IEEE Copyright Notice

Accepted to be Published in: Proceedings of the 35rd IEEE International Parallel & Distributed Processing Symposium, May 17-21, 2021 Portland, Oregon, USA

Matrix Engines for High Performance Computing: A Paragon of Performance or Grasping at Straws?

Jens Domke*§, Emil Vatai*§, Aleksandr Drozd*§, Peng Chen†, Yosuke Oyama§, Lingqi Zhang§, Shweta Salaria*§, Daichi Mukunoki*, Artur Podobas‡, Mohamed Wahib†*, Satoshi Matsuoka*§
* RIKEN CCS, Japan {jens.domke,emil.vatai,aleksandr.drozd,shweta.salaria,daichi.mukunoki}@riken.jp
† National Institute of Advanced Industrial Science and Technology, Japan {mohamed.attia,chin.hou}@aist.go.jp
‡ KTH Royal Institute of Technology, Stockholm, Sweden podobas@kth.se
§ Tokyo Institute of Technology, Japan {oyama.y.aa,zhang.l.ai}@m.titech.ac.jp

***Abstract*—Matrix engines or units, in different forms and affinities, are becoming a reality in modern processors; CPUs and otherwise. The current and dominant algorithmic approach to Deep Learning merits the commercial investments in these units, and deduced from the No. 1 benchmark in supercomputing, namely High Performance Linpack, one would expect an awakened enthusiasm by the HPC community, too.**

Hence, our goal is to identify the practical added benefits for HPC and machine learning applications by having access to matrix engines. For this purpose, we perform an in-depth survey of software stacks, proxy applications and benchmarks, and historical batch job records. We provide a cost-benefit analysis of matrix engines, both asymptotically and in conjunction with state-of-the-art processors. While our empirical data will temper the enthusiasm, we also outline opportunities to "misuse" these dense matrix-multiplication engines if they come for free.

I. Introduction

With both Dennard's scaling [1] and Moore's law [2] gone, computer scientists and architects are perhaps facing their grandest challenge to date. Today, computer scientists are actively chasing Post-Moore alternatives such as the intrusive neuromorphic and quantum computers [3]. However, not all options need to be intrusive, and some merely require us to move away from traditional von-Neumann architectures. Among the more salient of these options is architectural specialization [4]. Hardware specialization focuses on accelerating application-specific core components to reduce the needless energy tax [5] that a traditional von-Neumann general-purpose system demands. Instead, the aspiration is to maximize data locality and fully eliminate the operation control cost that is continuously present in GPUs (e.g., instruction fetching and decoding, etc.) [6]. Architecture-specialization is not a new concept in itself, where co-processors and accelerators based on Field-Programmable Gate Arrays (FPGAs) [7], [8], Coarse-Grained Reconfigurable Architectures (CGRAs) [9], or Application-Specific Integrated Circuits (ASICs) (e.g., Anton [10] or Grape [11]) have continuously accompanied computer systems in their historical road to performance.

Among the more popular candidates for architecture specialization, much thanks to the limitless popularity of Deep-Learning [12], is to target General Matrix Multiplication (GEMM). Targeting GEMM is perhaps not entirely unmotivated: GEMM is often *claimed* to be the core compute-intensive component in many scientific applications spanning multiple domains, such as Computational Fluid Dynamics (e.g., NEK5000 [13]) or Deep Learning (DL) [14]. Today, there are already a large bulk of *application-specific* (primarily DL) accelerators that are based around systolic arrays [15] (essentially GEMM engines), such as Huawei's Ascend 910 [16] and Google Tensor Processing Units (TPUs) [17].

More importantly, the trend of adopting hardware acceleration for GEMM operations is coming even to *general-purpose architectures* and their Instruction Set Architecture (ISA). NVIDIA introduced the Tensor Cores [18] in the Volta, Ampere, and Turing series of accelerators. Both Intel (with Sapphire Rapids [19]) and IBM (with Power10 [20]) are extending their SIMD-capabilities to support matrix operations, with similar proposals by authors dating back a decade [21]. The unspoken question is: *Is the inclusion of specialized matrix engines in general-purpose processors truly motivated and merited, or is the silicon better invested in other parts?*

In this paper, we aspire to holistically look at the inclusion of matrix engines—abbreviated **ME** hereafter—into the general-purpose processor and its expected impact on High-Performance Computing (HPC) applications. It is important to emphasize that we consider DL as one of many workloads in HPC, and not the application we solely focus on. In this study, we target to answer the following three questions:

- Does the occurrence and usage of matrix operations in scientific workloads truly merit matrix engines' inclusion into general-purpose ISAs?
- What performance benefits can we expect from using MEs on existing scientific applications that can leverage them?
- Performance projection of using matrix engines on future scientific workloads using a model empirically derived from the NVIDIA V100 GPUs.

To answer the above questions, our contributions are:

- We inspect software management packages, historical batch job records, profiles, and source code of a board set of HPC and Machine Learning proxy applications and benchmarks to identify dense matrix requirements.
- We provide a cost-benefit analysis of projected performance gains from matrix engines, driven by resource usage per domain in different production supercomputers.
- A detailed discussion of opportunities and challenges in adopting matrix engines, from the perspective of HPC workloads.

TABLE I: Overview of existing and emerging general-purpose and AI architectures that leverage matrix engines to accelerate computations (**f16** = IEEE-754 16-bit or BFloat16, **f32** = IEEE-754 single prec., **f64** = IEEE-754 double prec.; **GF** = Gflop/s; INT4/8 support omitted)

Type	System	Tech.	Die size	ME size	Tflop/s (f16)	Tflop/s (f32)	Tflop/s (f64)	Support
General	Intel Sapphire Rapids[1]	10 nm	—	16x32	—	—	—	f16
	IBM Power10 [2]	7 nm	602 mm^2	4x4	16.4 (27.2 GF/mm^2)	8.2 (13.6 GF/mm^2)	4.1 (6.8 GF/mm^2)	f16, f32, f64
	NVIDIA Tesla V100	12 nm	815 mm^2	4x4x4	125.0 (153.4 GF/mm^2)	15.7 (19.3 GF/mm^2)	7.8 (9.6 GF/mm^2)	f16
	NVIDIA Tesla A100[3]	7 nm	826 mm^2	4x4x4	312.0 (377.7 GF/mm^2)	19.5 (23.6 GF/mm^2)	19.5 (23.6 GF/mm^2)	f16, f32, f64
AI	Google TPUv2	20 nm	—	128x128	45.0 (—)	—	—	f16
	Google TPUv3	16 nm	—	128x128	90.0 (—)	—	—	f16
	Habana Labs Gaudi	16 nm	500 mm^2	Shared	—	—	—	f16, f32
	Huawei Ascend 910[4]	7 nm	1228 mm^2	16x16x16	256.0 (208.5 GF/mm^2)	—	—	f16

II. Matrix Engines from a Hardware Perspective

In this section, we describe historical and current developments of ME adoption by vendors. The section is structured as follows: first, we discuss the historical lead up to MEs, then elaborate on current trends in hardware, and finally, we highlight the HPC community's motivation for leveraging MEs.

