, possibly through redundant recomputation. We also expect the benchmarks to change to include benchmarks that modify the graph on the fly.

In these clusters, the cores can be out-of-order commodity cores, with modest floating-point support. Since there is a large frequency of memory accesses that miss in the caches, these cores do not need to be very wide-issue. They do not need to be equipped with substantial floating-point hardware. They are frequently stalled due to reorder buffer or load/store queues being full. One interesting issue is whether graphics processing units (GPUs) can be used, as they become more tolerant of computation divergence. However, it may be necessary to change the graph algorithms, typically reducing the efficiency of the algorithms.

Memory bandwidth needs to be high. In the next few years, we expect to see an aggregate memory bandwidth of about 10 petabytes per second, thanks to the arrival of stacked in-package memory such as HBM. This will deliver a good performance boost, at the expense of increasing the cost of the machine. Note that memory needs are large in a machine with in-DRAM graphs—such problems need hundreds of terabytes and even petabytes. One interesting question is what can be accomplished with the arrival of relatively fast non-volatile memory (NVM). With NVM, the amount of memory available will increase substantially. However, it may require re-writing the algorithms.

The highest gains will likely be obtained by improving the global network bandwidth and latency. We expect to see more custom-designed networks and possibly optics-based networks, hopefully soon, delivering an aggregate bandwidth into the network of around one terabit per second. These networks need to provide fast all-to-all communication, so that cores can exchange boundary information quickly. They need to support irregular communication patterns. Some form of network topology that handles non-local traffic effectively will work best.

4.2. Artificial Intelligence

The application area of Artificial Intelligence (AI), including the specific domain of Machine Learning (ML), refers to a class of techniques that learn from data, in order to enable applications that can make meaningful decisions—such as classifications, predictions or recommendations. As such, these techniques do not need to be explicitly programmed. Over the last few years, advancements in the family of AI/ML techniques known as Deep Neural Networks (DNNs) have demonstrated superhuman accuracies at tasks such as computer vision, speech recognition and language translation.

There were three important ingredients behind the success of this paradigm. First was the ready availability of vast datasets of labelled examples for training. Second was the inherent but non-obvious scalability of DNNs, which are based on techniques known since the mid-1980s—supervised training using backpropagation and stochastic gradient descent. What we know now is that when these types of networks are made larger, with more neurons and more weights, and are trained with more data, they almost always perform better. Thus, one of the key ingredients to the recent success of AI/ML have been powerful computational platforms. Hardware that was already readily available, such as CPUs and GPUs, were critical for empirically proving that the DNN approach was in fact scalable, and for the first applications. But now we seek to extend upon the inherent scalability of these algorithms by expressly designing hardware for AI/ML—using a mix of CPU, GPU, FPGA, custom accelerator ASICs, and perhaps even more exotic hardware approaches—in order to address a growing set of commercially relevant applications.

The focus of this section is on Application Benchmarking of the AI/ML area. Our goal is to track progress in this field, to enable continued improvement of existing computational approaches (e.g., CPUs and GPUs), and to provide accurate performance targets for both evolutionary and revolutionary computing approaches for this application area. We attempt to accurately report the recent trend-lines in the performance metrics of interest—throughput, latency, accuracy, and energy efficiency—so that we can attempt to extrapolate the future evolution of these trends, at least into the near-future. Figure AB-3 shows the various choices that must be made when implementing an ML application, and thus illustrates the significant difficulty of quantifying improvements enabled by hardware, given the presence of so many other factors that can (and do) influence ML performance metrics.

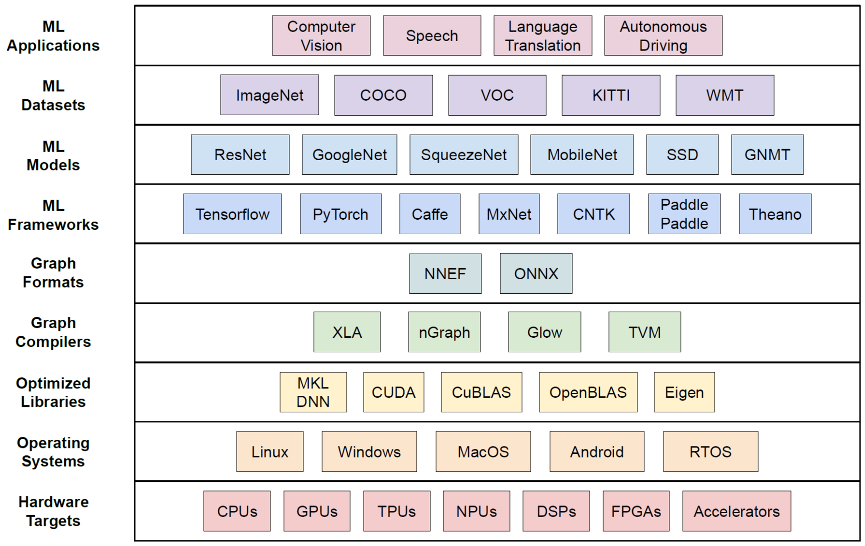


Figure AB-3 The variety of software and hardware options at various levels greatly complicate the benchmarking of AI/ML systems[5]

4.2.1. APPLICATION DOMAINS

While AI/ML algorithms have been around for many years, the specific class of Deep Neural Networks have found commercial application on a wide variety of workloads in the past decade, including image classification, object detection, speech recognition, machine translation, recommendation systems, sentiment analysis/classification of text, language modeling, text-to-speech, face identification, image segmentation, and image enhancement[5]. In the business world, AI/ML is used for process automation, cognitive insight, fraud detection, cognitive engagement, and other applications. In consumers' interactions with devices like smartphones, and with large internet companies like Google, Facebook, YouTube, Netflix, Uber, Amazon and the like, it takes less time to list the functionality that has *no* dependence on AI/ML than it does to describe the functionality that now depends on AI/ML to some degree.

4.2.2. AI/ML DEPLOYMENT PLATFORMS

AI/ML applications are deployed on a wide variety of platforms, based on target application needs. The computational needs of these platforms vary based on application needs and platform form-factor. To understand the tradeoffs involved in choosing a specific AI/ML deployment mode, it is important to understand the different workload characteristics and their constraints.

Here we focus on the supervised learning behind Deep Neural Networks, for which AI/ML techniques have two different distinct phases – training and inference.

- Training is the process whereby labelled data is fed into the system and a trained model of a particular accuracy is built. The training of a model is not an absolute process, but can spit out models of varying accuracy, depending on how good the model is, on how well the data is chosen, and on careful tuning of various hyper-parameters. In general, more training data and larger models lead to higher accuracies, although at the cost of increasing the computation and memory requirements. As models become extremely large, training is more and more a distributed task, thus involving networking between multiple nodes all working together on the same overall task or set of tasks.
- Inference is the process whereby the trained model is deployed, actual input data is fed to it, and inferences are made.

Training and Inference need not run on the same hardware, and these compute tasks can be partitioned across multiple hardware platforms based on the target application constraints. The target application constraints are typically latency, power, cost, inference speed, training speed, and accuracy (or similar metric for decision quality).

For self-driving cars, latency is absolutely critical for the car to react to a pedestrian or a traffic signal to avoid accidents and so inference runs on an edge computer in the car. For a photo tagging application on a smartphone, there are no latency constraints, so cloud servers could be used for the inference. For a low-cost gas sensor that can sense dangerous gases using AI/ML methods, one can use inexpensive microcontrollers for running inference.

With this basic taxonomy, we describe the three basic deployment models based on the constraints shown in table below: "High Performance" AI/ML performed in the cloud, "Real Time Inference" as performed at the Edge, and "Resource-constrained Inference." This last category is sometimes referred to as "Endpoint Inference" or "TinyML." It is also performed at the Edge, but by Micro-controller Units (MCU) or in Internet-Of-Things (IOT) sensors where resources such as cost, or the volume-, area-, or memory-footprint, or the amount of battery power/energy available may be severely constrained.

Training has high computational needs and so mostly runs on the cloud. The aforementioned strong correlation between the size of the training set and the model performance, and the difficulty of avoiding "catastrophic forgetting" when re-training a model with only some of the original training set[6], imply that there are significant challenges to an edge-based training approach. However, there is some research in running training on edge computers for data privacy reasons, with data scalability supported by using distributed or federated training methods that can exchange weight-update data and enjoy the benefits of network scale, without sacrificing data privacy or other data-locality requirements.

Table AB-4 AI/ML Deployment Platforms and Constraints

Deployment Platforms	***High Performance (Cloud)***	***Real Time Inference (IoT Edge)***	***Resource-constrained Inference (IoT Edge MCU/IOT)***
Training System	*Cloud*	*Cloud*	*Cloud*
Inference System	*Cloud*	*IoT Edge Computer*	*IoT Edge Microcontroller*
Training Speed	High	High	High
Inference Throughput (pages/second)	High	Medium	Low
Inference Latency (milliseconds)	High	Low	Low
Compute	High	Medium	Low
Cost	High	Medium	Low
Power	High	Medium	Low

4.2.3. Computing Architectures

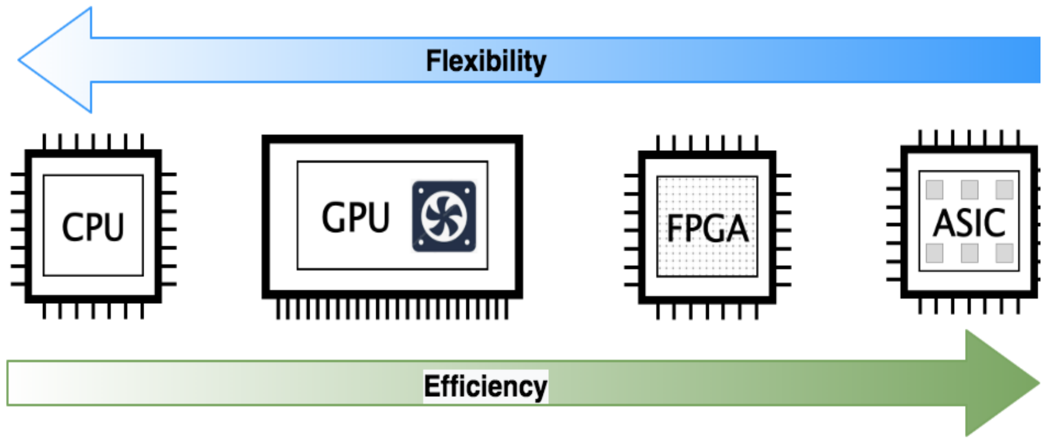


Figure AB-4 General tradeoffs among Computing Architectures

The AI/ML-specific portion of the market for computing hardware is growing rapidly, and is expected to account for almost 20 percent of all computing demand by 2025. This would translate into about $67 billion in revenue[7]. As mentioned previously, AI/ML applications run on a variety of platforms. Here we describe the hardware used across these platforms.

4.2.3.1. CPU

CPUs remain the ubiquitous choice for running AI/ML workloads. Inference is run heavily over many-cores in datacenters and on simple CPUs on smartphones or IoT devices. However, the end of performance scaling in CPUs means that getting increased performance as technology scales can no longer be expected. This becomes exceptionally challenging as DNN models are evolving rapidly.

4.2.3.2. GPU

GPUs were the hardware platform that propelled the current strong interest in AI/ML, since their parallel processing capabilities can be leveraged to accelerate matrix multiplications, which are at the heart of the computations within Deep Neural Networks. A single modern GPU can have upwards of 5,000 tiny cores capable of highly parallel operation. These systems also provide high internal memory bandwidth, which makes them ideal for the intense computation needs of AI/ML. NVIDIA's GPUs offer "Tensor Cores," tiny engines for running dense matrix multiplications efficiently at reduced precision. Training requires a high number of high-precision floating point operations with high memory bandwidth, and GPUs remain the most popular choice. For inference however, the large power envelope of most GPUs makes them less efficient for mass deployment in edge applications.

4.2.3.3. FPGA

FPGAs are a popular choice for DNN inference, as their reconfigurable substrate helps tailor the datapath for the DNN model. Moreover, FPGAs are bit-configurable, and are thus highly popular for running inference at lower precision for compute- and memory-efficiency. FPGAs can also offer field re-programmability as DNN models evolve. Today's FPGAs are less suited for training though, as they lack high-precision floating point units. The most vibrant example of FPGAs being used for AI/ML today is Microsoft's Brainwave project[8]. On the downside, the overall energy-efficiency and operating frequency of FPGAs is still lower than ASIC counterparts.

4.2.3.4. ASIC Accelerators

Digital ASIC Accelerators

The inefficiencies in CPUs, GPUs and FPGAs described above has led to a surge in specialized Digital ASIC-based accelerators for AI/ML[9]. From a high-level view, all digital DNN accelerators are essentially large arrays of Multiply-Accumulate (MAC) units fed via custom scratchpads, along with special functional units for pooling, activation, and so on.

There are two key aspects that can be used to classify the various prototype (commercial and academic) digital accelerators that have been demonstrated: ***microarchitecture*** and ***dataflow***.

***Microarchitecture*—**At the microarchitecture level, we can classify these digital ASIC Accelerators into two categories, depending on the control paradigm of the MAC units: systolic array and SIMD/MIMD arrays.

Systolic arrays are built as a grid of MAC units connected as a pipeline in two dimensions. Data (inputs/weights/partial-sums) move every cycle in a horizontal, vertical (or both) direction, depending on the dataflow strategy. The entire array operates as a monolithic matrix-matrix multiplication core with a centralized controller. Individual MAC units cannot be stalled.

In contrast, SIMD/MIMD arrays are built using a collection of processing elements (PEs), with each containing a MAC unit and some local storage. These arrays can be operated as full MIMD units (i.e., every PE has independent control and can stall if it is waiting for operands), or multiple SIMD units, where the PEs within each SIMD unit operate in lock-step, but different SIMD units can be controlled independently.

Both approaches appear in products today. For instance, Google's TPU[10] uses a systolic array, while NVDLA (Nvidia Deep Learning Accelerator)[11] and TESLA FSD[12] use SIMD/MIMD spatial arrays. Systolic arrays are simpler and offer higher compute density but may suffer from under-utilization if the problem dimension does not map well over the array, or has sparsity (i.e., zeros). SIMD/MIMD arrays are better for utilization and provide more mapping flexibility but come at the cost of more complexity due to distributed control[13].

***Dataflow*—**In this digital ASIC accelerator domain, dataflow collectively refers to the data iteration order and the partitioning strategy across the MAC units.

The choice of dataflow determines overall data reuse, which in turn affects the amount of data movement in/out of the array. Data reuse can be augmented in two ways: by reuse of weights and input excitations across the multiple data examples within a minibatch that is processed together, and by using networks with Convolutional Layers, in which the small number of weights in a convolution kernel are re-used across many input excitations, as the convolution kernel passes over an input or intermediate image. Weight reuse is particularly important, since it amortizes the large energy cost of retrieving data from memory (particularly from off-chip memory) over a much larger number of computations that reuse that data.

In terms of iteration order, accelerators can follow different strategies, including weight/input stationary (where the weight/input tensor is kept stationary and the input/weight tensor is streamed in) or output stationary (where the weight and input are both streamed in and the output is computed in-place). Google's TPU uses the weight-stationary approach[10].

For the partitioning strategy, different dimensions of the input/weight tensors can be partitioned across the MAC units for parallelism. For example, NVIDIA's NVDLA partitions the input channels along one dimension, and output channels along the other. For each dataflow, compilers can come up with appropriate tile sizes depending on the amount of on-chip buffering.