A. Matrix Engines Lineage

Early processors were scalar: executing a single operation on pairs of operands one at a time, possibly using Out-of-Order (OoO) and superscalar execution to improve performance via increased Instruction Level Parallelism (ILP) [22]. As time moved on, Moore's law facilitated the inclusion of more complex BLAS-1 (Vector-Vector) operations in hardware through the use of Single Instruction Multiple Data (SIMD) [23] units. These additions, refined in systems such as ARM A64FX's Scalable Vector Extensions (SVE) [24], Intel's AVX512 [25], or abstracted away through threading in NVIDIA's CUDA [26], did increase the performance of processors at the cost of a small area of silicon; area that was available due to Moore's law. Moving (or upgrading) these SIMD units to support GEMM operations is only the next natural step, with one small caveat: Moore's law is approaching its end [27]. The obvious question becomes: *Are MEs really what we should be spending our silicon on, given that Moore's law is about to die out?*

B. Matrix Engines: State-of-the-Art Performance and Trends

We compile Table I to highlight some well known commercial general-purpose systems or AI accelerators that already exist or are about to be released (for the plethora of academic AI accelerators, we refer interested readers to surveys on the subject [28], [9], [29]). All these new systems contain some form of ME integrated into the architecture. From the general-purpose side, both the upcoming Intel Sapphire Rapids and the IBM Power10 will be augmented with MEs. According to public documents, IBM Power10 [20] will be the more general CPU of the two, with support for a large variety of numerical representations (Int[4|8|16], FP[16|32|64]), while the Intel's AMX unit will focus more on AI applications (bfloat16), based on the available information in the programming reference [19]. IBM Power10 features a hybrid model, which means that it accumulates into a wider representation than what it multiplies in; the only exception is for FP64, where it both multiplies and accumulates into the same representation.

From the GPU side, both NVIDIA V100 [18] and A100 feature MEs, called Tensor Cores, abbreviated TCs hereafter. The MEs in the V100 are hybrid, meaning that they accumulate into a wider numerical representation (in V100's case: FP32) than what it multiplies with (FP16). This limitation was overcome in A100, which supports up to double-precision (FP64) in its MEs. From a performance perspective, the GPU-based systems—in particular the new A100—are grossly outperforming the other systems in both a higher peak performance (Tflop/s) and a higher compute density ($Gflop/s / mm^2$), where the IBM Power10 only reaches 18% of the compute-density of an NVIDIA V100. The NVIDIA A100 is reported to reach up to 312 Tflop/s of half-precision (FP16) performance.

The available information for AI-accelerators is more sparse and most architectures primarily report peak performance. Furthermore, most focus exclusively on bfloat16 (same as Intel Sapphire Rapids), and overall show a higher performance over general-purpose CPUs. The Habana Labs Gaudi [30] architecture uses a shared ME unit accessible to all the cores, but most of the architectural details (including performance) are undisclosed. The highest performance and compute density of the AI accelerators delivers the Huawei's Ascend 910 [16], which reaches 256 Tflop/s of raw FP16 performance, leading to 208 $Gflop/s / mm^2$ of compute density—nearly an order of magnitude (7.7x) more than IBM Power10 and but still only 55% of the NVIDIA A100's peak performance.

To summarize: modern architectures are moving towards integration of MEs into the core fabric. Specializing the silicon towards acceleration of GEMM can be worthwhile, and can yield substantial increase in performance and compute densities (with orders of magnitude better performance see e.g., NVIDIA A100 vs. P100 in FP16; the latter with 18.7 Tflop/s peak). This could, however, come at a cost and a potential loss in generality, which may be counter-productive from a HPC perspective.

C. Matrix Engines: Motivation from HPC Side

The consideration for inclusion of MEs into future systems can be motivated by improvements in performance and energy efficiency by such a transition. Energy-consumption is one of the two limiting factors [31] (the other is cost) when building supercomputers, and any architectural choices that reduce energy (without sacrificing performance) will directly contribute to better systems. Furthermore, we can measure the impact

[1] AMX listed for completeness (16x64 ME for INT8); performance unknown.

[2] Performance is calculated assuming 16 SMT8 cores running at 4 GHz.

[3] The A100 also offers a unique hybrid 19-bit TF32 format combining a 10-bit mantissa of IEEE-754 half-precision and 8-bit exponent of BFloat16, yielding up to 156 Tflop/s with TCs.

[4] Die size includes the Nimbus co-accelerator and four HBM2 stacks.

TABLE II: Energy-eff. of Vector Extensions on a Intel Xeon CPU

Precision	Vector extension	Walltime	Energy-efficiency
DGEMM	—	34.22 s	1.23 Gflop/J
	AVX2	12.49 s	2.92 Gflop/J
SGEMM	—	16.79 s	2.65 Gflop/J
	AVX2	6.36 s	5.92 Gflop/J

of past architectural design-choices on energy consumption; choices that are similar to those we are looking at today.

Consider, for example, exercising the Intel Xeon E5-2650v4 processor, which supports both scalar and vector instructions (AVX + AVX2) with a GEMM operation. The energy-efficiency (in $^{\text{flop}}/_{\text{J}}$) of this GEMM operation differs between the scalar and vectorized version and yields an average 2.3x observed increase in energy-efficiency, in favor of the vectorized version. To measure this, we use OpenBLAS [32], compiled with and without AVX support. OpenBLAS GEMM calls (for both single and double precision) are invoked with square matrices of size $n = 5000$, and we repeat the call 30 times; resulting in a total of $2 \cdot n^3 = 7.5\,\text{Tflop}$. The energy consumption is measured using Intel Performance Counter Monitor (PCM) and we list the results in Table II. The similar results for both DGEMM and SGEMM suggests that this increase in efficiency is agnostic to the numerical precision for this particular experiment.

Another example compares the energy efficiency of modern GPUs (in this case, an NVIDIA V100), which contains mixed-precision MEs. Executing a GEMM operation on such a processor, with and without using the built-in MEs, will lead to the observable performance and power consumption which we show in Figure 1. We observe that the accelerator's power can be greatly reduced using TCs. As Figure 1 shows, we measure with square matrices of size $n = 16384$ using TCs at half precision: see HGEMM (with TC), and GPU cores at single and double precision (see SGEMM & DGEMM). Two important points to notea are: a) SGEMM and DGEMM draw power close to the TDP (300 W), and b) SGEMM or DGEMM cannot run concurrently with HGEMM. This indicates that SGEMM and DGEMM are already running at the highest possible performance (under TDP constraints), without any compromise due to resources occupied by the TCs.

Finally, with Moore's law ending, adding architecture support is no longer free, but comes at the expense of removing something else, and there is a large number of other architectural optimization that could yield better performance and/or energy efficiency for the same amount of silicon, e.g., improved caches, more aggressive OoO, more cores, etc. Hence, to reason whether spending silicon on MEs is a good architectural direction to pursue for HPC, we first must answer a more critical question: *To what extent do HPC applications and DL workloads actually invoke GEMM-like operations?*

III. Matrix Engines from a Software Perspective

In this section we describe, categorize, and analyze a broad set of applications and benchmarks, from traditional HPC workloads to modern Deep Learning, and analyze historical data from one supercomputer. The section is structured as follows: first we perform a historical data and an offline software dependency analysis, then conduct various measurements on

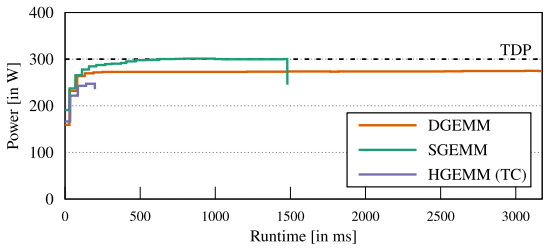


Fig. 1: Power consumption evaluation of GPU cores and TCs on a single Tesla V100 GPU using squared matrices of size $n = 16384$; collected using NVML API calls (via nvmlDeviceGetPowerUsage).

AI and HPC applications, and finally, for completeness, provide exact details about our evaluation environments.

A. Analysing Historical Data of the K Computer

For the K Computer, RIKEN collected multiple metrics of the executed application [33] and the system, such as power consumption and failure statistics. For parallel applications, launched via MPI, their database includes additional information about the application binary's symbol table (through the Linux tool `nm`), but excludes data from linked shared libraries.