Some of these architecture and dataflow concepts being explored in the context of digital ASIC accelerators can be exploited in other contexts, particularly when programming FPGAs for these AI/ML applications. In addition, newer FPGA designs are starting to merge digital ASIC accelerator concepts within conventional FPGA configurations[14].

COMPUTE-IN-MEMORY (CIM) MACROS

Compute-in-Memory is a paradigm that achieves the parallel multiply-accumulate capabilities of the "Analog AI" techniques described below, but without requiring analog conductance states. Instead, digital memory devices such as SRAM or a compact NVM such as RRAM are used to encode multi-bit synapses in a customized memory array, which is then designed to perform some or all of the multiply-accumulate operations needed for AI/ML inference (and training) during the actual memory read[15]. These approaches typically involve multiple bit- or word-lines for each synaptic weight, frequently assuming factor-of-2 contributions from different bit- (word-)lines in such a way that the multiplication of the synaptic weight by the neuron excitation gets performed during the memory read. As such, these CIM macros depend more critically on tight device-to-device variability than conventional memory, but with the advantage of avoiding the time of accessing memory words individually, and the area and power of the circuit blocks that would otherwise be required to multiply and add the data. The advantages of SRAM-based CIM macros are that the high-endurance of SRAM allows small macros to be reused across many different weight values, so that macro size does not need to match the total size of the AI/ML model; the advantages of RRAM-based CIM macros are, at least eventually, in higher density (e.g., lower area-per-bit).

ANALOG AI

An emerging class of accelerators is based on analog computing—leveraging current summation within dense analog arrays to perform large vector-matrix computations extremely efficiently, allowing both high speed and high energy efficiency[16]. This approach is particularly well-suited to DNN layers with small weight-reuse factors, such as the fully-connected layers found in LSTM and Transformer networks. However, many of these analog systems depend on emerging nonvolatile memory devices, which are not yet a widely-available technology in semiconductor foundries, or on the extension of existing nonvolatile memories that have stopped scaling, such as NOR-FLASH. In addition, it is not yet clear whether the inherently limited signal-to-noise ratios in analog systems can be brought high enough to robustly support the very large models used in modern DNN systems. In contrast, the designer of a digital accelerator can choose to dial down the precision of the digital computations almost arbitrarily, aiming for the best possible energy efficiency while still fully supporting the original neural network accuracy.

4.2.4. CONSIDERATIONS FOR OPTIMIZATION

In this section, we list some of the design-knobs being used extensively today to gain performance while minimizing any loss in accuracy/precision, as well as important variables that must be considered, ranging from software through mixed hardware-software to fully hardware-specific techniques.

4.2.4.1. QUANTIZATION

Quantization is the idea of running inference at lower precision[17]. Most modern DNNs perform inference at 16b and 8b, and in fact some also go lower to 4b. Lower precision naturally reduces the amount of data to move, the time to perform a multiply-accumulate, and add more compute resources in the available area. Both the total number of bits, and for floating-point formats, the number of bits dedicated to the exponent, can have a significant impact.

4.2.4.2. PRUNING

Pruning is an approach leveraged during training, whereby certain excitations, weights, or both are set to zero based on some heuristic (e.g., threshold-based, or block-based)[18]. It has been shown that pruned networks can operate at the same accuracy as the dense ones, if additional training is performed after pruning. Pruning can decrease the required memory footprint by reducing the total number of weights, as well as decrease total runtime and latency (and increase throughput) by reducing the required number of computations. However, the subsequent DNN is then described by sparse matrices. Unless the hardware is expressly designed to take advantage of sparsity, this can affect the utilization of the compute units within hardware.

4.2.4.3. HARDWARE-SOFTWARE CO-DESIGN

Hardware-software co-design is an approach where the hardware AND the neural network models are jointly designed together. This path can lead to significantly higher energy efficiencies, although it works best when the set of DNN models that must be supported is small (typically a small set of Convolutional Networks for particular image-processing applications), and when the long cycle time for custom hardware can be tolerated.

4.2.4.4. UTILIZATION

Utilization is the fraction of time that digital processing elements—within a CPU, GPU or digital accelerator—are performing useful computations. Typically, poorly-designed dataflows can lead processing elements to be idle because the next data to be computed has not yet arrived. Given the large amounts of leakage power in the many transistors within digital systems, keeping utilization high by minimizing any idle processing elements is key for high energy efficiency.

4.2.5. BENCHMARKING

A standardized and robust benchmarking methodology is crucial, both for making the right tradeoffs across this varied hardware landscape during platform selection, and for identifying key bottlenecks that in turn can drive innovation in the ecosystem.

4.2.5.1. Metrics of Interest

The metrics of interest when benchmarking AI platforms depend on the use-case of inference vs training. Table AB-4 highlighted the differences in platform capabilities depending on the deployment model.

Training in the Cloud

For training in the cloud, the metrics of interest are typically the final trained accuracy, and the runtime needed to get to that specific accuracy target. While accuracy is fundamentally dependent on the DNN model and its datasets, it also depends on the parallelization strategy and hardware precision employed. For instance, data parallel training divides the training set into "mini-batches"—and the size of the mini-batch has been found to affect overall training accuracy. Training with 32-bit floating point, vs 16-bit floating point formats (e.g., including the custom bfloat16 format, which offers the same range of exponents as the 32-bit format) can also affect overall accuracy. The runtime to get to a specific accuracy target depends on the efficiency of the hardware platform, which depends on both the capability of a single node (e.g., GPU) and the memory plus network capabilities of a cluster of nodes (when running distributed training). Other approaches for speeding distributed training involve careful compression of the weight update data exchanged between the nodes.

Inference in the Cloud

Inference in the cloud is characterized by end-to-end latency, and the observed accuracy/quality of the output. Inference on the cloud often leverages mini-batching as well, which increases energy efficiency by data reuse. However, given the stochastic nature of AI/ML inference requests, latency constraints can sometimes force the sub-optimal execution of inference jobs with less than a full mini-batch of input data.

Inference at the Edge

Inference at the edge is characterized by real-time latency, the observed accuracy/quality of the output, and energy-efficiency. This is because edge platforms are highly energy constrained, and trading off accuracy for runtime and energy-efficiency is often an acceptable trade-off. There is thus an emerging trend in developing edge-platform friendly DNN models (e.g., MobileNetv2, SqueezeNet) leveraging frameworks such as TensorFlowLite. Typically, edge inference is performed in smaller batches, although inference in autonomous vehicle may involve a continuous stream of images from multiple sensors.

Training at the Edge

As discussed in Section 4.2.2, training on the edge is another possible deployment platform, although not yet mainstream. Here the metrics of interest might be the amount of new data needed to improve the accuracy of the model while under deployment.

4.2.5.2. Benchmarking Environment

For an effective AI/ML benchmark, first the benchmark would have to standardize the testing/benchmarking environment. For AI/ML applications, this involves standardizing models based on use-cases like computer vision, speech recognition and others. It is evident that given the difference in processing, the benchmark should be separate for Training and Inference processes. Some of the recommended metrics that can be captured as part of the benchmarking efforts are listed in Table AB-5.

Table AB-5 AI/ML Benchmarking Metrics

	Description	**Units**
Compute	Raw Compute Capability	TOPS (Tera, or 1e12 operations per second)
Inference Speed	Number of features classified per second	Frames/Sec
Training Speed	Time to train any model at specified accuracy, e.g., 90% accuracy	Time in minutes
Power	Processing output / Watt	Inference Speed/ Watt Training Speed / Watt

4.2.5.3. MLPERF.ORG

In May 2018, a consortium of companies and universities announced[19] the creation of the MLperf.org website and github repository. Currently grown to 64 companies and 8 universities, the consortium's website describes two broad benchmark suites for ML applications, in both training and inference.

Motivated by the SPEC benchmark, "MLPerf's mission is to build fair and useful benchmarks for measuring training and inference performance of ML hardware, software, and services. We believe that a widely accepted benchmark suite will benefit the entire community, including researchers, developers, hardware manufacturers, builders of machine learning frameworks, cloud service providers, application providers, and end users.

"Our goals include:

- Accelerate progress in ML via fair and useful measurement
- Serve both the commercial and research communities
- Enable fair comparison of competing systems yet encourage innovation to improve the state-of-the-art of ML
- Enforce replicability to ensure reliable results
- Keep benchmarking effort affordable so all can participate"[20]

Recently, the first set of MLperf Inference results (v0.5) were released[5]. Other AI/ML benchmarks include AI Benchmark, EEMBC MLMark, Fathom, AIXPRT, AI Matrix, DeepBench, TBD (Training Benchmarks for DNNs), and DawnBench. [5 and references within]. So far, no MLperf results have included power, although this is planned for future versions of the benchmark.

The goals of the MLperf organization are very closely aligned with this chapter. Key differences are that MLperf focusses on a current "snapshot" of AI/ML performance, and explicitly compares particular systems with each other by name. Improvements that arise from better frameworks, microcode, or mapping from frameworks may be conflated with improvements coming from the underlying hardware. In contrast, the IRDS roadmap is more interested in longer-term trends in improved AI/ML systems, as explicitly traceable to changes in the underlying hardware.

If MLperf is the right source for information on which commercial system to acquire today, the goal we are striving towards with this IRDS/AB section is to predict what level of AI/ML performance will be commercially relevant in 2–3 years' time.

4.2.5.4. DIFFICULTIES IN BENCHMARKING

While this is a lofty goal, there is well-known Danish saying, sometimes attributed to Niels Bohr: "It is difficult to make predictions, especially about the future."

A key challenge with designing and benchmarking AI platforms is the tightly integrated stack—from the DNN models down to hardware (Figure AB-3). Unlike traditional SW-HW platforms where clear abstractions exist, the efficiency of AI platforms actually comes from breaking the SW-HW boundary. This makes it challenging to tease apart benefits coming from innovations in hardware, compilers, and DNN development frameworks (e.g, TensorFlow, PyTorch).

While there are some emerging benchmarks that focus on essential component blocks of DNN systems[21], it is not clear that a benchmarking approach focused solely on component blocks could accurately track full forward-evaluation or training times, nor how such an approach would provide reassurance that an approximate digital or analog technique was achieving effectively identical classification accuracies.

Finally, compute efficiency is typically measured in terms of TOPS/W, short for TeraOPS/W, or 1e12 operations per second per Watt. Older hardware is sometimes rated in terms of TFLOPS/W, but the use of reduced precision has led to the need to exclude the phrase "Floating Point" to avoid confusion. In terms of energy efficiency, there is a significant difference between peak and actual compute efficiency, especially for DNN models that fall well below 100% utilization. For digital accelerators, this can be quantified nicely in the form of a rooftop curve, plotting computational performance as a function of operations[10].

4.2.6. PERFORMANCE TRENDS AND PREDICTION

In the meantime, here we show some publicly available data on raw computational capabilities of various systems either released or announced for training and inference of DNNs.

Figure AB-5 shows a plot of computational efficiency in units of TOPS/W for recent CPUs, GPUs, FPGAs and projected or advertised capabilities for some custom DNN hardware (labelled as ASICs). Here we focus on hardware that could be used for training, using either FP32 or BF16 precision formats. For completeness, we also include FP16 hardware as well, although some reports indicate that not all models can be trained accurately with the original half-precision format without adaptations, and that the BF16 format—which uses the same number of bits for the exponent as FP32, but many fewer in the mantissa—is required. On the other hand, there are also recent reports that training can be done at 8-bit precision as well[22]. Figure AB-6 replots the data from Figure AB-5, showing how TOPS (data points) and the resulting TOPS/W (assessed against the green dotted lines) vary as a function of reported system power.

The separate spreadsheet associated with this report shows a breakdown of numbers used to generate this plot, including references for where each data point was obtained.

Figure AB-7 shows a similar plot of computational efficiency in units of TeraOPS/W for numerous recent CPUs, GPUs, and FPGAs, as well as projected or advertised capabilities for some near-future DNN hardware explicitly designed for inference and labelled as ASICs. Note that across these plots, one-time improvements based on reduced precision (from 32 to 16 to 8 bits) are clearly revealed. Figure AB-8 analogously breaks out the TOPS and TOPS/W performance as a function of reported system power.

Finally, Figure AB-9 plots the efficiency of AI/ML compute as a function of claimed compute performance in TOPS. Efficiency is computed by taking the ratio of actual TOPS delivered in EEMBC testing on three workloads (SSDMobileNet 1.0, MobileNet 1.0, and ResNet-50 1.0)[23] to the claimed peak TOPS. Details of these calculations, including references for the performance results, are available in the separate spreadsheet associated with this report. Even for well-known convolutional neural networks that feature significant weight reuse, and which are widely known and used, only a portion of the claimed peak TOPS are apparently available when running on real workloads. For higher peak TOPS, which are also associated with higher power systems, efficiency can be at or below 10%.

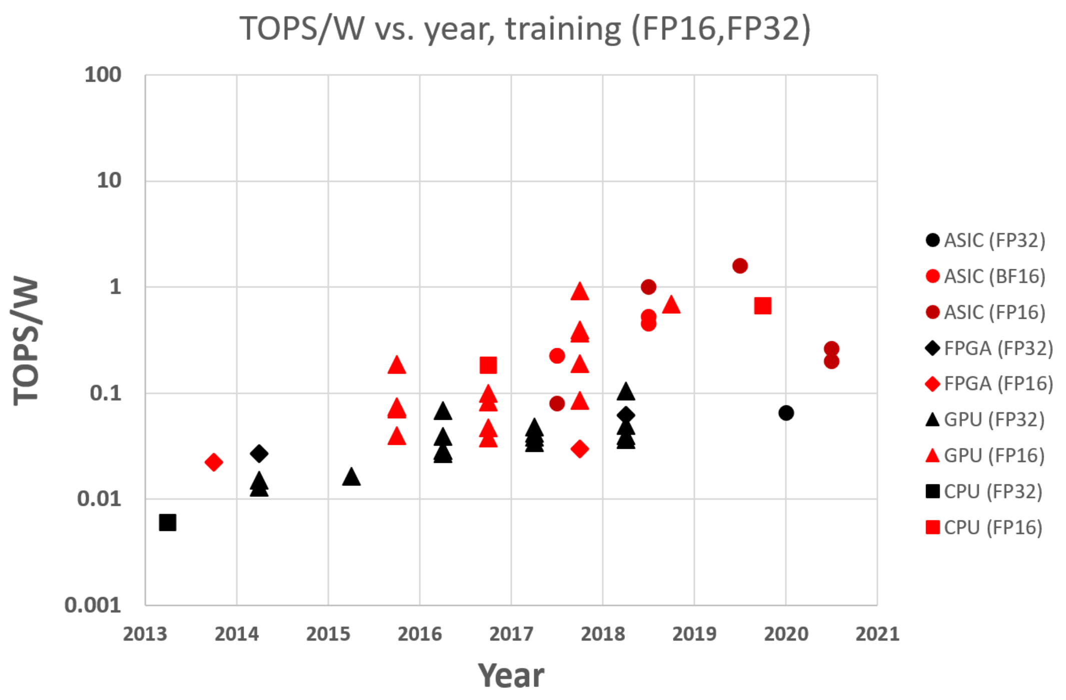


Figure AB-5 Compute Efficiency (TOPS/W) of Existing and Projected Hardware for Training

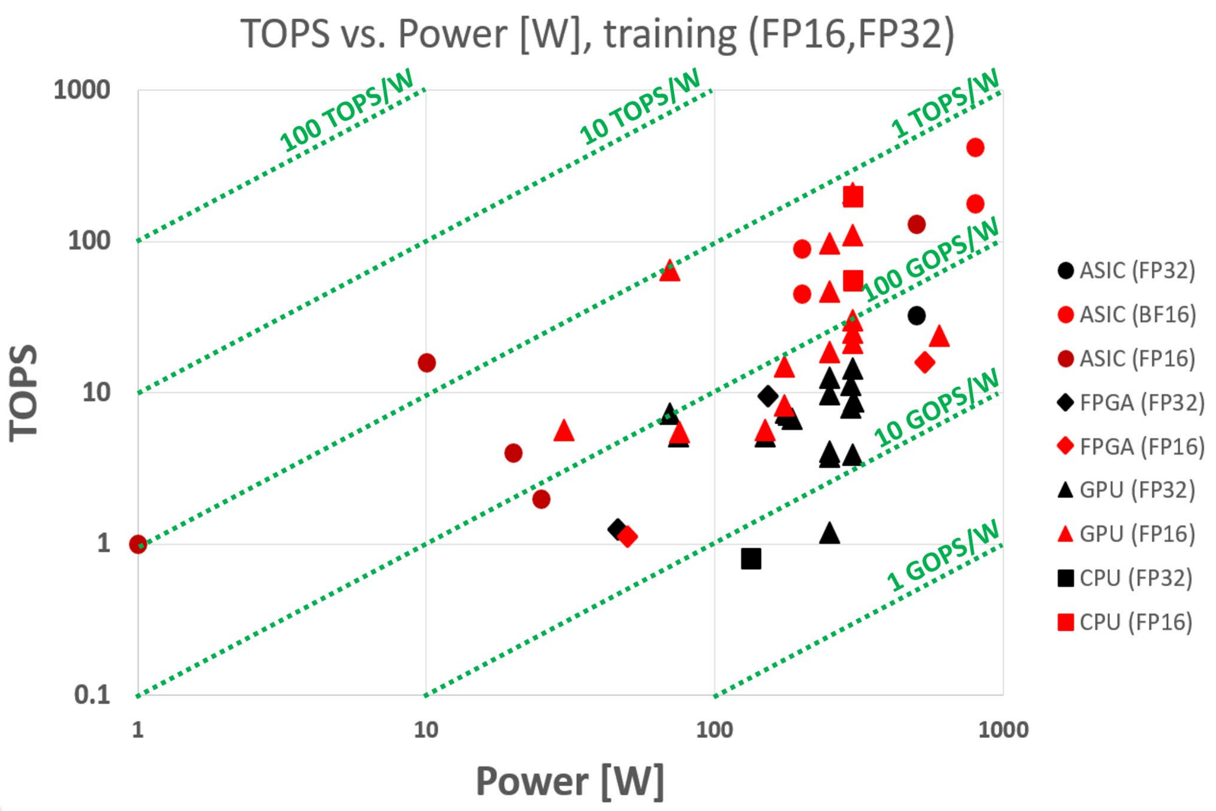


Figure AB-6 Data from Figure AB-5 for Existing and Projected Hardware for Training

REPLOTTED AS TOPS AS A FUNCTION OF REPORTED SYSTEM POWER

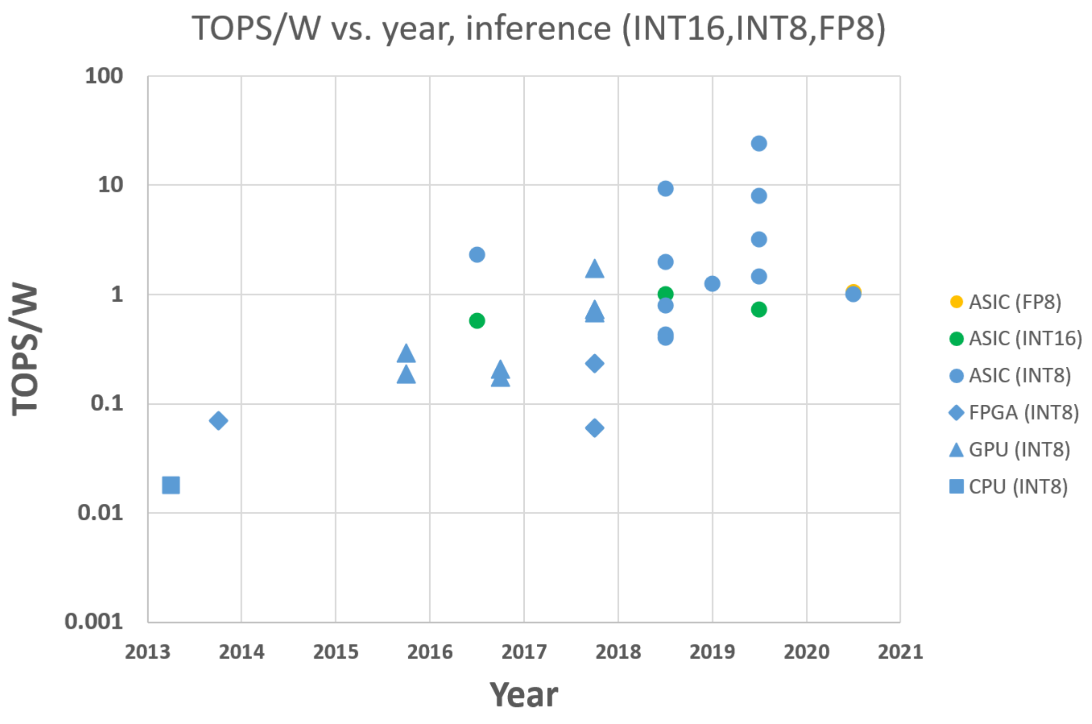


Figure AB-7 Compute Efficiency of Existing and Projected Hardware for Inference

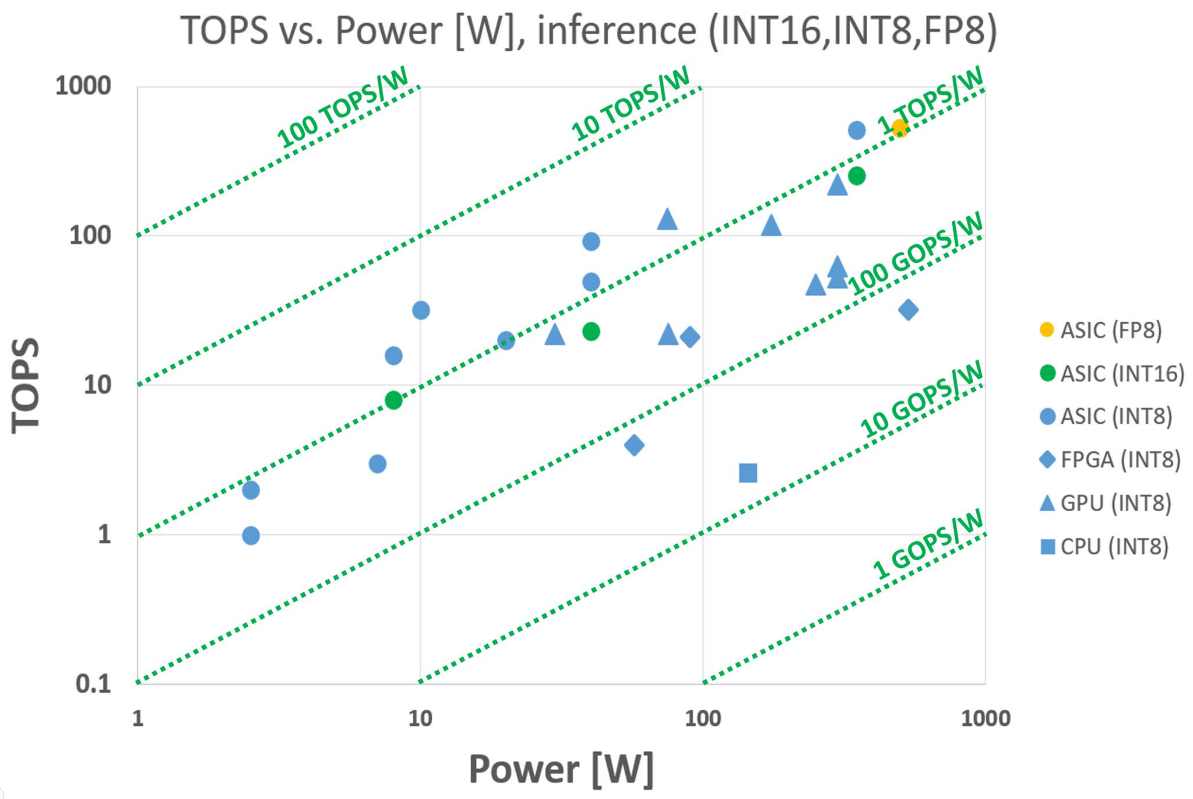


Figure AB-8 Data from Figure AB-7 for Existing and Projected Hardware for Inference

REPLOTTED AS TOPS AS A FUNCTION OF REPORTED SYSTEM POWER

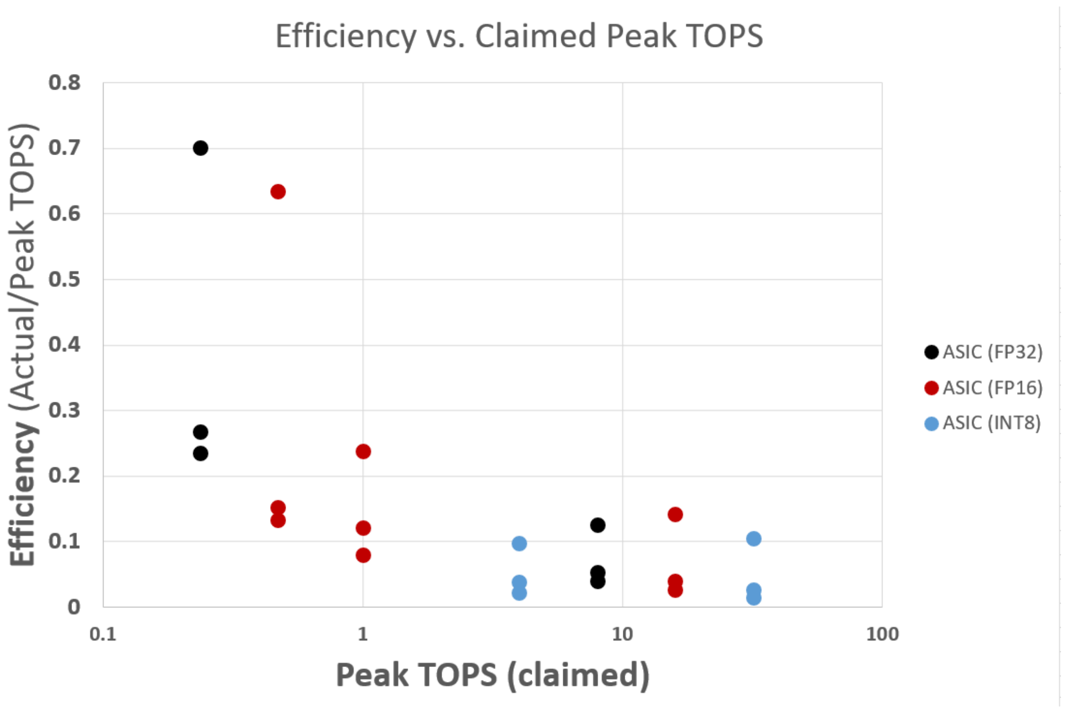


Figure AB-9 Efficiency as a function of claimed peak TOPS, showing the percentage of claimed compute power actually delivered in practice

4.2.7. Technology Needs

Key technology needs for Artificial Intelligence depend on the computing architecture (see Table AB-4). Generally, efficiency increases with specialization. Thus, more specialized FPGA and ASIC approaches for both training and inferencing will lead to the highest efficiencies. Any technology that enables higher efficiencies in FPGA and ASIC implementations will accelerate Artificial Intelligence. Also technologies that enhance near memory computation will improve pruned DNNs since they in turn result in sparse matrices.

4.2.8. System and Architectures Impact

Approaches to deployment platforms and their corresponding constraints are summarized in Table AB-4. As can be seen, the highest performance deployment platforms are the least energy constrained (e.g., cloud platforms). AI remains constrained by the memory bandwidth and latency. Because of optimizations such as pruning's impact on memory access patterns due to sparsity, increasing the capacity of caching does not necessarily lead to higher performance. Training is constrained also by the available parallelism on more commodity platforms such as multicore processors and GPUs.

4.3. Discrete Event Simulation

A discrete event simulation (DES) models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system[24]. Between consecutive events, the simulation can directly jump in time from one event to the next. In contrast, with continuous simulation, time ins broken up into small time slices and every time slice is simulated to update the system state. DES can typically run much faster than continuous simulation as it does not have to simulate every time slice.

Common uses of DES include diagnosing process issues, testing performance improvement ideas, evaluating capital investment decisions (with Monte Carlo), and simulating computer networks. DES typically include the following components: State, Clock, List of events, Random-number generators, and Statistics.

471.omnetpp in the SPEC CPU 2006 benchmark suite represents a DES workload[25]. This benchmark performs discrete event simulation of a large ethernet network. The simulation is based on the OMNeT++ discrete event simulation system, a generic and open simulation framework[26]. OMNeT++'s primary application area is the simulation of communication networks, but its generic and flexible architecture allows for its use in other areas such as the simulation of IT systems, queueing networks, hardware architectures or business processes as well.

Although SPEC CPU 2006 has been deprecated in favor of SPEC CPU 2017, OMNeT++ lives on in SPEC CPU 2017 with a much larger model and input sets. While 471.omnetpp in SPEC CPU 2006 modeled networks with Gigabit Ethernet, 620.omnetpp_s in SPEC CPU 2017 simulates 10 Gbps network model. For the reference workload, the simulated network models a large network comprised of six backbone switches, eight small LANs, twenty medium LANs and twelve large LANs.

Galois also includes a benchmark that represents discrete event simulation[27]. Two different implementations, based on ordered and unordered algorithms, are present in the Galois suite. Unlike 471.omnetpp in SPEC CPU 2006, the Galois implementations are multi-threaded, although their initial scalability analysis indicates that high number of threads is limited due to lack of parallelism in the workload. Threads have to wait more often to remove items from the worklist, which becomes a bottleneck.

4.3.1. Performance Trends and Prediction

In this section, we analyze the performance trend of 471.omnetpp and 620.omnetpp_s using historical data points uploaded to the SPEC database. For 471.omnetpp from SPEC CPU 2006, we downloaded roughly 9,500 data points of various systems running the workload ranging from March 2006 to December 2017. SPEC has stopped accepting new results for the SPEC CPU 2006 benchmarks starting in 2018. We also downloaded roughly 3,000 data points for 620.omnetpp_s ranging from October 2017 to September 2019. We bucketed these data into monthly bins and calculate the maximum and average scores for each bin. SPEC reports the "base" and "peak" scores for each workload. According to SPEC, the base metrics are required for all reported results and have stricter guidelines for compilation. For example, the same flags must be used in the same order for all benchmarks of a given language. The peak metrics are optional and have less strict requirements. For example, different compiler options may be used on each benchmark, and feedback-directed optimization is allowed.

Figure AB-10 shows the performance of 471.omnetpp over time. We plot both base and peak performance with maximum and average scores per monthly bins, represented by the four solid lines. We also plot the linear regression of each metric, represented by the four dotted lines. In general, performance appears to be improving over time. Note that there are occasional steep jumps in performance. These generally align well with major CPU releases as depicted in the figure. One thing to note is that the data uploaded to SPEC only has the test date associated with the data, not the system release date. While, in theory, it is possible to look up all the system release dates for all 9,500 data points, we assumed that, in general, newer systems would be more popular to benchmark at any given time, and, thus, the test dates would align well with the system release dates.