We analyze this data for a full year of K's operation (April '18 to March '19), during which the system executed $487,563$ scientific applications distributed over 543 million node hours. Symbol table data is only available for 96% of these node hours, while the remainder can be attributed to interactive jobs, non-parallel jobs, or jobs for which the collection was intentionally disabled by the user. When searching the `nm` data for GEMM functions, we can attribute 53.4%, or $277,258,182$ node hours, to applications which likely executed GEMM operations[5]. However, the data does not reveal the exact node hours spent inside of GEMM routines.

To conclude, in the absolute best case, the inclusion of MEs into the K computer could (in theory) have halved the number of node hours without any loss in quality for the applications, but a significant reduction in energy consumption (and, possibly, repair-costs), or an increase science throughput of the machine.

B. Software Dependency Analysis via Spack

The analysis of the job history of the K computer is rather narrow, both in terms of common HPC workloads found in other countries as well as in terms of possible math libraries providing and utilizing GEMM kernels. Hence, we broaden the scope by also analyzing the software/library dependencies in Spack [34], which is a package manager that targets supercomputers. Spack provides easy access to thousands of scientific software packages to users without the necessity of the administrator's installation and maintenance.

We identified all libraries, which provide dense linear algebra routines, among the 4,371 packages currently supported by Spack, namely: AMD BLIS, Atlas, BLIS, Eigen, ESSL, Intel MKL, Netlib's LAPACK, ScaLAPACK and XBLAS, OpenBLAS, CUDA (cuBLAS), py-blis, libxsmm, and veclibfort.

[5]Fujitsu's compiler infrastructure defaults to selectively including individual functions from their math kernel library, instead of linking the entire library.

TABLE III: Dependency Analysis of Dense Linear Algebra Libraries for Spack Package Manager (w/ & w/o Python and R sub-packages)

Dependency Distance	# and % of Packages	excluding py-* & R-*
0	14 (0.32)	14 (0.55)
1	239 (5.47)	226 (8.87)
2	762 (17.43)	541 (21.23)
3	968 (22.15)	714 (28.02)
1–∞	3061 (70.03)	1311 (51.45)

Hereafter, we refer to these packages as BLAS libraries with dependency distance of 0. Using Spack's ability to track and list dependencies, we subsequently identify packages which directly depend on libraries of dependency distance 0, i.e., a set of packages with dependency distance of 1, and so on and so forth. While the number of packages with dependency distance 0 is only 14, the number increases to 239 in the next step (excluding the set of dependency distance 0), or 5.47% of all Spack packages.

Table III lists further dependency distances, with the last column adjusting for the fact that Spack includes a large number of sub-packages for Python and R, which we merge under their parent packages. As we can see, *51% (or 70% without sub-package adjustment) of Spack's packages depend directly or indirectly on BLAS libraries*. While the similarity of the percentage with Section III-A is mere coincidental, our findings show that in the best case, only about half the packages could benefit from MEs. Unfortunately, our analysis is oblivious of a more important metric: *How much time of the execution is actually spent in these BLAS routines?* In order to reason around said question, we split our efforts into two paths: **(i)** What is the impact of using MEs in Deep Learning workloads, which are known to contain a lot of GEMM operations?, and **(ii)** What *could* be the impact of using MEs in regular HPC applications whose GEMM usage today remains unquantified?

C. Matrix Engines in Deep Learning Applications

Deep Learning (DL) has been a major driver in the active development of architectures with MEs, and corresponding software support, in recent years. As a result, workloads of this class are ready for immediate empirical evaluation. Moreover, since DL applications are typically comprised from a small number of well-known kernels (e.g., convolution), optimized implementations typically do not statically link to BLAS libraries. They either decide at run-time to call BLAS depending on the kernel parameters, or inline matrix multiplications inside of kernels. Therefore, we rely on empirical evaluation of DL applications, listed in the next Section, using our benchmarking framework[6] with PyTorch [35] as the backend.

1) Machine Learning Benchmarks: For this evaluation we use a number of DL models spanning various domains and largely intersecting with the popular MLPerf benchmark [36]. Namely, we use: BERT [37], a language model based on self-attention mechanism; Cosmoflow [38], a 3D convolutional neural net (CNN) used in computational cosmology; VGG16 [39] and ResNet50 [40], two 2D CNN for image recognition; DeepLabV3 [41], a 2D CNN for image segmentation; SSD300 [42], a 2D CNN for object detection; and NCF [43], a recommender model based on collaborative filtering. Additionally, we benchmark several individual layers: GEMM/dense; LSTM [44] and GRU [45], both forms of recurrent units; 2D convolution layer; and self-attention [46].

[6]**Benchmarker** tool (available: http://benchmarker.blackbird.pw/) uses synthetic data sets, with same input sample size but reduced number of samples, to execute DL models on arbitrary number of GPUs; 1 GPU in our tests.

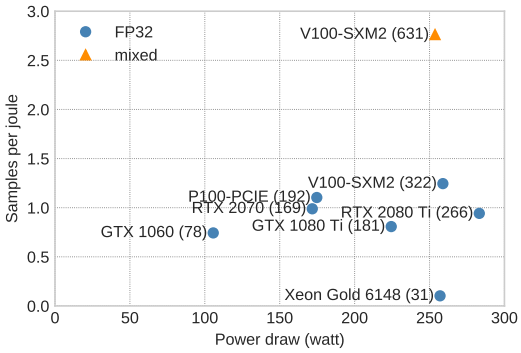


Fig. 2: Energy-efficiency of ResNet50 training; Throughput in images/s (samples per second) indicated in parenthesis

2) Power Evaluation of DL Workloads: We first perform energy-efficiency measurements of ResNet50 training across a range of consumer and data-center chips (cf. Section III-E2 for further details). Evaluation results are summarized in Figure 2, which shows that while new GPUs are reported to be delivering better performance for DL applications (among other things), this increase comes at cost of higher power consumption while showing only a marginal generational improvement in energy-efficiency ($^{\text{flop}}/_{\text{J}}$; or image samples per Joule). However, using TCs does yield higher energy efficiency (cf. `mixed` in Figure 2), resulting from doubled image throughput at roughly the same power consumption. This 2x practical improvement for ResNet50 is below our expected ≈7x gain, derived from moving SGEMM operations to mixed precision HGEMM on TCs as visualized in Figure 1, and hence we analyze the fraction of GEMM operations in DL models in the next Section.

3) Evaluation of Tensor Core Occupancy: Table IV shows a range of DL models run with PyTorch framework in FP32 precision and in mixed precision mode, using apex library, on a Tesla V100 GPU. We use NVIDIA's `nvprof` tool to collect function call profiles for individual kernels. It should be noted, that these results can be biased in both directions: for some kernels, e.g., 3D convolutions of Cosmoflow model, the implementation utilizing TCs is not yet available and the reported speedup is less than possible. On the other hand, LSTM kernel running in reduced precision mode is utilizing a completely different underlying algorithm—thus the reported performance improvement can not be attributed to more efficient matrix multiplications. However, after manually verifying which kernels are being executed, we consider these results to be representative of AI workloads in general. To summarize, the investigated DL applications show performance improvements of 2x (typical ConvNets) to 4x (Transformers), when using TCs. These numbers, while not as high as the 7.6x of pure GEMM kernels, still provide substantial runtime

TABLE IV: Throughput Improvement from FP32 to Mixed Precision. **%TC**: percentage of time spent on Tensor Cores (relative to total time); **%TC comp**: compute time spent on TCs excluding data movement; and **%Mem**: time for data movement between host and device.