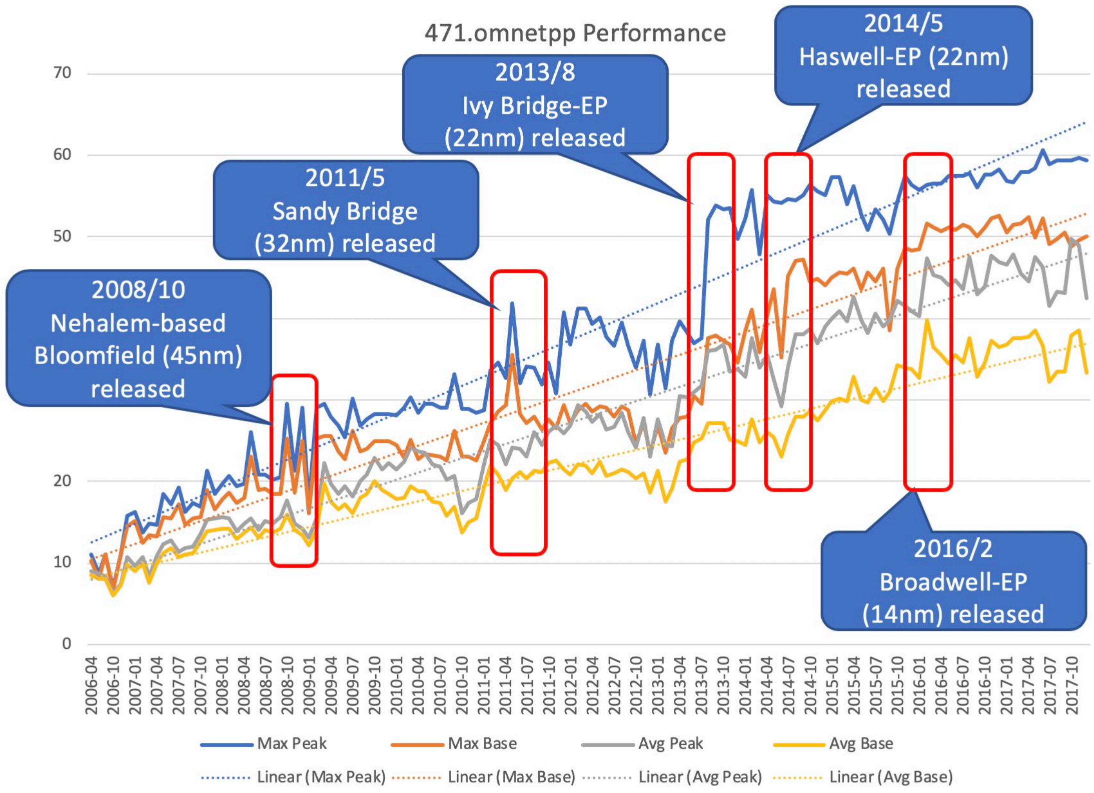


Figure AB-10   Historical Performance of 471.omnetpp Over Time

Figure AB-11 shows the performance trend of 620.omnetpp_s from SPEC CPU 2017. We have roughly 3,000 data points from spec.org, but the history is not long enough to project a meaningful trend yet. We will continue to track this workload going forward.

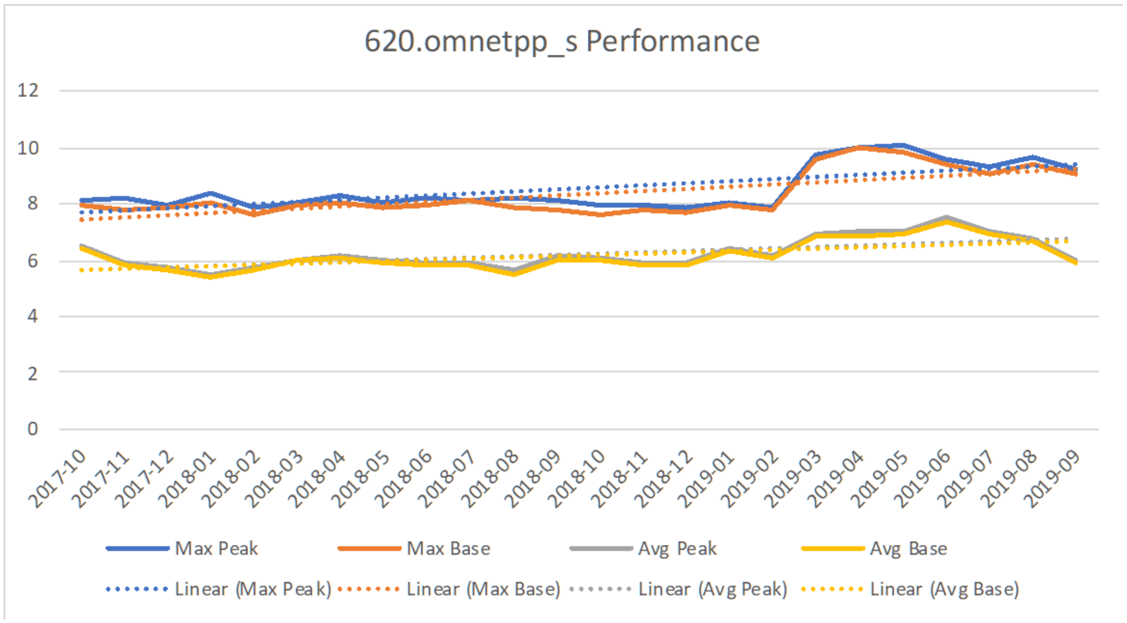


Figure AB-11   Historical Performance of 620.omnetpp_s over time

4.3.2. Technology Needs

DES, represented by 471.omnetpp in SPEC CPU 2006, generally has lower than average instructions-per-cycle (IPC). It is characterized to be memory bound with high L2 cache miss rates. It also has a relatively large code footprint and, thus, is sensitive to the instruction cache size.

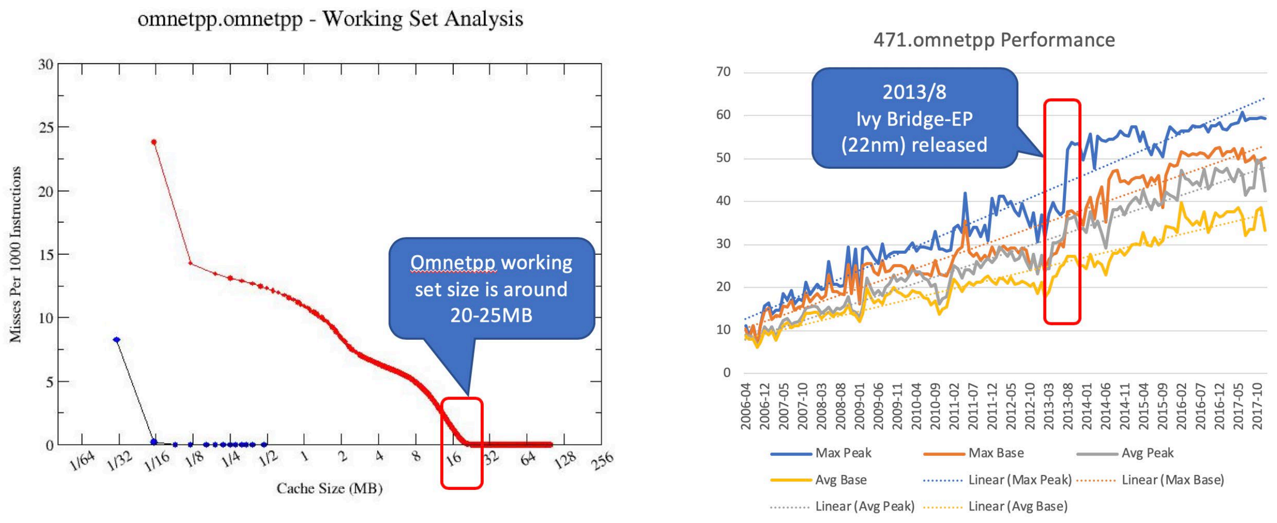


Figure AB-12 Working Set Analysis of 471.omnetpp and Dramatic Jump in Peak Performance Due to Working Set Fitting in Cache

Figure AB-12 (left) shows the working set analysis of 471.omnetpp from[28]. Once the data cache size reaches 20–25 MB, the working set almost completely fits in the cache and the misses per kilo-instructions (MPKI) reaches zero. This could explain the largest jump in 471.omnetpp performance around August of 2013. The Ivy Bridge-EP processor released around this timeframe is the first processor to have a 25 MB last-level cache, which is enough to fit the entire working set of 471.omnetpp. This could also explain why the max peak performance of the benchmark has remained somewhat flat after that point. Of course, the whole domain of DES should not be restricted to this single benchmark with a limited size input set. However, being memory bound and cache sensitive would be a general characteristic to describe DES workloads.

471.omnetpp is also notorious for having an irregular memory access pattern and thus being hard to prefetch. The event queues in DES workloads are typically modeled as priority queues sorted by event time. Regardless of the order in which events are added to the event set, they are removed in strictly chronological order. To improve performance, priority queues typically use a heap as their backbone, giving better performance for inserts and removals. From a computational-complexity standpoint, priority queues are similar to sorting algorithms. Sorting algorithms can exhibit an irregular memory access pattern depending on the input data. Intelligent prefetching and cache replacement to reduce the effective memory access latencies can improve the performance of DES workloads.

With SPEC CPU 2017, 620.omnetpp_s continues to carry similar characteristics from its CPU 2006 version, but the memory pressure appears to have intensified. Table AB-6[29] shows the L2 and LLC cache access breakdown for all SPEC 2017 Integer Speed workloads. We can see that 620.omnetpp_s has higher LLC misses per 1K instructions and higher LLC miss rate than 605.mcf_s, which is traditionally considered very memory intensive.

Table AB-6 L2 and LLC access breakdown for SPEC Int Speed 2017

int_speed	L2 ref. per 1K Inst.	L2 misses per 1K Inst.	% L2 misses	LLC ref. per 1K Inst.	LLC misses per 1K Inst.	% LLC misses
600.perlbench_s_a	16.72	4.27	25.53	6.95	0.35	5.04
600.perlbench_s_b	38.55	8.47	21.98	13.34	0.31	2.32
600.perlbench_s_c	23.36	7.24	30.99	12.15	4.39	36.37
602.gcc_s_a	119.41	33.05	27.68	50.24	17.41	34.81
602.gcc_s_b	80.09	22.77	28.43	37.17	2.50	6.71
602.gcc_s_c	72.91	21.32	29.24	34.92	3.03	8.67
605.mcf_s	330.48	108.55	32.85	156.32	28.55	18.27
620.omnetpp_s	170.54	52.77	30.93	79.66	33.75	42.37
623.xalancbmk_s	243.56	88.07	36.16	131.54	5.14	3.90
625.x264_s_a	21.99	4.13	18.80	7.10	2.19	30.66
625.x264_s_b	24.38	5.54	22.69	9.69	0.93	9.66
625.x264_s_c	21.52	4.87	22.64	8.43	0.86	10.13
631.deepsjeng_s	18.84	3.65	19.32	5.32	3.51	65.98
641.leela_s	10.10	1.50	14.87	2.79	0.04	1.52
648.exchange2_s	0.22	0.01	5.04	0.02	0.00	0.22
657.xz_s_a	40.20	11.41	28.42	18.09	2.02	11.19
657.xz_s_b	107.89	36.56	33.88	55.77	21.46	38.47

4.3.3. System and Architectures Impact

Traditional general-purpose CPUs would still be suited to run DES workloads efficiently. It is unclear whether any type of accelerators can be helpful. Being memory-bound, process-in-memory systems might be interesting, but the need for frequent synchronization would limit the scalability as shown by the Galois benchmarks mentioned earlier.

Most DES workloads would benefit from larger and faster caches. Novel prefetching schemes that can effectively capture the irregularity in the access pattern would significantly help with further improving the performance.

Larger-scale DES workloads would require breaking up the physical system into multiple logical processes. However, this is complicated as the computing nodes will be exchanging timestamped messages while often operating at different points of simulation time. Highly efficient synchronization methods must be used to effectively scale such workloads.

4.4. Physical System Simulation

Computer simulation of physical real-world phenomena emerged with the invention of electronic digital computing and is increasingly being adopted as one of the most successful modern methods for scientific discovery. One of the main reasons for this success has been the rapid development of novel computer technologies that has led to the creation of powerful supercomputers; large distributed systems such as high-performance computing facilities; access to huge data sets, and high throughput communications. In addition, unique and sophisticated scientific instruments and facilities, such as giant electronic microscopes, nuclear physics accelerators, or sophisticated equipment for medical imaging are becoming integral parts of those complex computing infrastructures. Subsequently, the term 'e-science' was quickly adopted by the professional community to capture these new revolutionary methods for scientific discovery via computer simulations of physical systems[30].

The most important application codes for physical system simulations are typically based on finite-element algorithms—such as boundary element method, N-body problem, fast multipole method, hierarchical matrices, iterative stencil computations—while the computations constitute heavy workloads that conventionally are dominated by floating-point arithmetic. Example applications include areas such as climate modelling, plasma physics (fusion), medical imaging, fluid flow, and thermo-evolution. In addition, physical system simulation is critical to product design in the automobile and aerospace industries as well as for obtaining more accurate climate modelling and prediction.

From the point of view of application programmers and end-users, the following major benchmarking efforts have been part of the development of this field over the years:

a. The NAS Parallel Benchmarks (NPB) include the descriptions of several (initially eight) "pencil and paper" algorithms[31]. All are realistic kernels, although the authors' claim that they include three "simulated applications" as a more accurate description 25 years ago than it is today. The NPB cover only the Computational Fluid Dynamics (CFD) application domain that is a primary interest of NASA.
b. The GENESIS Distributed-Memory Benchmarks[32] were developed in a 3-layer hierarchy – low-level micro-benchmarks, kernels, and compact applications. This was intended to express the performance of higher level codes via a composition of performance results produced by the codes in the layer below. However, this proved to be a difficult task, particularly with including sufficiently broad computational science codes in the compact application layer.
c. The PARKBENCH Public International Benchmarks for Parallel Computers[33]. This was an ambitious international effort to glue together the most popular parallel benchmarks at that time – NPB, GENESIS, and several kernels including LINPACK. The PARKBENCH suite adopted the hierarchical approach from GENESIS together, thus inheriting the same difficulties described above.
d. All major machine vendors have participated in the development of SPEComp2001, since achieving portability across all involved platforms was an important concern in the development process[34]. The goal was to achieve both functional and performance portability. Functional portability ensured that the makefiles and run tools worked properly on all systems, and that the benchmarks ran and validated consistently. To achieve performance portability, SPEComp2001 accommodated several requests by individual participants to add small code modifications that took advantage of key features of their machines. There are many SPEComp2001 benchmarking results available, but their main role is to confirm that new hardware products and platforms have been validated by the vendors.
e. Another more recent "pencil and paper" parallel benchmark suite is the Dwarfs Mine based on the initial "Seven Dwarfs" proposal (2004) by Phillip Colella. The Dwarfs (computation and communication patterns) are described as well-defined targets from algorithmic, software, and architecture standpoints. The number of Dwarfs (which are really kernels with some of them mapped to NPB) was then extended to 13 in the "View from Berkeley" Technical Report [35]. The report confirms "presence" of the 13 Dwarfs in 6 broad application domains – embedded computing, general-purpose computing, machine learning, graphics/games, databases and RMS (recognition/mining/synthesis). While this is a very interesting approach, the availability of results is very limited and even more importantly, the application domains are different from the ones selected by IRDS AB IFT. Some recent studies suggest that more Dwarfs (kernels) should be added for other application domains, while it is also not clear if the existing ones are sufficient for the domains described in the "View from Berkeley" Technical Report.