Benchmark	Speedup	% TC	% TC comp	% Mem
BERT	3.39x	50.86	55.26	7.97
Cosmoflow	1.16x	0.04	0.05	22.90
VGG16	1.71x	12.30	12.74	3.45
Resnet50	1.97x	16.32	16.78	2.76
DeepLabV3	1.75x	16.33	16.44	0.69
SSD300	1.78x	8.55	8.66	1.32
NCF	0.97x	22.37	26.79	16.50
GEMM	7.59x	20.08	99.90	79.90
GRU	3.67x	6.59	7.48	11.94
LSTM	5.69x	11.63	13.85	16.03
Conv2D	1.12x	0.27	0.32	16.78
Attention	3.49x	44.49	58.19	23.55

improvements to users and clearly validate the usage of TCs in the DL domain. Furthermore, considering the percentage of execution time spend utilizing TCs, we expect even greater speedup with more efficient matrix engines.

D. HPC Proxy-Applications and SPEC Benchmarks

In contrast to the previous section, here we are in looking into more traditional supercomputing and HPC applications.

1) Application Overview and Input Selection: We select benchmarks from six different sets which represent typical workloads and are commonly used by HPC centers and vendors for architecture comparisons and hardware procurement:

- TOP500 Benchmarks [47]: The HPC community utilizes High Performance Linpack (HPL) and High Performance Conjugate Gradients (HPCG) for a world-wide performance ranking of supercomputers and HPC systems.
- Exascale Computing Project (ECP) Proxy Applications [48]: The ECP released with version 1.0 a set of 12 workloads, which are used for procurement by HPC centers in the USA. We investigate eleven of them in our study[7].
- RIKEN CCS' Fiber Miniapp Suite [49]: The eight proxy applications developed by RIKEN CCS represent the priority areas of the Japanese government and were used in the procurement of Supercomputer Fugaku [50].
- SPEC Benchmarks [51]: While the SPEC corporation hosts benchmarks for numerous areas of interest, such as clouds or accelerators, we focus on SPEC CPU 2017 V1.1 (24 benchmarks; contains OpenMP-accelerated versions), and the two sets derived from HPC workloads, namely SPEC OMP 2012 V1.1 (14) and SPEC MPI 2007 V2.0.1 (18).

The detailed list of all 77 HPC benchmarks, i.e., individual names per set, as well as an overview of the scientific domain they are representing, is available in Table V.

Our selection of inputs and configurations for the individual benchmarks of the TOP500, ECP, and RIKEN Fiber sets remains unchanged from our previous publication on double-precision floating-point unit utilization, and therefore readers may consult Sec. II (B) of [52] for further details.

SPEC benchmarks generally provide three different input sizes (ordered by short to long runtime): `test`, `train`, and `ref`(erence). The difference between those three options is usually the problem size, e.g., time steps, grid size, #atoms, etc., with a few exceptions as stated in the SPEC documentation (yet we expect no major changes in compute patterns). Hence, we select the `train` input configuration for all our SPEC measurements. For SPEC MPI `mtrain` is preferred over `ltrain` whenever possible. Furthermore, we setup SPEC to perform `peak` runs (not `base`) which allows the usage of OpenMP threads for SPEC CPU benchmarks.

2) Measurement Methodology: From our prior work [52], we know that TOP500, ECP, and RIKEN's benchmarks are highly optimized proxies for the represented applications (with the sole exception of Laghos), which utilize high performance math libraries, such as Intel's MKL whenever suitable. To identify the dense, GEMM-like operation which could be accelerated by a ME, we employ the profiling functionality of the Score-P performance analysis tool [53]. We create a Score-P library wrapper for all functions found in the header files for dense-matrix compute libraries of MKL, i.e., (C)BLAS, PBLAS, BLAS-like Extension Transposition Routines, (C)LAPACK, and ScaLAPACK.

Furthermore, since the applications can exhibit considerable amounts of initialization and post-processing phases, we identify these phases and exclude them from profiling via Score-P API calls. Additionally, we search the source codes for function names indicating GEMM operations or Fortran's `matmul` intrinsic, and instrument them to be included in our application profiles for the TOP500, ECP and RIKEN's set.

Unfortunately, SPEC benchmarks are implemented without external dependencies for portability and ease of use reasons. Hence, we are not able to rely on our Score-P library wrapper for these three SPEC sets. Our workaround involves a three step approach—besides searching the source codes for relevant function names as above—before collecting the necessary metrics: first, we utilize the Intel Advisor tool to perform a Roofline analysis and extract all source code regions which are tagged as "compute intensive", meaning functions and loops with sufficient Arithmetic Intensity, i.e., flop/byte ratio ≥ 7 for our CPU testbed (cf. System 1 in Section III-E1). This list was further pruned by requiring a Point Weight (PtW) ≥ 1, i.e., the self-elapse time of this region is at least 1% of the total elapse time of the benchmark. The next step consists of *two independent, manual code inspection of all 598 location* to determine their compute pattern (majority vote; with third check if needed), and instrument them to be included in our application profiles when they perform a GEMM-like operation. We identify and instrument 14 source code locations[8]. And lastly, we rely on Score-P's automatic compiler instrumentation and function filter ability—filtering the same function names we collected for the MKL wrapper above—to profile source code regions where the benchmark programmer replaced external library calls with hand-written functions. For SPEC benchmarks, we omit the exclusion of initialization and post-processing phases, since these are negligible[9].

[7]We exclude CANDLE, since Section III-C covers AI workloads extensively.

[8]Similarly, we identified numerous hand-written GEMM kernels in Nekbone.

[9]According to SPEC's rules for benchmark submission: 95% of runtime must be compute-bound and spend within the benchmark's own source code.

TABLE V: Overview of (Proxy-) Applications used for this Study; '(R)' for SPEC CPU indicates lack of OpenMP parallelization

Set	Name	Sci. / Eng. / AI Domain	Name	Sci. / Eng. / AI Domain	Name	Sci. / Eng. / AI Domain
Deep Learning	BERT	Natural Language Processing	DeepLabV3	Image Segmentation	GRU	Single Layer
	Cosmoflow	Computational Cosmology	SSD300	Object Detection	LSTM	Single Layer
	VGG16	Image Recognition	NCF	Recommender Systems	Conv2D	Single Layer
	Resnet50	Image Recognition	GEMM	Single Layer	Attention	Single Layer
TOP500	HPL	Math/Computer Science	HPCG	Math/Computer Science		
ECP	AMG	Physics and Bioscience	miniAMR	Geoscience/Earthscience	SW4lite	Geoscience/Earthscience
	CoMD	Material Science/Engineering	miniFE	Physics	SWFFT	Physics
	Laghos	Physics	miniTRI	Math/Computer Science	XSBench	Physics
	MACSio	Math/Computer Science	Nekbone	Engineering (Mechanics, CFD)		
RIKEN	FFB	Engineering (Mechanics, CFD)	mVMC	Physics	NTChem	Chemistry
	FFVC	Engineering (Mechanics, CFD)	NGSA	Bioscience	QCD	Lattice QCD
	MODYLAS	Physics and Chemistry	NICAM	Geoscience/Earthscience		
SPEC CPU	blender(R)	Math/Computer Science	exchange2	Artificial Intelligence	omnetpp	Math/Computer Science
	cam4(R)	Geoscience/Earthscience	fotonik3d	Physics	perlbench	Math/Computer Science
	namd(R)	Material Science/Engineering	gcc	Math/Computer Science	pop2	Geoscience/Earthscience
	parest(R)	Bioscience	imagick	Math/Computer Science	wrf	Geoscience/Earthscience
	povray(R)	Math/Computer Science	lbm	Engineering (Mechanics, CFD)	roms	Geoscience/Earthscience
	bwaves	Physics	leela	Artificial Intelligence	x264	Math/Computer Science
	cactuBSSN	Physics	mcf	Math/Computer Science	xalancbmk	Math/Computer Science
	deepsjeng	Artificial Intelligence	nab	Material Science/Engineering	xz	Math/Computer Science
SPEC OMP	applu331	Engineering (Mechanics, CFD)	fma3d	Physics	mgrid331	Engineering (Mechanics, CFD)
	botsalgn	Bioscience	ilbdc	Engineering (Mechanics, CFD)	nab	Chemistry
	botsspar	Math/Computer Science	imagick	Math/Computer Science	smithwa	Bioscience
	bt331	Engineering (Mechanics, CFD)	kdtree	Math/Computer Science	swim	Geoscience/Earthscience
	bwaves	Engineering (Mechanics, CFD)	md	Material Science/Engineering		
SPEC MPI	[d]leslie3d	Engineering (Mechanics, CFD)	[l]GemsFDTD	Physics	socorro	Material Science/Engineering
	[d]milc	Lattice QCD	lu	Engineering (Mechanics, CFD)	tachyon	Math/Computer Science
	fds4	Engineering (Mechanics, CFD)	[l]wrf2	Geoscience/Earthscience	tera_tf	Geoscience/Earthscience
	GAPgeofem	Physics	pop2	Geoscience/Earthscience	zeusmp2	Engineering (Mechanics, CFD)
	lammps	Material Science/Engineering	RAxML	Bioscience		

For TOP500, ECP and RIKEN's benchmarks, we compile and execute the tests equivalently (except for FFB[10]) to our previous work (i.e., using the Intel compiler suite, compiler flags as listed in Sec. III (A), and a combination of MPI processes and OpenMP threads as provided in Tab. IV of [52]). These combinations yield the best performance for each application on our benchmarking hardware.

In contrast, we primarily make use of the GNU compiler suite for SPEC benchmarks[11], except for botsspar, bt331, fds4, perlbench, and socorro, which require Intel's compilers to avoid runtime issues. The main compiler optimizations are `-O3`, `-march=native` or `-xHOST` (depending on the compiler suite), and additional flags for vectorization, loop unrolling, and fast math[12]. We run SPEC CPU and OMP benchmarks consecutively and with 48 OpenMP threads to match the number of logical cores on our testbed. Similarly, all SPEC MPI test are run with 48 MPI processes matching the core count, because SPEC MPI benchmarks are not OpenMP-parallelized. In the next section, we will provide the results and analysis of our profiling measurements.

3) Evaluation of Measurement Results: The results of our measurements, as outlined in Section III-D2, to identify the GEMM usage per HPC (proxy-)application is visualized in Figure 3. We analyze the Score-P profiles for the runtime spend in the profiled regions[13]. Furthermore, we distinguish between, and show, four different compute regions in the Figure: (1) GEMM operations, (2) BLAS operations (non-GEMM), i.e., all BLAS Level[1|2|3] functions except for matrix-matrix multiplications, (3) LAPACK and ScaLAPACK functions, and (4) all other code regions. This distinction allows for an easy classification of code sections which can be either directly accelerated (1), or potentially indirectly accelerated (2 & 3), or most probably not accelerated (4) by MEs. To clarify, the indirectly acceleration can result from use of GEMM operations within (Sca)LAPACK routines or from mapping other BLAS function (such as GEMV) to MEs.

Unfortunately, as we can see from Figure 3, the number of existing HPC applications and workloads which can directly benefit from MEs is rather sparse. Obviously, the most dominant example is High Performance Linpack. In our test, HPL's compute kernel spends 76.81% in GEMM operations and 0.14% in other BLAS function, and hence, executing the same test on a CPU with matrix engine could substantially reduce the runtime of HPL. Other combinations of benchmarks and chosen input parameters which perform GEMM are: Laghos (41.24%), NTChem (25.78%), Nekbone (4.58%), SPEC OMP's botsspar (18.9%) and bt331 (14.16%), and SPEC MPI's milc & dmilc (40.16% and 35.57%, respectively) and socorro benchmark (9.52%). We have instrumented GEMM regions in other benchmarks as well, but our measurement indicate that these regions are either dormant, or the type of input results in alternative code paths, bypassing the GEMM operations.

[10]Using combination of Score-P and Intel Compiler for FFB results in erroneous intermediate results and early benchmark termination.

[11]Only for GNU compilers, Score-P performs filtering of instrumented function at compile time, not at runtime, reducing the profiling overhead.

[12]For compilation details see our framework: gitlab.com/domke/MEstudy

[13]Percentages derived from ratio of {region runtime} to {total application runtime minus MPI_Init/Finalize, and initialization/post-processing phases}

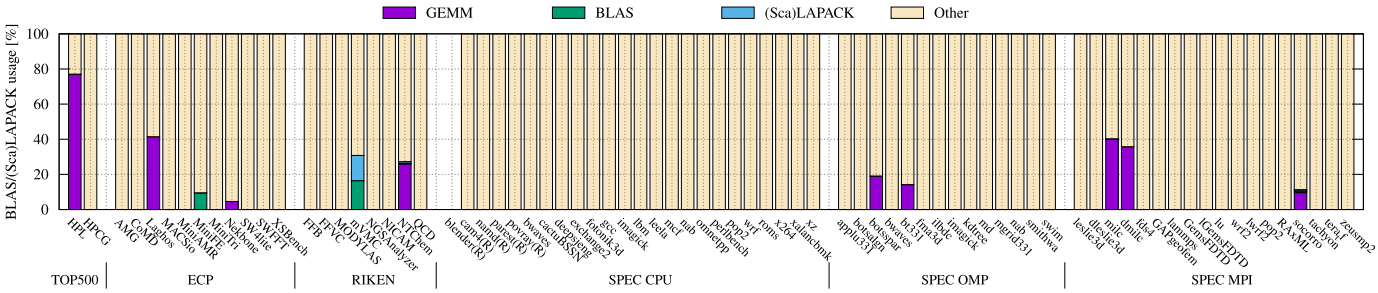


Fig. 3: GEMM, BLAS (non-GEMM functions), and (Sca)LAPACK utilization across 77 HPC benchmarks, see Section III-D3 and Table V for details [Data for blender(R) missing due to unresolvable runtime errors; however, the source code contains GEMM function calls]

Furthermore, we see that MiniFE, NTChem, mVMC, and socorro utilizes other BLAS functions during 9.38%, 0.45%, 16.41%, and 0.99% of their runtimes, respectively. Upon closer inspection of the profiles, it is unlikely that the former two benchmarks can benefit from MEs, because they utilize only BLAS Level 1 (i.e., vector-vector operations), while the other two also call BLAS Level 2 functions. Looking at the (Sca)LAPACK dependencies and percentages of runtime in Figure 3 yields similar results: mVMC spends noticeable amount of runtime (14.35%) in those libraries, while NTChem (0.95%) and socorro's (0.73%) usage is negligible. Nevertheless, only these three out of all 77 could benefit from MEs if the used (Sca)LAPACK functions can be mapped to them.

Overall, we can state that only ten out of the 77 HPC benchmarks, which we investigate at for this study, perform GEMM operation or outsource dense linear algebra computations to libraries. Assuming an idealized equal distribution of the node hours spend per our 77 benchmark on a supercomputer results in an average runtime of 3.5% spend in GEMM operations, or roughly 12 and a half days of a full year.

E. Details of Evaluation Environments for Reproducibility

1) CPU-based Measurements: For the CPU-based measurement of our HPC (proxy-)applications in Section III-D, and the power efficiency evaluation in Section II-C we use a dual-socket x86_64 compute node with Intel Xeon CPUs. The details of this system are listed in Table VI under **System 1**.

2) GPU-based Measurements: For the DL energy-efficiency measurements in Section III-C we use NVIDIA GPUs ranging from consumer models, i.e., GTX 1060 & 1080 Ti and RTX 2070 & 2080 Ti, up to the data center lineup of Tesla P100-PCIE and V100-SXM2 GPUs. Additionally, the section also includes results of a Intel Xeon Gold CPU (compute node of ABCI [54]; see **System 2** in Table VI) for comparisons. The analysis of TC utilization by different kernels, see Table IV, as well as power evaluation of GPU cores in Section II-C, and the comparisons between native and emulated precision GEMMs in Section IV-B, is done on a Tesla V100-SXM2 of ABCI.