4.4.1. PERFORMANCE TRENDS AND PREDICTION

Most of the benchmarking projects mentioned above cover predominantly legacy dense applications, in which high computational intensity carries over when parallel implementations are built to solve bigger problems faster than can be handled by single processing server nodes. This means that inter-node communication can carry far less data per second than memories provide inside a node, and communication over such interconnection networks can be supported largely by explicit program calls to software stacks that manage the data transfers. As long as emphasis was on dense problems, this approach resulted in systems with increasing computational performance and was the presumption behind the selection of the LINPACK benchmark for the very popular semi-annual TOP500 rankings of supercomputers[2].

However, many new applications with very high economic potential—such as big data analytics, machine learning, real-time feature recognition, recommendation systems, and even physical simulations - have been emerging in the last 10-15 years. These codes typically feature irregular or dynamic solution grids and spend much more of their computation in non-floating-point operations such as address computations and comparisons, with addresses that are no longer regular or cache-friendly. The computational intensity of such programs is far less than for dense kernels, and the result is that for many real codes today, even those in traditional scientific cases, the efficiency of the floating-point units that have become the focal point of modern core architectures has dropped from the >90% to <5%. This emergence of applications

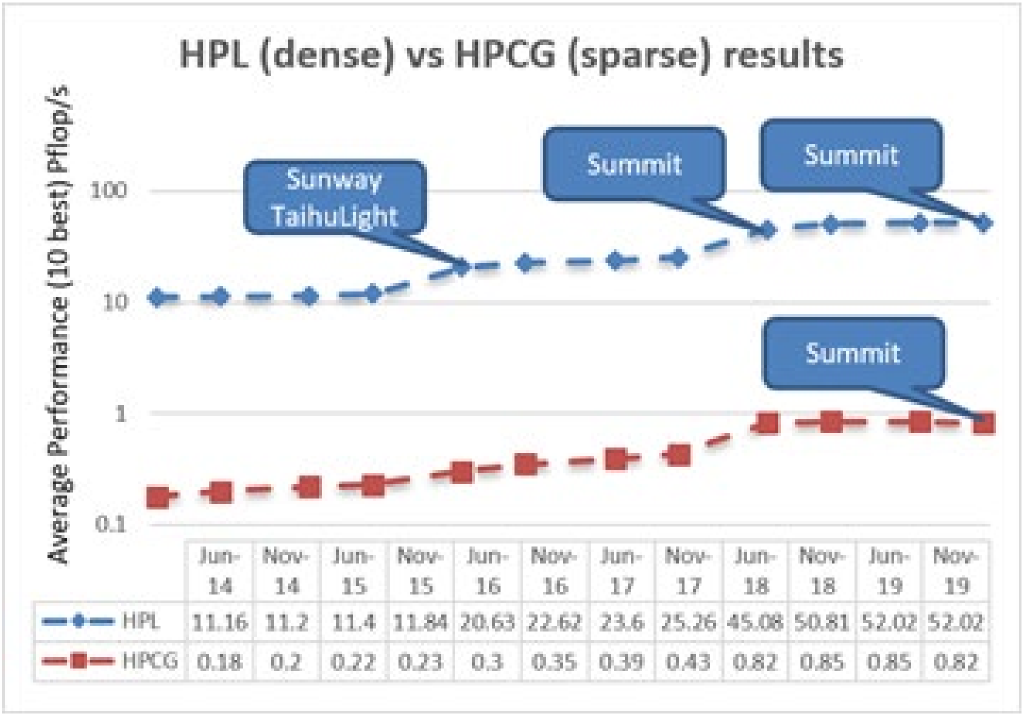
[2] *https://www.top500.org/*

with data-intensive characteristics—e.g., with execution times dominated by data access and data movement—has been recognized recently as the 3rd Locality Wall for advances in computer architecture[36].

To highlight the inefficiencies described above, and to identify architectures which may be more efficient, a new benchmark was introduced in 2014 called HPCG[3] (High Performance Conjugate Gradient). HPCG also solves Ax=b problems, but where A is a very sparse matrix—normally, with 27 non-zeros in rows that may be millions of elements in width. On current systems, floating point efficiency mirrors that seen in full scientific codes. For example, one of the fastest supercomputers in the world in terms of dense linear algebra is the Chinese TaihuLight, but that same supercomputer can achieve only 0.4% of its peak floating-point capability on the sparse HPCG benchmark. Detailed analysis lead to the conclusion that HPCG performance in terms of useful floating-point operations is dominated by memory bandwidth to the point that the number of cores and their floating-point capabilities are irrelevant[37]. There are of course application codes with highly irregular and latency-bound memory access that deliver significantly lower performance, but they are uncommon. While HPCG does not represent the worst-case scenario, it has been widely accepted as a typical performance yardstick for memory-bound applications.

Therefore, our selected benchmark codes that cover the "Physical System Simulation" application area of interest are the High-Performance LINPACK (HPL)[38] [39] and the HPCG[40]. Both are very popular codes with very good regularity of results since June 2014. Another very important reason for selecting HPL and HPCG that they represent different types of real-world phenomena—the HPL models dense physical systems while the HPCG models sparse physical systems. Therefore, the available benchmarking results provide excellent opportunities for comparisons and interpretation, as well as lay out a relatively well-balanced overall picture of the whole application domain for physical system simulation.

Our approach is to explore a 3-dimensional space—dense systems performance, sparse systems performance, and energy efficiency for both cases. With HPL as the representative of dense system performance and HPCG as the representative for sparse systems, there are readily available performance and energy results published twice per year (June and November) with rankings of up to 500 systems for those two benchmarks since June 2014. We have further decided to use the average of the top 10 performance and energy results for each of these two benchmarks. This latter choice could be a point for further discussion and optimization of the benchmarking approach for this application domain. We have selected the 10 best only (rather than a larger number) because of the very limited HPCG results in the early years of publicly available HPCG measurements.



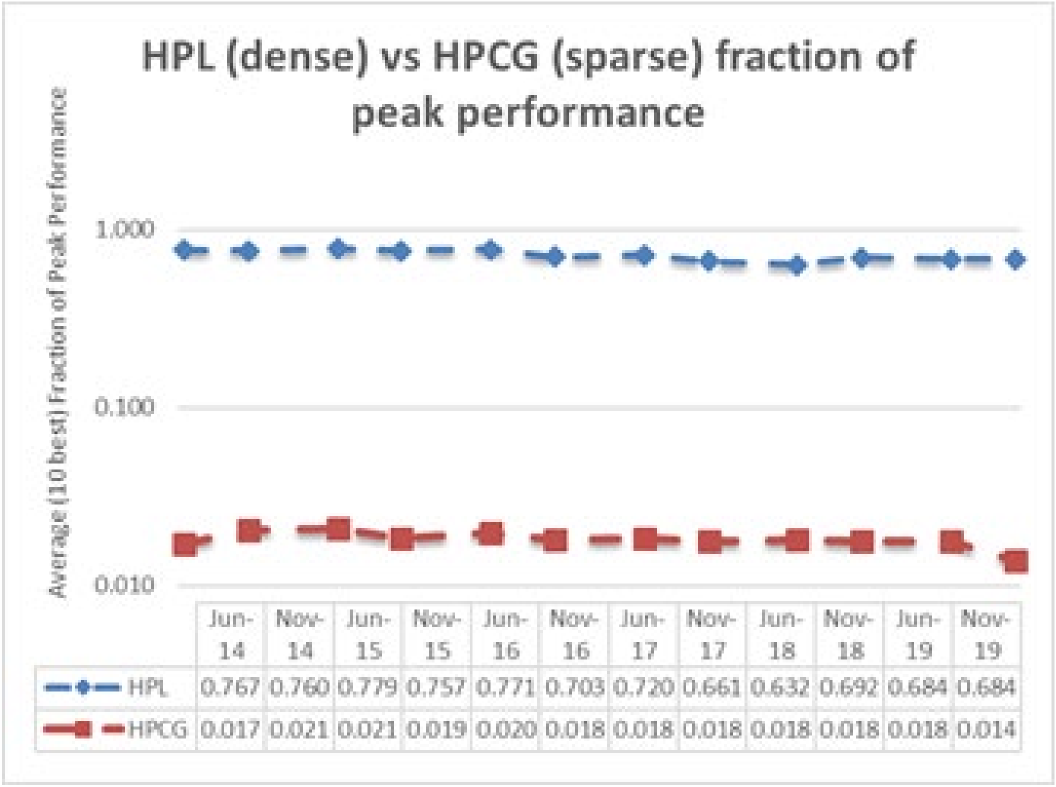
	Jun-14	Nov-14	Jun-15	Nov-15	Jun-16	Nov-16	Jun-17	Nov-17	Jun-18	Nov-18	Jun-19	Nov-19
HPL	11.16	11.2	11.4	11.84	20.63	22.62	23.6	25.26	45.08	50.81	52.02	52.02
HPCG	0.18	0.2	0.22	0.23	0.3	0.35	0.39	0.43	0.82	0.85	0.85	0.82

Figure AB-13 HPL (dense systems, blue) vs. HPCG (sparse systems, red) Average Performance

[3] *http://www.hpcg-benchmark.org/*

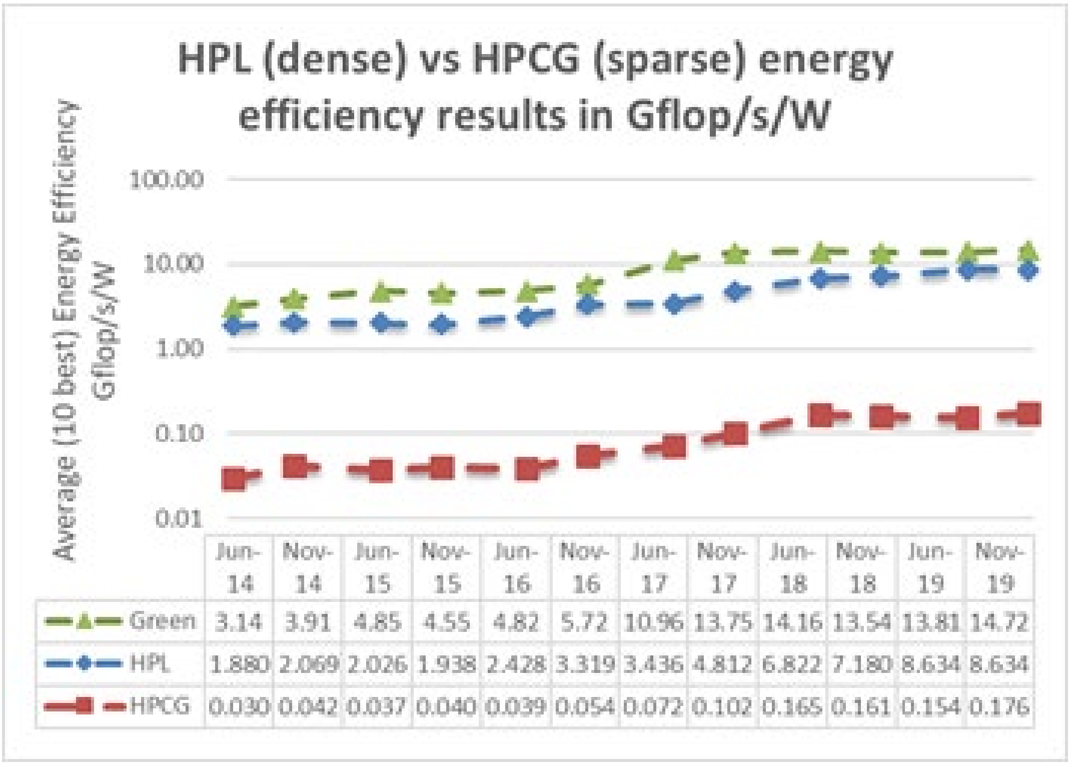
Figure AB-13 shows a significant performance gap of nearly 2 orders of magnitude between HPL and HPCG results in the last several years. The increase of the average HPL performance since June 2016 is because of the introduction of the Chinese Sunway TaihuLight system. The most recent increase of both HPL and HPCG performance is visible since June 2018 after the installation of the Summit supercomputer at ORNL. An optimistic expectation here would be to observe that the gap keeps closing and then assess the rate of this progress. Unfortunately, we do not have any evidence that the observed performance gap is in fact closing to any degree. Thus, we can draw the conclusion that one of the main challenges ahead will be to significantly increase sparse systems performance with any future computing systems designed for this application domain. While it is clear that reaching ExaFLOP/s performance with HPL will happen soon, it is equally clear that this achievement will leave this significant gap between dense and sparse system performance unchanged.

Figure AB-14 complements the above analysis by showing a similar gap of approximately 2 orders of magnitude for the fraction of peak performance between HPL (dense) and HPCG (sparse). This provides clear evidence of something we have known for years – our production codes (which are usually sparse systems) are unable to deliver more than a few percent of the peak system performance that HPL results would seem to promise. The figure shows that this gap has not been reducing, and further points out the need to address sparse system performance in the next generation of computer architectures designed for this application domain.



	Jun-14	Nov-14	Jun-15	Nov-15	Jun-16	Nov-16	Jun-17	Nov-17	Jun-18	Nov-18	Jun-19	Nov-19
HPL	0.767	0.760	0.779	0.757	0.771	0.703	0.720	0.661	0.632	0.692	0.684	0.684
HPCG	0.017	0.021	0.021	0.019	0.020	0.018	0.018	0.018	0.018	0.018	0.018	0.014

Figure AB-14 HPL (dense systems, blue) vs. HPCG (sparse systems, red) Fraction of Peak Performance



	Jun-14	Nov-14	Jun-15	Nov-15	Jun-16	Nov-16	Jun-17	Nov-17	Jun-18	Nov-18	Jun-19	Nov-19
▲ Green	3.14	3.91	4.85	4.55	4.82	5.72	10.96	13.75	14.16	13.54	13.81	14.72
◆ HPL	1.880	2.069	2.026	1.938	2.428	3.319	3.436	4.812	6.822	7.180	8.634	8.634
■ HPCG	0.030	0.042	0.037	0.040	0.039	0.054	0.072	0.102	0.165	0.161	0.154	0.176

Figure AB-15 HPL (dense systems) vs. HPCG (sparse systems) vs. the most energy-efficient supercomputers on the Green 500 list

The energy efficiency dimension of our evaluation is depicted in Figure AB-15. The current supercomputing designs appear to be able to scale up to 200 PetaFLOP/s while remaining within the recommended 20 MW system power consumption envelope. An optimistic estimate based on this would require five times improvements in energy efficiency, and seven times improvements in the HPL performance currently delivered by the Summit supercomputer. However, such improvements are not realistic, since the best energy efficiency results and rankings are different from the HPL ranking (see comments above about the top 10 ranked results). Therefore, a more realistic projection based on the current (end of 2019) Summit results is that one needs ten times energy efficiency improvement and ten times higher HPL performance to reach the ExaFLOP/s barrier. Unfortunately, this would only achieve the desired performance and energy efficiency for the computation of dense physical systems such as the HPL benchmark.

Similar performance vs. energy efficiency analysis and projections for sparse systems based on the HPCG results look much more pessimistic. Here the two orders of magnitude lower performance delivered for sparse systems by the current supercomputing architectures strongly impact the energy efficiency.

4.4.2. Technology Needs

The technological challenges that could help drive further developments in the field of physical system simulation include:

a. Reduced Data Movement.

For several decades, significantly reducing the data movement has been one of the most important challenges towards achieving higher computer performance. Achieving higher bandwidth and lower latency for accessing and moving data—both locally (memory systems) and remotely (interconnection networks)—are key challenges towards building supercomputers at ExaFLOP/s level and beyond. Breakthrough architecture solutions addressing those challenges could potentially enable up to two orders of magnitude higher performance particularly for sparse physical system simulations. More specifically, forthcoming designs of High Bandwidth Memory (HBM) such as HBM3+ and HBM4 expected to be released between 2022 and 2024, are likely to change substantially the application performance landscape for future supercomputers.

b. Efficient Floating-Point Arithmetic.

After several decades, the IEEE 754 Standard for Floating-Point Arithmetic was renewed again in July 2019 [41]. However, the level of interest in this standard has been declining following critical comments about various important aspects of IEEE 754 including wasted cycles, energy inefficiencies, and accuracy. Unfortunately, the path forward is unclear at present and may involve keeping this standard as an option at least for backward compatibility while developing and implementing novel and more efficient solutions. Several efforts to address these problems follow two main approaches:

- Analysis of specific algorithms and re-writing of existing codes in order to improve the performance by using lower floating-point precision without compromising accuracy. This approach has been shown to work well but only for specific algorithms/codes, and with significant dedicated efforts for each case[42].
- More radical approaches proposing new solutions have been under development including the Posit Arithmetic proposal[43]. This work introduces a new data type — posit — as a replacement for the traditional floating-point data type because of its advantages. For example, posits guarantee higher accuracy and bitwise identical results across different systems which have been recognized as the main weaknesses of the IEEE 754 Standard. In addition, they enable more economical design with high efficiency which lowers the cost and the consumed power while providing higher bandwidth and lower latency for memory access.

c. Low Consumed Power.

During the last two decades, further developments of computer architecture and microprocessor hardware have been hitting the so-called "Energy Wall" because of their excessive demands for more energy. Subsequently, we have been ushering in a new era with electric power and temperature as the primary concerns for scalable computing. This is a very difficult and complex problem which requires revolutionary disruptive methods with a stronger integration among hardware features, system software and applications. Equally important are the capabilities for fine-grained spatial and temporal instrumentation, measurement and dynamic optimization, in order to facilitate energy-efficient computing across all layers of current and future computer systems. Moreover, the interplay between power, temperature and performance add another layer of complexity to this already difficult group of challenges.

Existing approaches for energy efficient computing rely heavily on power efficient hardware in isolation which is far from acceptable for the emerging challenges. Furthermore, hardware techniques, like dynamic voltage and frequency scaling, are often limited by their granularity (very coarse power management) or by their scope (a very limited system view). More specifically, recent developments of multi-core processors recognize energy monitoring and tuning as one of the main challenges towards achieving higher performance, given the growing power and temperature constraints. To address these challenges, one needs both suitable energy abstraction and corresponding instrumentation which are amongst the core topics of ongoing research and development work. Another approach is the use of application-specific accelerators to improve the application performance, while reducing the total consumed power which in turn minimizes the overall thermal energy dissipation.

4.4.3. System and Architectures Impact

The "Physical System Simulation" application area urgently needs novel and innovative architectures that provide solutions resolving the 3rd Locality Wall challenge. This includes both novel memory systems and interconnection networks offering much higher bandwidth and lower latency. Energy efficiency indicators also need urgent improvements by at least an order of magnitude. This requirement is equally valid for both homogeneous and heterogeneous architectures (including accelerators and FPGAs) that need further comparisons and analysis. Since the application area of physical system simulations is based predominantly on floating-point arithmetic, novel architecture proposals that address floating-point processing challenges can also be expected to have substantial impact, particularly for dense system computations.

4.5. Optimization

The optimization application area is generally large. We narrow this area to integer-based problems and more specifically to problems that map to the well-known Traveling Salesman Problem (TSP). Of these approaches, there are two different categories: near-optimal techniques (also known as "heuristic techniques"), and (true) optimal techniques. The former is dominated by two classes of algorithms: simulated annealing (SA) and genetic algorithms (GA). (Note that GA is actually a subclass of Evolutionary Algorithms (EA), however it is the most prominent member of EAs.) For true optimal techniques, there are several algorithmic approaches such as dynamic programming (DP), integer linear programming (ILP) and branch-and-bound (BaB). For TSP, each of these approaches can be shown to be NP-hard.

The AB Focus Team found only a few benchmarks that meet the criteria enumerated in Section 1 for true optimal techniques. Thus, for this edition of the roadmap, we focus on near-optimal techniques. For the remainder of these, we assessed several options:

Potential Public Benchmarks

(1) SPEC CPU 2006 [44] 429.mcf: although this solves a vehicle scheduling problem and uses the (accurate, non-NP-hard) network simplex algorithm.

(2) SPEC CPU 2006 [45] 473.astar: performs a heuristic (A* algorithm) routing of a path through a 2D graph.

(3) PARSEC: canneal is a cache-aware simulated annealing (SA) to minimize the routing cost of a chip design.

(4) SPEC CPU 2017 631.deepsjeng_s: alpha-beta search based on Deep Sjeng WC 2008, the 2008 World Computer Speed-Chess Champion

(5) SPEC CPU 2017 641.leela_s: Go with Monte-Carlo-based position estimation, selectrive tree search based on Upper Confidence Bounds, and move valuation based on Elo ratings

(6) SPEC CPU 2017 648.exhange2_s: Sudoku Puzzle Generator

We ruled out options (1) and (3) for different reasons. For option (1), 429.mcf is known not for its computational load but for its unpredictable memory behavior. Since the Big Data application area already addresses the latter and 429.mcf's memory stressload and since we do not believe it is typical of the Optimization application area, we chose to not use 429.mcf. For option (3), we could not find sufficient published results for canneal (or any PARSEC benchmark) on real systems. This is because of the intent of PARSEC: to promote research into parallel systems, not to benchmark existing systems. Thus, we chose 473.astar as one of the benchmark to track for the Optimization application area in this report.

There is not a version of astar included in the latest SPEC CPU2017 benchmark suite. New benchmarks have been added. (4) Deepsjeng, (5) leela, and (6) exchange2 model areas of Artificial Intelligence in the form of alpha-beta tree search (Chess), Monte Carlo tree search (Go), and recursive solution generator (Sudoku), respectively. These benchmarks can be broadly classified as optimization algorithms. In addition to (2) 473.astar, we track performance of (4)–(6) benchmarks. We will continue to track these as more results become available.

4.5.1. Performance Trends and Prediction

In this section, we analyze the performance trend of (1) 473.astar from SPEC CPU 2006, and (2) 631.deepsjeng_s, 641.leela_s, and 648.echange2_s from SPEC CPU 2017, using historical data points uploaded to the SPEC database. This analysis mirrors the analysis of 471.omnetpp above in Section 4.2. We downloaded roughly 9,500 data points of various systems running the workload ranging from April 2006 to December 2017. SPEC has stopped accepting new results in 2018 for this benchmark. We bucketed these data into monthly bins and calculated the maximum and average scores for each bin. SPEC reports the "base" and "peak" scores for each workload. According to SPEC, the base metrics are required for all reported results and have stricter guidelines for compilation. For example, the same flags must be used in the same order for all benchmarks of a given language. The peak metrics are optional and have less strict requirements. For example, different compiler options may be used on each benchmark, and feedback-directed optimization is allowed.

Figure AB-16 shows the performance of 473.astar over time. We plot both base and peak performance with maximum and average scores per monthly bins, represented by the four solid lines. We also plot the linear regression of each metric, represented by the four dotted lines. In general, performance appears to be improving over time. Note that there are occasional steep jumps in performance. As with 471.omnetpp, these generally align with major CPU releases from industry. One thing to note is that the data uploaded to SPEC only has the test date associated with the data, not the system release date. While, in theory, it is possible to look up all the system release dates for all 9,500 data points, we assumed that, in general, newer systems would be more popular to benchmark at any given time, and, thus, the test dates would align well with the system release dates.

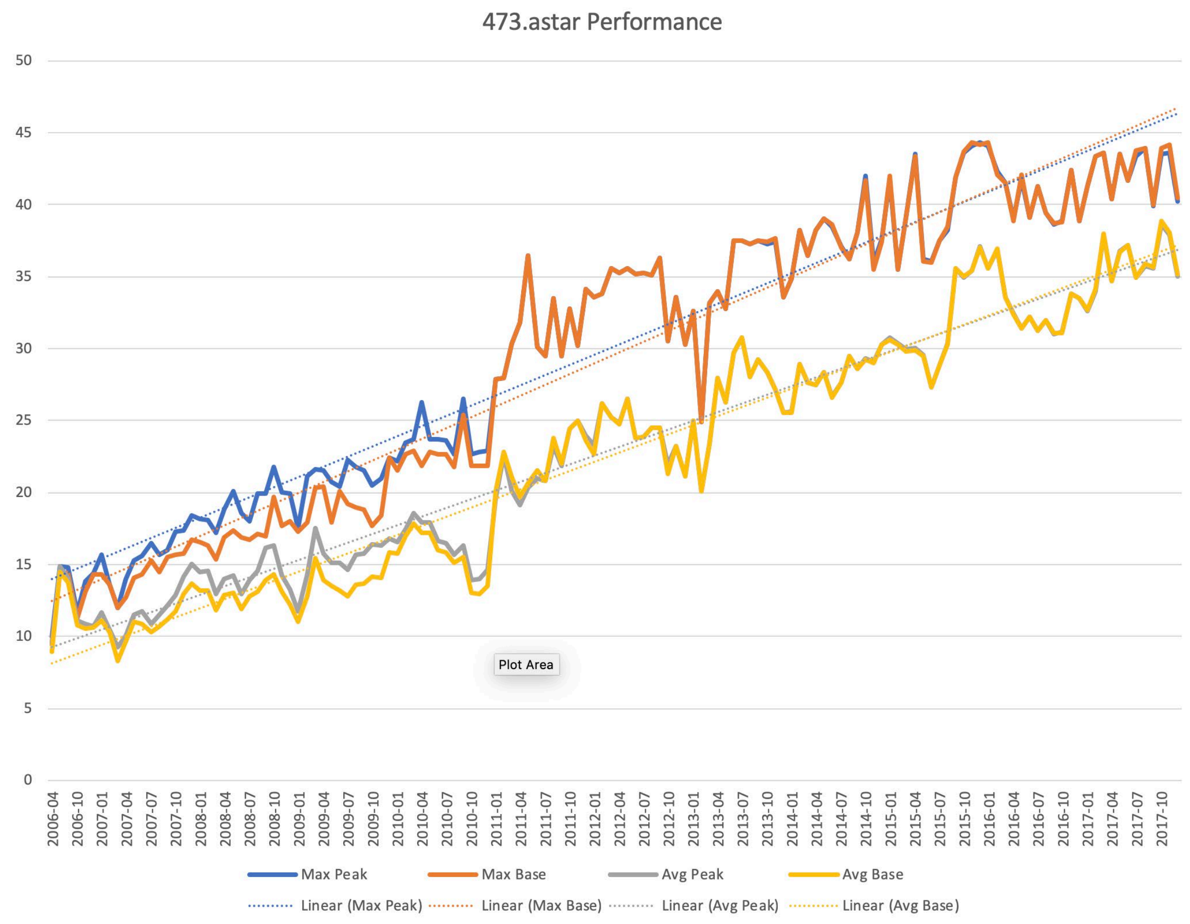


Figure AB-16 Historical Performance of 473.astar Over Time

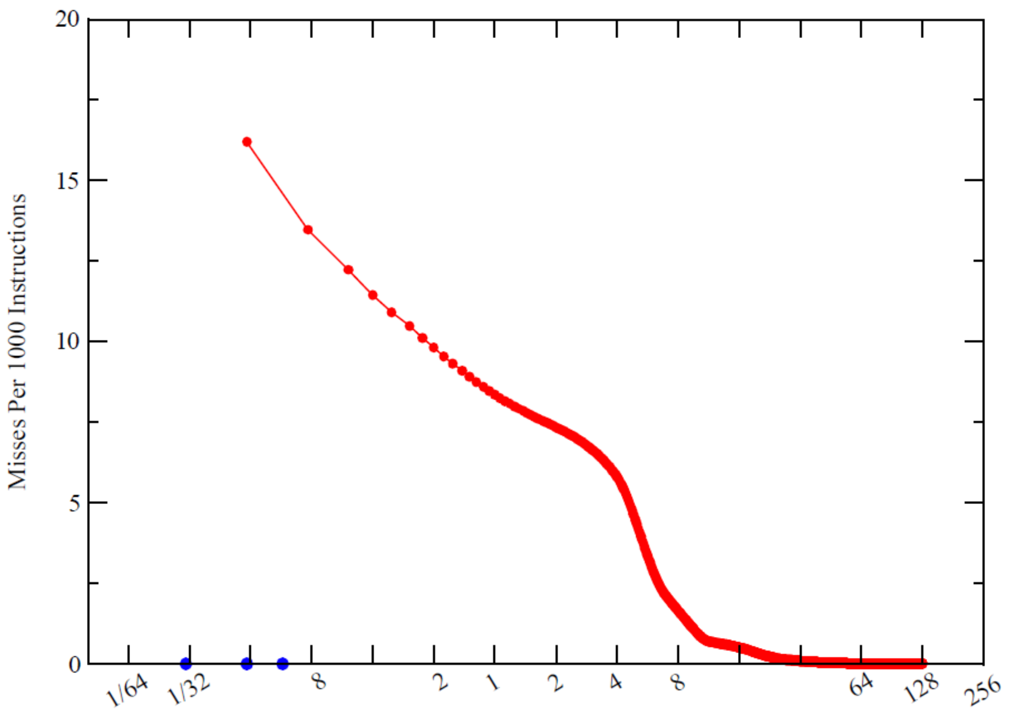


Figure AB-17 Working Set Analysis of 473.astar is Approximately 16–20 MB

Figure AB-17 shows the working set analysis of 473.astar from[46]. Once the data cache size reaches between 16 and 20 MB, the working set almost completely fits in the cache and the misses per kilo-instructions (MPKI) reaches zero. As with omnetpp, 473.astar is also notorious for having an irregular memory access pattern and thus being hard to prefetch.

474.astar has been deprecated with SPEC CPU 2006 in favor of the newer SPEC CPU 2017 suite. However, unlike OMNeT++, which lived on with a newer version in SPEC CPU 2017, astar has been removed from the SPEC CPU suite. Therefore, we track the three new benchmarks, 631.deepsjeng_s, 641.leela_s, and 648.exchange2_s, that represent optimization algorithms going forward.

Figure AB-18 shows the Top-Down analysis [45] of the SPEC CPU 2017 sing-threaded speed benchmarks conducted on an Intel Core i7-8700K-based system taken from [45]. Top-Down breaks down the execution into cycles where useful instructions are retiring, cycles wasted due to bad speculation (branch misprediction), or cycles wasted due to being front-end or back-end bound. Back-end bound cycles can further be broken into memory-bound or core-bound. We can observe that the highlighted optimization workloads have higher front-end and branch-prediction pressure compared to other workloads. 631.deepsjeng_s is relatively more memory bound than the other two optimization workloads.

Figure AB-19 shows the performance trends of the optimization workloads based on roughly 3,000 data points from spec.org ranging from October 2017 through September 2019. Figure AB-20 shows the combined trend of these workloads represented by the geometric mean of the scores for these workloads. The history is not long enough to exhibit a meaningful trend yet, but we will continue to follow the trend going forward.