3) Auxiliary Software: We employ various auxiliary software packages to facilitate our measurements on the different hardware architecture, such as compilers, Deep Neural Network (DNN) libraries, and performance analysis frameworks. The main software components are listed in Table VII.

[14]Version 2019 (of Intel's P.S. XE) lacks needed feature for roofline analysis.

TABLE VI: CPU-based Compute Nodes used for Measurements

	System 1 (§II-C, §III-D3)	System 2 (§III-C2)
Mainboard	Supermicro X10DRG-Q	Fujitsu Primergy-RX2540-M4
CPU	2x Intel Xeon E5-2650v4	Intel Xeon Gold 6148
#Cores	24 (hyper-threading enabled)	20 (hyper-threading enabled)
Memory	256 GiB DDR4 2400 MHz	32 GiB DDR4 2666 MHz
OS	CentOS Linux (v7.8.2003)	CentOS Linux (v7.5.1804)
Kernel	3.10.0-1062	3.10.0-862

TABLE VII: Auxiliary Software used for Measurements

Package	Vers.	Package	Vers.
Intel Parallel Studio XE	2019; U1	NVIDIA CUDA Toolkit	10.[1&2]
Intel Advisor	2020; U2[14]	NVIDIA cuDNN	7.6.5
GNU Compiler Coll.	8.4.0	PyTorch ML Framework	7.6.5
FUJITSU Software Suite	1.2.27b	Score-P Analysis Frame.	6.0
Spack Package Manger	0.15.1	Intel PCM tools	201710

IV. Gain from adapting Matrix Engines in HPC

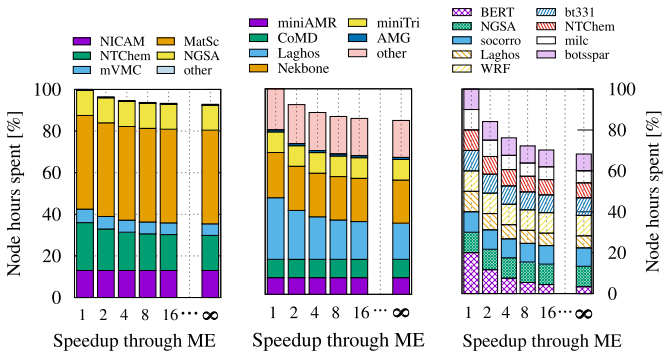
Assuming future CPUs and GPUs extensively integrate low (and/or high) precision MEs, then we assess here the potential benefit for the HPC community when utilizing these units.

A. Performance Extrapolation for Matrix Engines in HPC

From annual HPC system utilization reports for the K computer [55], and breakdown by domain, we know that the system was used primarily for material science (45%), chemistry (23%), geoscience (13%), biology (12%), and physics (6.5%) calculations, plus 0.5% "other". For the following thought experiment, we select for each science domain (see Table V) a representative RIKEN Fiber benchmark, with FFB & MODYLAS & QCD representing material science equally (short MatSc), and we assume "other" applications spend 10% in GEMM. In Figure 4a, we visualize an extrapolation of node hours spend while assuming that these applications were accelerated by a ME for all GEMM and (Sca)LAPACK operations. Under these idealized conditions, and assuming a ME providing 4x speedup over baseline, the consumed node hours would have only reduced by 5.3%. Even an infinitely fast ME would only yield a 7.1% reduction.

Similarly, in Figure 4b, we show the theoretically consumed node hours for the systems of the Argonne National Laboratory (ANL) [56], and we utilize our measurements from the ECP applications for the GEMM portion. Here, Laghos (representing 30% physics simulations at ANL), Nekbone (rep. 22% engineering), and "other" contribute to a 11.5% reduction in node hours for a ME, which provides 4x speedup.

We perform the same extrapolation in Figure 4c for a fictional, future supercomputer. In this case, we assume that 20% of its compute cycles are spend on AI/DL tasks, either



(a) K Computer (b) ANL systems (c) Future System

Fig. 4: Node hour reduction by utilizing hypothetical ME; Breakdown of node hours per science domain based on historical data for Figure 4a and 4b; Future HPC system (4c) assumed to execute 20% AI/DL tasks; In 4a, **MatSc** represents FFB+MODYLAS+QCD in equal fractions

within AI applications or as AI subroutine of another scientific application. The remaining 80% are equally distributed across our eight science domains (see Table V), and we select one representative application, with the highest GEMM and (Sca)/LACPACK percentage, for each. For the AI portion, we select BERT and assume its GEMM occupancy is 83.2%[15]. The inclusion of MEs, assuming they provide an 4x (or infinite) speedup, could reduce the node hours of our devised system by 23.8% (or 32.8%, respectively).

Obviously, these results in Figure 4 extrapolate the absolute best case scenario, since we sampled applications with the highest GEMM percentage, but in a real environment the node hour reduction is further constrained by non-GEMM applications (cf. Figure 3) and overheads, such as I/O or MPI.

B. Emulating High-precision Compute with Low-precision MEs

While the industry is moving towards adopting MEs, whose impact on HPC we have analyzed in prior sections, there remains the issue regarding the numerical representation that these MEs will support. While the recent NVIDIA A100 GPU does support full IEEE-754 double-precision in their MEs, other manufacturers may chose to provide MEs only for a shorter numerical representation (e.g., Intel Sapphire Roads supports bfloat16). Are those low-precision MEs still valuable for use in future HPC systems? It turns out that they can be, as we can *emulate* higher precision numerics using low-precision arithmetics. While there exists several recent papers on the subject (e.g., [57], [58]), we will go into the details of one particular method called the Ozaki scheme [59]. It enables us to emulate a high-precision matrix-multiplication utilizing multiple low-precision matrix-multiplications. For example, double-precision DGEMM can be computed using GEMM executed on Tensor Cores (cublasGemmEx) [60].

In this Ozaki scheme, the mantissa and the exponent are computed separately. At a glance, firstly, the input matrices are element-wisely split into several matrices, starting with the one

[15]The 83% is derived from BERT's `%TC comp` in Table IV assuming the TCs yield 4x speedup over FP16 baseline using: $4 \cdot p / \big(4 \cdot p + (100 - p)\big) * 100$

TABLE VIII: Performance of cuBLAS routines ($m = n = k = 8192$) and GEMM-TC (software emulation using TCs) on Tesla V100.

Implementation	Condition	Tflop/s	Watt	Gflop/J
cublasGemmEx	FP16/FP32-mixed	92.28	270.9	340.70
cublasSgemm	—	14.54	276.1	52.66
cublasDgemm	—	7.20	286.5	25.14
SGEMM-TC	input range: 1e+8	4.721	284.1	16.62
	input range: 1e+16	2.140	278.9	7.67
	input range: 1e+32	1.758	264.5	6.65
DGEMM-TC	input range: 1e+8	1.097	273.7	4.01
	input range: 1e+16	0.716	271.4	2.64
	input range: 1e+32	0.618	270.4	2.29

with the largest absolute value of the elements. The exponent part is separated at this point. Next, the all-to-all product of the split matrices is computed. Here, the GEMM executed on TCs (cublasGemmEx) can be used for those matrix-multiplications. The exponent part is computed using integer operations. Finally, by summing the all-to-all product and restoring the exponent part, the accurate result of the matrix-multiplication is obtained.

Although this scheme uses FP64 arithmetic for the split and summation, cublasGemmEx dominates the total execution time. When the numbers of split matrices for matrices A and B are s_A, s_B, respectively, the most accurate result is obtained by performing $s_A \cdot s_B$ matrix-multiplications, but if only a DGEMM-equivalent accuracy is desired, the numbers of split matrices and matrix-multiplications can be reduced (cf. [60] for details). However, we note that the performance is input-dependent, i.e., the number of split matrices required depends on the absolute value range of the elements of the input matrix, the number of significand-bits in the elements, and the number of dimensions in the inner product direction of the matrices.