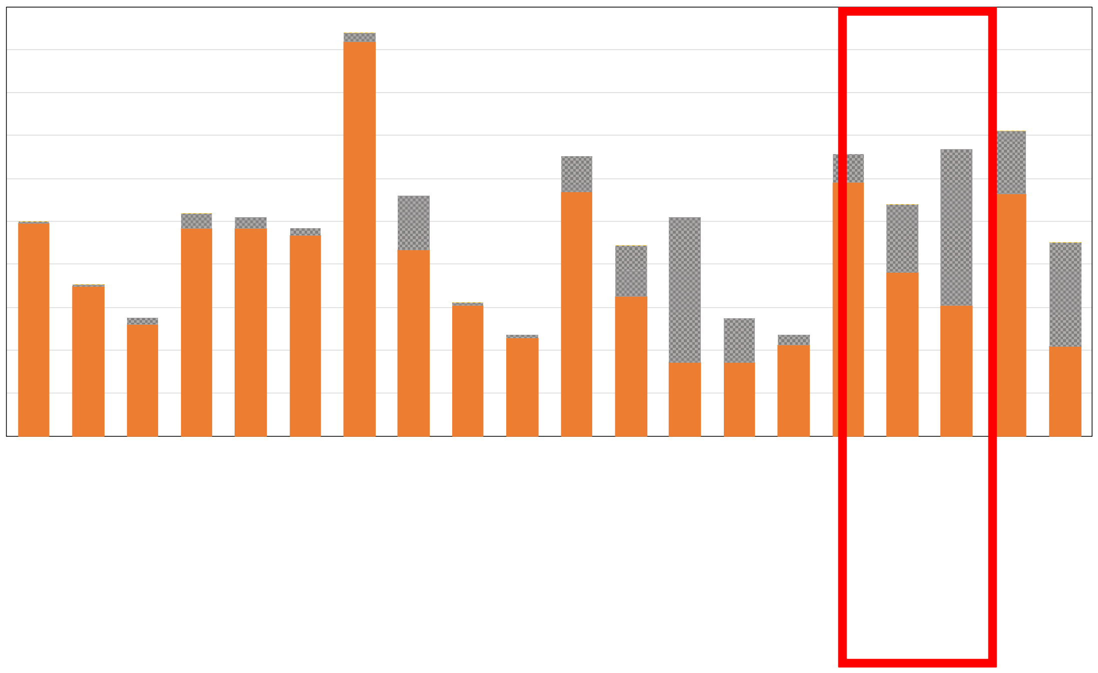


Figure AB-18 Top-down analysis of SPEC CPU 2017 benchmarks on an Intel Core i7-8700K-based system

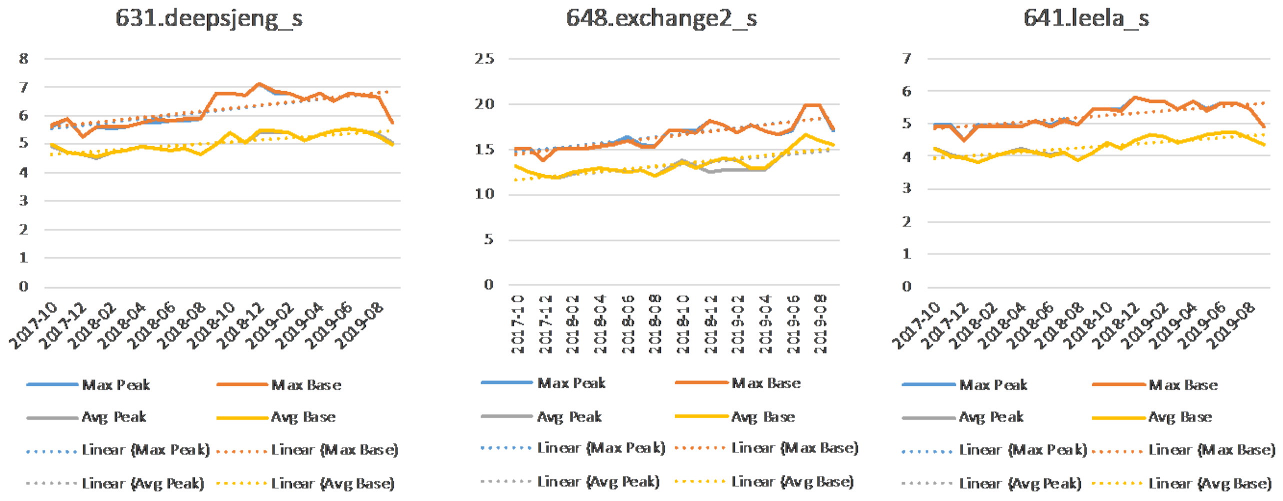


Figure AB-19 Performance trends of optimization workloads from SPEC CPU 2017

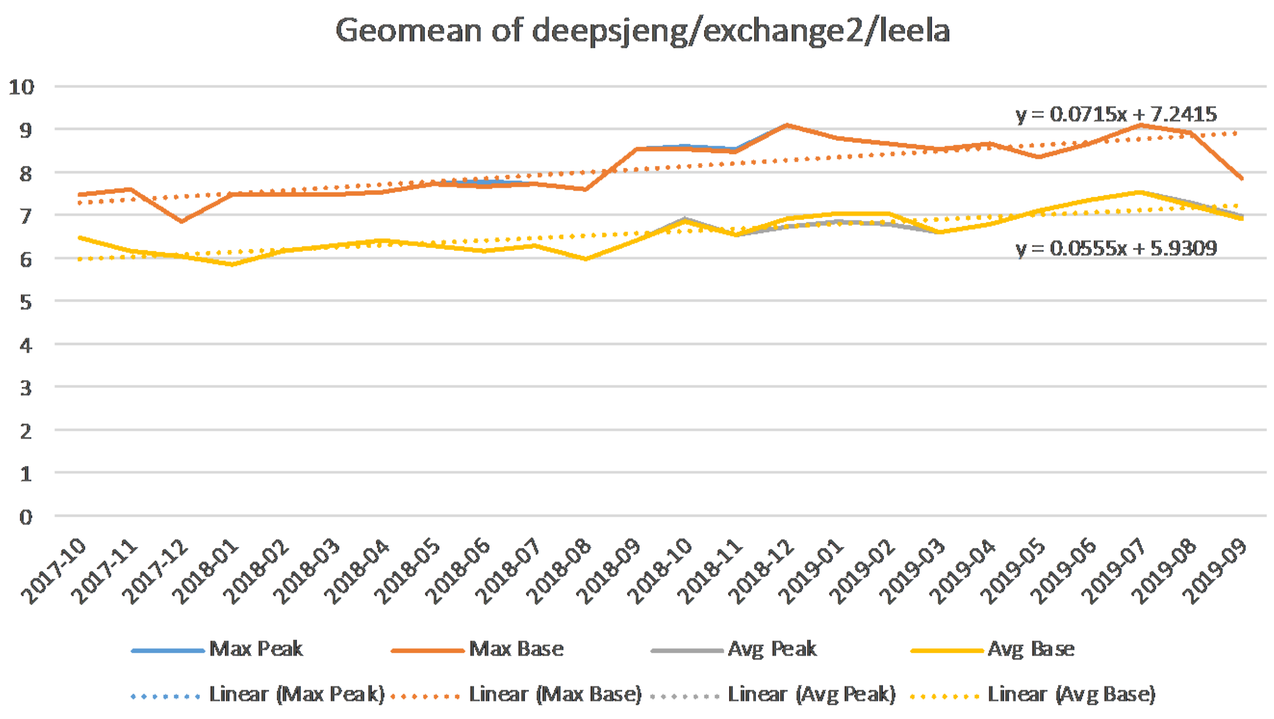


Figure AB-20 Combined trend of optimization represented by the geomean of 631.deepsjeng_s, 648.exchange2_s, and 641.leela_s

4.5.2. TECHNOLOGY NEEDS

In general, near-optimal search should be a compute-bound problem. However, and especially in the case of 473.astar, the data used for the optimization (in this case a topological map) is large and very susceptible to cache and memory bandwidth and latency. Thus, for larger optimization problems, the memory subsystem also becomes critical.

Analyses from 631.deepsjeng_s, 648.exchange2_s, and 641.leela_s show that the optimization workloads also have higher front-end pressure and higher branch misprediction rate that other workloads. Most optimization workloads will benefit from better branch prediction mechanisms. However, many of these workloads have many data-dependent branches that are notoriously difficult to predict using conventional branch prediction schemes based on branch history. Novel branch prediction schemes that help with these hard-to-predict branches.

4.5.3. SYSTEM AND ARCHITECTURES IMPACT

Optimization in general is an integer core processing problem. When the space being optimized is large, it is constrained by memory. Thus, large HPC-oriented nodes perform best on these problems. We would expect that optimization improves with the improvement in raw compute ability per core.

4.6. GRAPHICS/VR/AR

The AB FT decided that because the best available benchmark that fits the criteria enumerated in Section 1 was the SPECviewperf12 benchmark from SPEC in the 2017 edition. However, SPEC has retired SPECviewperf12 in favor of SPECviewperf13. The latter version of SPECviewperf uses the same input sets, although several of the sets have been expanded. We have chosen to apply the data from both SPECviewperf versions in our analysis.

SPECviewperf is described as:

> *The benchmark measures the 3D graphics performance of systems running under the OpenGL and Direct X application programming interfaces. The benchmark's workloads, called viewsets, represent graphics content and behavior from actual applications[47].*

There are four versions of SPECviewperf 12/13, versions 12.01, 12.02, 12.1 and 13.0. The version 12 variations relate generally to minor changes and the results across these versions are comparable. One significant change concerns the viewsets. There are nine viewsets: 3dsmax-05, catia-04, creo-01, energy-01, maya-04, medical-01, showcase-01, snx-02 and sw-03. Of these, 3dsmax-05 was added in SPECviewperf 12.1 in 2016. As such, it has a shorter history than the others. We chose to exclude it from our analysis for this reason. However, once a longer history becomes available, we will re-examine this decision.

4.6.1. Performance Trends and Prediction

The SPECviewperf 12 results for viewsets catia-04, creo-01, energy-01, maya-04, medical-01, showcase-01, snx-02 and sw-03 were combined via geometric mean per month and per year, and are shown below in Figure AB-21.

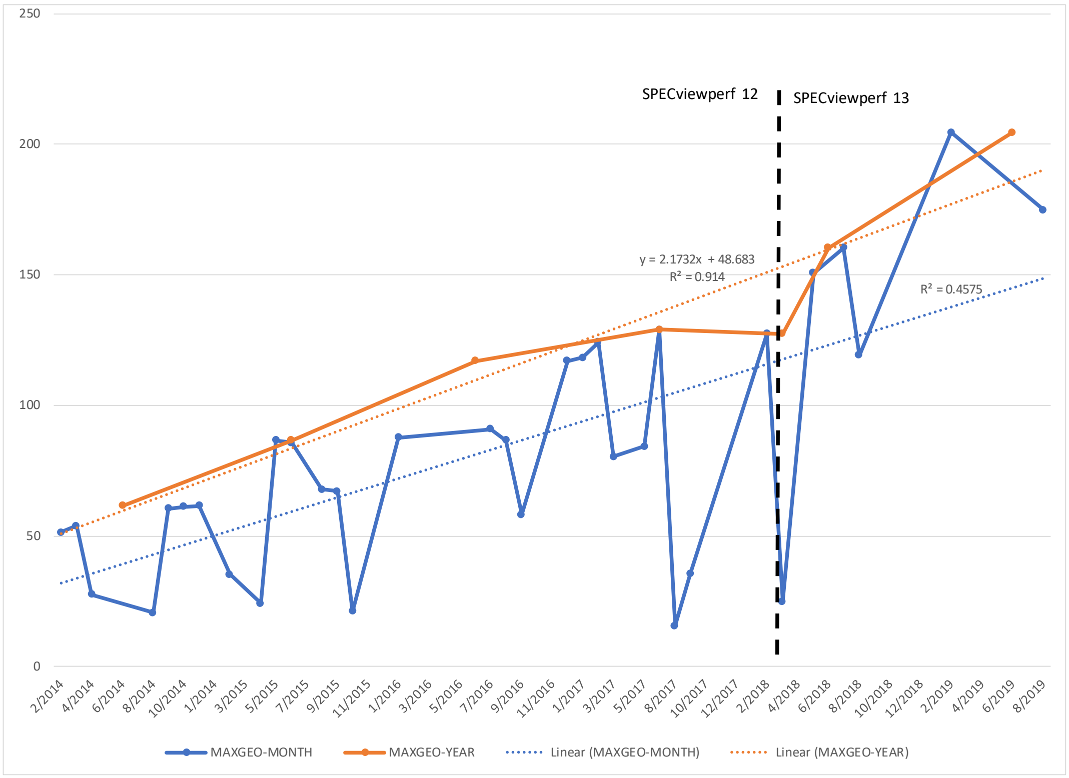


Figure AB-21 Geometric Mean Across Viewsets vs. Time for SPECviewperf12 and SPECview13 Benchmarks

Shown in Figure AB-21 is (a) the maximum of the geometric mean for results reported in a given month (MAXGEO-MONTH), and (b) the maximum of the geometric means for results reported in a given year (MAXGEO-YEAR). The SPECviewperf 13 results are included beginning May 2018 (as indicated by the dotted line). Also shown are R-squared values for the two linear trend lines.

There is much noise in the *MAXGEO-MONTH* data due to the differences in system classes under test. The trendline for max geomean shows a rate of change of 2.17 per month for the yearly trend line which is the less noisy of the two with the higher R-squared value. It is unclear whether SPECviewperf13 will have comparable performance to SPECview12 or not. We will address this in the next roadmap update.

4.6.2. Technology Needs

Graphics performance depends on many variables such as software framework implementation, algorithm advances, and hardware advances, and SPECviewperf12 and 13 include all of these. However, a closer inspection of the results shows that there is a correlation between the introduction of new GPU architectures and an increase in results. Generally, GPU architectures improve in two dimensions: the amount of parallelism possible in hardware and the memory bandwidth provided. The former of these two saturates as overall parallelism, although high for graphics applications, is not infinite. The latter is the driver: the higher the memory bandwidth, the better the performance of Graphics/VR/AR workloads.

4.6.3. SYSTEM AND ARCHITECTURES IMPACT

We believe the best architectures for Graphics/VR/AR workloads are highly parallel and have high bandwidth to memory. Because the parallelism levels are high, it is memory bandwidth more than memory access latency that dominates this applications area. Emerging near-memory-processing architectures achieve both high bandwidth and, for well partitioned data, low access latency. Note that existing GPUs grew out of fixed-function accelerators optimized for the (now retired) fixed graphics pipeline. For this reason, this workload is also receptive to fixed-function acceleration.

4.7. CRYPTOGRAPHIC CODEC

As a preliminary matter, the AB IFT has not yet found enough empirical data to establish a trend for this application area. However, this section outlines the procedures for establishing a trend once the data is available.

Cryptographic acceleration focuses on improving the performance and energy-efficiency of cryptographic *primitives*, which are the building blocks used to create more sophisticated *cryptosystems*. These cryptosystems are at the foundation of everything from web commerce and wired homes, to secure communications and cryptocurrency. Decades of research spent evolving various strategies for securing the internet has today yielded a robust set of primitives, the most ubiquitous being:

- **SHA** - Secure Hash Algorithm, designed by the NSA, which takes an arbitrarily long sequence of bytes and generates a fixed-size digest with a property of being essentially unique (extremely low probability of non-uniqueness)
- **AES** - Advanced Encryption Standard, a NIST-standard symmetric-block cipher, with various authenticated encryption modes (CCM, GMC, GMAC)
- **RSA** - Rivest-Shamir-Adleman, public-key encryption based on modulo logarithms and exponentiation
- **ECC** - Elliptic-curve Cryptography, public-key encryption based on the algebraic structure of elliptic curves
- **DH** - Diffie-Hellman key exchange, a method for secretly exchanging keys

All of these primitives utilize integer math exclusively, making them prime candidates for hardware acceleration. SHA and AES are algorithms that either accept or generate data with fixed-sized blocks, whereas public-key infrastructures (PKI) perform calculations on arbitrary-sized integers. PKIs require basic arithmetic functionality as well as exponentiation and modulus math. Unlike SHA and AES, the integers used in PKI often have hundreds of digits, so most compute time is spent inside what is known as "big-integer" manipulation code (a.k.a., "*bigint*").