Table VIII shows the computational performance of cuBLAS routines compared to emulated GEMM using TCs (cf. [S|D]GEMM-TC) on a Tesla V100 ($m = n = k = 8192$). We approximate the flop/s with $2 \cdot n^3$/runtime, and measure the power consumption through NVIDIA's management library (NVML). Although the performance of GEMM-TC does not outperform cuBLAS on this particular GPU, the techniques makes it possible to use TCs (or other matrix engines) to mitigate the lack of FP[32|64] MEs or to mitigate a limited availability of high-precision FPUs in other processors. In fact, DGEMM-TC outperforms cublasDgemm on a NVIDIA Titan RTX, where 64-bit FPUs are limited [60].

Other notable features of this scheme are: (1) bit-wise reproducibility (independent of the thread count and regardless of the rounding-error caused by GEMMs used in the computation), and (2) it can be used to compute dot-product and matrix-vector multiplication [61]. In the latter case, matrix engines could be used for the internal computations of the BLAS calls.

V. Implications

This section compiles a list of arguments for and against adding MEs to the compute systems designed for HPC.

A. Opportunities for Replacing/Complimenting Vector Units

1) Dark Silicon: As our experiments in Section II-C with TCs have shown, either of FPUs or TCs will get close to the TDP of the device; and both FPUs and TCs can not be used at

the same time. Accordingly, one could treat the chip area taken by TCs as simply non-valuable; even if one would release this chip area used by TCs, we are not able to add other resources to contribute to compute performance coming from the FPUs without changing the TDP (since both SGEMM and DGEMM are already running close to TDP as Figure 1 shows).

2) Other Compute Patterns Benefiting from Matrix Engines: The use of MEs can be expanded to compute patterns that could be represented as dense matrix operations. Note that the use of MEs in Deep Learning is driven by re-structuring convolution filters into matrices [12], an approach which may soon be obsolete [62]. In HPC, similar examples are emerging, e.g., efforts include accelerating sparse matrix-matrix multiplication by fitting tiles of the sparse matrix which contain one or more elements to TC fragments [63]. Other efforts try to exploit MEs at a lower level by using compiler-based approaches (i.e. polyhedral analysis) to automatically transform compute-intense nest loops to TCs [64].

3) Lower/Mixed Precision in Scientific Computing: MEs can be designed to have hardware support for arbitrary precisions (as discussed in Section II). However, the demand from AI/ML driving the rise of MEs incentivizes vendors to use lower precision. Lower precisions that are gaining more support in MEs could in turn incentivize the HPC community to re-evaluate its stance on precision requirements (we refer the reader to a detailed survey that elaborates on the opportunities of mixed precision in commonly used numerical methods [65]).

B. Challenges for Replacing Vector Units with Matrix Engines

1) Inefficiency for Level-1 and Level-2 BLAS Operations: Current MEs are mostly built using systolic arrays (cf. Section II). The 2D nature of systolic arrays makes them extremely efficient for matrix-matrix multiplication (Level-3 BLAS operations), but not very-efficient at other operations such as Matrix-Vector (Level-2 BLAS), or Vector-Vector (Level-1 BLAS) given one of the dimensions of the systolic array would be waiting for the vector to propagate through. SIMD-style designs meanwhile can work efficiently for BLAS Levels 1 & 2 [66].

2) Programmability Burden: The use of MEs in AI/ML is to a large extent hidden from the end users, i.e. happens inside the frameworks and libraries. The use of MEs for dense matrix operators is also hidden behind APIs provided by linear algebra libraries (e.g., NVIDIA exposes a API that enables linear algebra libraries to use TCs). However, for other compute patterns, unsupported by such libraries, it becomes the burden of the user to write source code to use the MEs. Approaches for auto-generation of code for MEs [64] are still in very early stages, and will take time to mature.

3) Reduced Portability: Portability of source code between SIMD vector extensions from different generations is often troublesome and requires manual tuning of the code (e.g. using `#ifdef` programming style). We have no historical evidence indicating that the situation will be any different with MEs—if not more complicated, given that there no convergence yet on how MEs are exposed to the end user and compiler.

4) Is the Dark Silicon Effect General?: As we showed, TCs in NVIDIA GPUs do not take away from the performance potential of 32-|64-bit FPUs, which run close to TDP. However, there is no clear evidence that this dark silicon effect would be the same for CPUs, or GPUs made by other vendors.

5) Overhead of Data Staging to Matrix Engine: When the compiler vectorizes a loop, vector registers are used for the operands. Accordingly, vectors extensions do not have any particular overhead for data movement (or staging). In comparison, TCs/MEs, in the currently available forms, come with an overhead attributed to the off-loading model that separates the ME memory hierarchy from the host processor (CPU or GPU).

VI. Related Work

Ahmad et al. [65] conducted a comprehensive survey—from the point of view of numerical methods—of efforts that utilize mix-precision hardware. The survey focuses on dense linear algebra methods such as HGEMM and LU factorization, communication compression using reduced precision, sparse algorithms, and other algorithms that are fundamental parts of HPC applications. They also provided an overview of existing math libraries that support mixed-precision arithmetic.

Markidis et al. [57] proposed a method to achieve the SGEMM-equivalent accuracy TC-accelerated GEMMs. Additionally, the authors provided a method to cope with precision loss. Zachariadis et al. [63] focus on mixed-precision sparse general matrix-matrix multiplication (spGEMM). Furthermore, Mukunoki et al. [60] proposed a method to utilize TCs to emulate SGEMM and DGEMM, and Sorna et al. [58] proposed a method to improve the accuracy of 2D fast Fourier transformation on TC. Overall, those efforts were directed to specific use cases of Tensor Cores, and do not have the HPC-wide perspective of MEs which we provide in this study.

VII. Conclusion

To conclude, we revisit our introductory questions. Our detailed analysis of the prospect of matrix engines, both for Deep Learning and for HPC workloads, gives little reason to believe that the HPC community at large should actively embrace and pursue the inclusion of more, faster, and higher-precision matrix engines. For DL the current architectures, such as Tensor Cores, will likely soon hit a point of diminishing returns due to Amdahl's Law and the research will likely be tilting towards more sparsity, while for traditional HPC the low utilization of GEMM does not seem to merit a wide adoption.

Furthermore, we explore opportunities that could arise from the availability of matrix engines in future systems, which signals that the supercomputing community should be careful not to prematurely dismiss MEs, specially as they proliferate due to market forces. Obviously, individual HPC centers need to revisit their particular priority applications to make a final assessment, and potentially an overall science throughput improvement of $\approx$1.1x, as we demonstrate for existing supercomputers, might justify the investment if all other architectural options have been exhausted.

ACKNOWLEDGMENT

This work was supported by the Japan Society for the Promotion of Science KAKENHI Grant Number 19K20286; by JST, PRESTO Grant Number JPMJPR20MA, Japan; by the New Energy and Industrial Technology Development Organization (NEDO); and the AIST/TokyoTech Real-world Big-Data Computation Open Innovation Laboratory (RWBC-OIL). This research utilized the Cygnus HPC system, provided by the Multidisciplinary Cooperative Research Program of the Center for Computational Sciences, University of Tsukuba.

REFERENCES

[1] R. H. Dennard *et al.*, "Design of ion-implanted MOSFET's with very small physical dimensions," *JSSC*, 1974.
[2] R. R. Schaller, "Moore's law: past, present and future," *Spectrum*, 1997.
[3] J. S. Vetter *et al.*, "Architectures for the Post-Moore Era," *Micro*, 2017.
[4] J. Shalf, "The future of computing beyond Moore's law," *Phil. Trans. R. A.*, 2020.
[5] R. Hameed *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *ISCA'10*, 2010.
[6] N. Jouppi *et al.*, "Motivation for and Evaluation of the First Tensor Processing Unit," *Micro*, 2018.
[7] I. Kuon *et al.*, *FPGA Architecture: Survey and Challenges*. Now Publishers, 2008.
[8] H. R. Zohouri *et al.*, "Evaluating and Optimizing OpenCL kernels for High Performance Computing with FPGAs," in *SC16*, 2016.
[9] A. Podobas *et al.*, "A Survey on Coarse-Grained Reconfigurable Architectures from a Performance Perspective," *arXiv:2004.04509*, 2020.
[10] D. E. Shaw *et al.*, "Anton, A Special-Purpose Machine for Molecular Dynamics Simulation," *Communications of the ACM*, 2008.
[11] J. Makino *et al.*, "GRAPE-8–An accelerator for gravitational N-body simulation with 20.5 Gflops/W performance," in *SC12*, 2012.
[12] Y. LeCun *et al.*, "Deep learning," *Nature*, 2015.
[13] J. Shin *et al.*, "Speeding up Nek5000 with Autotuning and Specialization," in *ICS'10*, 2010.
[14] P. Warden, "Why GEMM is at the heart of deep learning," *Peter Warden's Blog*, 2015.
[15] H. T. Kung *et al.*, "Systolic Arrays for (VLSI)," CMU, Tech. Rep., 1978.
[16] H. Liao *et al.*, "DaVinci: A Scalable Architecture for Neural Network Computing," in *Hot Chips*, 2019.
[17] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA'17*, 2017.
[18] J. Choquette *et al.*, "Volta: Performance and Programmability," *Micro*, 2018.
[19] Intel, "Intel® Architecture Instruction Set Extensions and Future Features Programming Reference," *Whitepaper*, 2020.
[20] W. Starke *et al.*, "IBM's POWER10 Processor," in *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, Aug. 2020, pp. 1–43.
[21] M. I. Soliman, "Mat-core: A matrix core extension for general-purpose processors," in *ICCES'07*, 2007.
[22] D. W. Wall, "Limits of instruction-level parallelism," in *ASPLOS91*, 1991.
[23] R. Duncan, "A survey of parallel computer architectures," *Computer*, 1990.
[24] T. Yoshida, "Fujitsu high performance CPU for the Post-K Computer," in *Hot Chips*, 2018.
[25] M. Cornea, "Intel AVX-512 instructions and their use in the implementation of math functions," *Whitepaper*, 2015.
[26] D. Kirk *et al.*, "NVIDIA CUDA software and GPU parallel computing architecture," in *ISMM'07*, 2007.
[27] T. N. Theis *et al.*, "The end of moore's law: A new beginning for information technology," *Computing in Science & Engineering*, 2017.
[28] A. Reuther *et al.*, "Survey and benchmarking of machine learning accelerators," *arXiv:1908.11348*, 2019.
[29] T. Wang *et al.*, "A survey of FPGA based deep learning accelerators: Challenges and opportunities," *arXiv:1901.04988*, 2018.
[30] Habana Labs Ltd., "Gaudi™ Training Platform White Paper," *Whitepaper*, 2019.
[31] J. Shalf *et al.*, "Exascale Computing Technology Challenges," in *VECPAR*, 2011.
[32] Z. Xianyi *et al.*, "OpenBLAS," 2012. URL: {http://xianyi.github.io/OpenBLAS}
[33] K. Yamamoto *et al.*, "The K computer Operations: Experiences and Statistics," in *ICCS*, 2014.
[34] T. Gamblin *et al.*, "The Spack package manager: bringing order to HPC software chaos," in *SC15*, 2015.
[35] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *NIPS*, 2019.
[36] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," in *ISCA*, 2020.
[37] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL-HLT*, 2019.
[38] A. Mathuriya *et al.*, "CosmoFlow: Using Deep Learning to Learn the Universe at Scale," in *SC18*, 2018.
[39] K. Simonyan *et al.*, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556*, 2014.
[40] K. He *et al.*, "Deep Residual Learning for Image Recognition," *arXiv:1409.1556*, 2015.
[41] L. Chen *et al.*, "Rethinking Atrous Convolution for Semantic Image Segmentation," *arXiv:1706.05587*, 2017.
[42] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," *arXiv:1512.02325*, 2015.
[43] X. He *et al.*, "Neural Collaborative Filtering," *arXiv:1708.05031*, 2017.
[44] S. Hochreiter *et al.*, "Long Short-Term Memory," *Neural Comput.*, 1997.
[45] J. Chung *et al.*, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *CoRR*, 2014.
[46] A. Vaswani *et al.*, "Attention is All you Need," in *NIPS17*, 2017.
[47] E. Strohmaier *et al.*, "TOP500," Nov. 2018. URL: http://www.top500.org/
[48] Exascale Computing Project, "ECP Proxy Apps Suite," 2018. URL: https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/
[49] RIKEN AICS, "Fiber Miniapp Suite," 2015. URL: https://fiber-miniapp.github.io/
[50] S. Matsuoka, "A64fx and Fugaku: A Game Changing, HPC / AI Optimized Arm CPU for Exascale," 2019. URL: https://connect.linaro.org/resources/san19/san19-300k1/
[51] Standard Performance Evaluation Corporation, "SPEC's Benchmarks," 2020. URL: https://www.spec.org/benchmarks.html
[52] J. Domke *et al.*, "Double-precision FPUs in High-Performance Computing: an Embarrassment of Riches?" in *IPDPS19*, 2019.
[53] A. Knüpfer *et al.*, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir," in *Tools for High Performance Computing 2011*, 2012.
[54] AIST, "AI Bridging Cloud Infrastructure (ABCI)," 2020. URL: https://abci.ai/en/about_abci/
[55] AICS, "K computer Annual Report 2016-17," RIKEN Advanced Institute for Computational Science, Tech Report. URL: https://www.hpci-office.jp/k-computer/documents/annualreport2016.pdf
[56] J. Collins *et al.*, "2016 Annual Report - Argonne Leadership Computing Facility," Argonne National Laboratory, Tech Report ANL/ALCF-17/1. URL: https://publications.anl.gov/anlpubs/2017/05/135670.pdf
[57] S. Markidis *et al.*, "NVIDIA Tensor Core Programmability, Performance Precision," in *IPDPSW2018*, 2018.
[58] A. Sorna *et al.*, "Optimizing the Fast Fourier Transform Using Mixed Precision on Tensor Core Hardware," in *HiPCW2018*, 2018.
[59] K. Ozaki *et al.*, "Error-free transformations of matrix multiplication by using fast routines of matrix multiplication & its applications," *Numer. Algor.*, 2012.
[60] D. Mukunoki *et al.*, "DGEMM Using Tensor Cores, and Its Accurate and Reproducible Versions," in *ISC2020*, 2020.
[61] ——, "Reproducible BLAS Routines with Tunable Accuracy Using Ozaki Scheme for Many-Core Architectures," in *PPAM2019*, 2019.
[62] T. Hoefler *et al.*, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," Tech Report, Feb. 2021. URL: https://htor.inf.ethz.ch/publications/img/hoefler-sparsity-in-deep-learning.pdf
[63] O. Zachariadis *et al.*, "Accelerating sparse matrix–matrix multiplication with GPU Tensor Cores," *Computers & Electrical Engineering*, 2020.
[64] S. G. Bhaskaracharya *et al.*, "Automatic Kernel Generation for Volta Tensor Cores," 2020.
[65] A. Abdelfattah *et al.*, "A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic," 2020.
[66] M. I. Soliman, "Performance Evaluation of Multi-Core Intel Xeon Processors on Basic Linear Algebra Subprograms," in *ICCES 2008*, 2008.