One primitive that doesn't quite fit in the list above is True Random Number Generators (TRNGs). PKIs have demonstrable vulnerabilities to incorrect or broken TRNG implementations. As a result, it is increasingly common to find NIST- and even BSI-compliant hardware on-die.

4.7.1. SYSTEM AND ARCHITECTURE IMPACT

Hardware acceleration for cryptographic primitives is available either through CPU instruction-set extensions, or off-silicon hardware accessed as a peripheral using *direct-memory access* (DMA) or *memory-mapped I/O* (MMIO). AES and SHA are easy targets for specific algorithmic optimization, compared to modulo or elliptic-curved based math which due to its procedural nature benefits from generic *big-integer arithmetic* acceleration (e.g. multiplication, division, exponentiation). Big-integer math operations—like multiplication, division and exponentiation—utilize arbitrary-sized integer inputs which can be hundreds of digits long.

Intel natively supports AES-NI[4] and SHA[5] extensions for AES and SHA1/SHA256 respectively[6] in its current processors. Similarly, Arm provides several dozen instructions for supporting AES, SHA1, SHA256, SHA512, SHA3, SM3 and SM4.[7] Other companies also provide their own acceleration including Arm-based Silicon Labs processors and Synopsys' ARC architecture.

[4] *https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni*
[5] *https://software.intel.com/en-us/articles/intel-sha-extensions*
[6] *"Intel® 64 and IA-32 Architectures, Software Developer's Manual, Volume 2 (2A, 2B, 2C & 2D): Instruction Set Reference, A-Z"*
[7] *"Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile", Section C3.5.27, "The Cryptographic Extension", page C3-247, Copyright 2013-2019*

4.7.2. BENCHMARKS

Benchmarks for assessing the performance and energy efficiency of these primitives are still limited as of 2019. EEMBC's SecureMark-TLS benchmark[8] measures both the performance and energy cost of all of the aforementioned primitives, except for RSA. Being geared toward IoT and low-power, the benchmark's final score is the inverse of the weighted energy cost (in Joules) associated with the operation. The selection of primitives, the number of contexts, and the data size are all selected to emulate the TLS v1.2 handshake with the "TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256" cipher-suite. There are currently two scores on the public website, one with acceleration on a Cortex-M33 and one without on a Cortex-M4; the scores are roughly one order of magnitude apart. As scores accumulate, this section will be updated with trend information (in the update to this edition of the roadmap).

5. CROSS MATRIX

The function of the Cross Matrix is to map application areas to SA-IFT-defined market drivers. In the May 2016 meeting in Leuven, Belgium, the initial market drivers were determined. In the December 2016 IRDS meeting, the final market drivers for the 2017 Roadmap were decided on. We continue with these in the current edition of the roadmap:

- **Internet-of-Things edge devices (IoT-e):** Although IoT is a broad class if computing applications spanning the server to the ultimate sensors and actuators, an IoT edge (IoTe) device is a wireless device with computation, sensing, communication, and possibly storage. The device may include one or more CPUs, memory, non-volatile storage, communication, security, and power management. It may be line powered, battery powered or utilize energy harvesting.
- **Cloud computing (CC):** This category is for server devices deployed in data centers. The term "cloud" refers to the engineering of data center scale computing operations: compute, storage, networking engineered for scale and for continuous resource redeployment and reconfiguration via APIs. Whether they are operated publicly or privately, they offer on demand, as-a-service consumption model. While they had their origins in web service; media streaming, shopping and commerce; they are increasingly broadening their applications base to big data for social networking, recommendations, and other purposes; precision medicine; training of AI systems, and high performance computation (HPC) for science and industry.
- **Cyber-Physical Systems (CPS):** This category encompasses computer-based control of physical devices characterized by real-time processing and used primarily in industrial control. Many cyber-physical systems are safety-critical. They interface to the systems they control via both standard and proprietary interconnects broadly know as Operational Technology (OT), where ruggedness, extended environmental capabilities, low cost have been paramount over considerations such as security and attestation.
- **Mobile (Mo):** This category encompasses devices that integrate computation, communication, storage, capture and display, and sensing. Mobile systems are highly constrained in both form factor and energy consumption. As a result, their internal architectures tend to be heterogeneous. Cores in modern mobile units include: mult-size multicore CPUs, GPUs, video encode and decode, speech processing, position and navigation, sensor processing, display processing, computer vision, deep learning, storage, security, and power and thermal management.

Below in Table AB-7 is the cross matrix that was developed, "X" means important application area(s), and "E" means important *and* energy-constrained application area(s).

Table AB-7 Cross Matrix of Application Area vs. Market Drivers

Application Area	IoT-e	CC	CPS	Mo
Big Data Analytics	E	X		
Artificial Intelligence	E	X	E	E
Discrete Event Simulation		X		
Physical System Simulation		X		
Optimization		X	E	
Graphics/VR/AR		X		E
Cryptographic codec	E	X	E	E

[8] *https://www.eembc.org/securemark*

6. CONCLUSIONS AND RECOMMENDATIONS

Applications are the drivers of much of the nanoelectronics industry. The challenge is how to connect the trends in applications to the nanoelectronics trends and needs. This chapter of the IRDS has sought to do this by extrapolating from representative benchmark programs for each application area to performance trends over time and critical technology needs. Table AB-2 shows a summary of the findings of the AB IFT. Memory bandwidth increases are critical for all of the application areas we studied. Not all of the application areas benefit from improvement in memory latency, however. This suggests that logic in memory is not a universal solution to meeting all applications. Rather, it is most important for simulation and optimization application areas.

Another emerging trend is to use fixed-function accelerators to speed up critical applications. This improves the power efficiency of microprocessors because it obviates the need to continuously fetch and decide instructions. For example, a common application area is discrete cosine transform, or DCT. A fixed-function accelerator for DCT has the advantage over a software implementation in that it does not require the fetching and decoding of the software instructions. Two key application areas are predicted to benefit most from fixed-function acceleration: Feature Recognition, Graphics/VR/AR and Cryptographic Codec.

The process of applications benchmarking is imperfect in several senses. The AB IFT makes the following recommendations for improving the process of applications benchmarking itself. (1) The benchmarks that are being used today must continue to be used into the future. In addition, (2) investment in new benchmark development is critically important for several of the application domains studied, including Feature Recognition, Discrete Event Simulation, and Optimization. In addition to those application areas, the AB IFT sees the need for investment in benchmarks to address cryptography and security and to address multimedia/DSP workloads. Both (1) and (2) are standards activities that the IEEE Standards Association could tackle with new working groups.

7. References

References for Table AB-3 and Table AB-4:

[1] www.eetimes.com/document.asp?doc_id=1317674

[2] www.nextplatform.com/2017/03/21/can-fpgas-beat-gpus-accelerating-next-generation-deep-learning/

[3] arxiv.org/abs/1704.04760

[4] wccftech.com/nvidia-volta-gv100-gpu-tesla-v100-architecture-specifications-deep-dive/

[5] images.nvidia.com/content/tesla/pdf/nvidia-tesla-m4-datasheet.pdf

[6] www.anandtech.com/show/10433/nvidia-announces-pci-express-tesla-p100

[7] images.nvidia.com/content/pdf/tesla/184457-Tesla-P4-Datasheet-NV-Final-Letter-Web.pdf

[8] images.nvidia.com/content/pdf/tesla/184427-Tesla-P40-Datasheet-NV-Final-Letter-Web.pdf

[9] en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units

[10] www.extremetech.com/computing/240908-amd-launches-new-radeon-instinct-gpus-tackle-deep-learning-artificial-intelligence

[11] hothardware.com/news/amd-radeon-rx-vega-unveiled-13-tflops-16gb-hbm2-4k60

[12] www.nextplatform.com/2017/05/22/hood-googles-tpu2-machine-learning-clusters/

[13] syncedreview.com/2017/04/15/what-does-it-take-for-intel-to-seize-the-ai-market/

[14] www.tweaktown.com/news/55875/amd-launch-monster-navi-10-2019-next-gen-ram/index.html

[15] wccftech.com/nvidia-xavier-soc-tegra-volta-gpu-announced/

[16] images.nvidia.com/content/tesla/pdf/nvidia-tesla-m4-datasheet.pdf

[17] en.community.dell.com/techcenter/high-performance-computing/b/general_hpc/archive/2017/03/22/deep-learning-inference-on-p40-gpus

[1] http://graph500.org/

[2] http://green.graph500.org

[3] Lifeng Nai, Yinglong Xia, Ilie G. Tanase, Hyesoon Kim, and Ching-Yung Lin, "GraphBIG: Understanding Graph Computing in the Context of Industrial Solutions", Supercomputing, 2015.

[4] Vasiliki Kalavri, Vladimir Vlassov, and Seif Haridi, "High-Level Programming Abstractions for Distributed Graph Processing", arXiv:1607.02646v1, Jul 2016.

[5] V. Janapa Reddi, C. Cheng, D. Kanter, P. Mattson, et al., "MLperf Inference Benchmark," https://arxiv.org/abs/1911.02549

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[7] https://www.mckinsey.com/industries/semiconductors/our-insights/artificial-intelligence-hardware-new-opportunities-for-semiconductor-companies

[8] *https://www.microsoft.com/en-us/research/project/project-brainwave/*

[9] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE* **105**(12), 2295-2329 (2017).

[10] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates et al., "In-datacenter performance analysis of a tensor processing unit." In 2017 *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1-12. IEEE, 2017. https://arxiv.org/abs/1704.04760

[11] http://nvdla.org

[12] D. Das Sarma and G. Venkataramanan, "Compute and redundancy solution for Tesla's Full Self driving computer," *Hot Chips 2019*. *www.hotchips.org ;* https://fuse.wikichip.org/news/2707/inside-teslas-neural-processor-in-the-fsd-chip/

[13] https://www.sigarch.org/dnn-accelerator-architecture-simd-or-systolic/

[14] https://www.xilinx.com/support/documentation/white_papers/wp506-ai-engine.pdf

[15] N. Verma, H. Jia, H. Valavi, et al., "In-Memory Computing: Advances and prospects." *IEEE Solid-State Circuits Magazine*, **11**(3), 43-55 (2019).

[16] W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing." *Proceedings of the IEEE* **107**(1), 108-122 (2018).

[17] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision.” In *Proc ICML*, pp. 1737–1746. (2015).

[18] S. Han, H. Mao, and W. J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv:1510.00149 (2015).

[19] https://www.hpcwire.com/2018/05/02/mlperf-will-new-machine-learning-benchmark-help-propel-ai-forward/

[20] http://mlperf.org

[21] https://svail.github.io/DeepBench/

[22] N. Wang, J. Choi, D. Brand, C.-Y. Chen and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *32nd Conference on Neural Information Processing Systems*, 2018.

[23] http://mlperf.org

[24] https://en.wikipedia.org/wiki/Discrete_event_simulation

[25] https://www.spec.org/cpu2006/Docs/471.omnetpp.html

[26] https://omnetpp.org/

[27] http://iss.ices.utexas.edu/?p=projects/galois/benchmarks/discrete_event_simulation

[28] Aamer Jaleel, “Memory Characterization of Workloads Using Instrumentation-Driven Simulation”, http://www.jaleels.org/ajaleel/publications/SPECanalysis.pdf (2010)

[29] http://www.ece.uah.edu/~milenka/docs/ranjan.hebbar.thesis.pdf

[30] V. Getov, e-Science: The Added Value for Modern Discovery, Computer 41(8), 30–31, IEEE Computer Society, 2008.

[31] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, S.K. Weeratunga. The NAS Parallel Benchmarks, Int. J. of Supercomputer Applications 5(3), 63–73, 1991, http://www.davidhbailey.com/dhbpapers/benijsa.pdf.

[32] C. Addison, V. Getov, A. Hey, R. Hockney, I. Wolton. The GENESIS Distributed-Memory Benchmarks. In: J. Dongarra and W. Gentzsch (Eds.) Computer Benchmarks, Advances in Parallel Computing 8, 257–271, Elsevier Science Publishers, 1993.

[33] D.H. Bailey, M. Berry, J. Dongarra, V. Getov, T. Haupt, T. Hey, R.W. Hockney, D. Walker, “PARKBENCH Report-1: Public International Benchmarks for Parallel Computers", TR UT-CS-93-213, Scientific Programming 3(2), 101–146, 1994, http://www.davidhbailey.com/dhbpapers/parkbench.pdf.

[34] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W.B. Jones, B. Parady, SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance, Proc. Int. WOMPAT 2001, LNCS 2104, 1–10, Springer, 2001.

[35] K. Asanovic, R. Bodik, B.C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S. W. Williams, K. A. Yelick, The Landscape of Parallel Computing Research: A View from Berkeley, TR UCB/EECS-2006-183, University of California, Berkeley, Dec. 2006, http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf .

[36] P.M. Kogge, Data Intensive Computing, the 3rd Wall, and the Need for Innovation in Architecture, Argonne Training Program on Extreme-Scale Computing, August 2017, video: https://youtu.be/ut9sBnwF6Kw, slides: http://extremecomputingtraining.anl.gov/files/2017/08/ATPESC_2017_Dinner_Talk_6_8-4_Kogge-Data_Intensive_Computing.pdf

[37] V. Marjanović, J. Gracia, and C. W. Glass, Performance modeling of the HPCG benchmark, Proc. Int. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, pp. 172–192, Springer, 2014.

[38] J. Dongarra, P. Luszczek, A. Petitet, “The LINPACK Benchmark: Past Present and Future”, Concurrency and Computation: Practice and Experience 15(9), 803–820, 2003.

[39] E. Strohmaier, H.W. Meuer, J. Dongarra, H.D. Simon, “The TOP500 List and Progress in High-Performance Computing”, Computer 48(11), 42–49, IEEE Computer Society, 2015.

[40] J. Dongarra, M.A. Heroux, P. Luszczek “High-Performance Conjugate-Gradient Benchmark: a New Metric for Ranking High Performance Computing Systems”, Int. J. of High Performance Computing Applications 30(1), 3–10, 2016.

[41] IEEE 754-2019 – IEEE Standard for Floating-Point Arithmetic, IEEE xPlore, July 2019.

[42] A. Haidar, P. Wu, S. Tomov, J. Dongarra, Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers, Proc.ScalA Workshop at SC’17, pp. 1–8, ACM, 2017.

[43] J.L. Gustafson, I.T. Yonemoto, “Beating Floating Point at its Own Game: Posit Arithmetic”, Supercomputing Frontiers and Innovations 4(2), 71–86, 2017.

[44] http://www.spec.org/cpu2006/Docs/429.mcf.html

[45] http://www.spec.org/cpu2006/Docs/473.astar.html

[46] Aamer Jaleel, “Memory Characterization of Workloads Using Instrumentation-Driven Simulation”, http://www.jaleels.org/ajaleel/publications/SPECanalysis.pdf (2010)

47 http://spec.org/gwpg/gpc.static/vp12.1info.html