SOFTWARE-DEFINED HARDWARE: A STUDY OF EMERGING HARDWARE FOR AI APPLICATIONS

WILLIAM REGLI, BEN JOHNSON[1], LAUREL MILLER-SIMS, ALI KESHAVARZI[2], MOHAMMAD NAYEEM TELI, ETC

Applied Research Laboratory for Intelligence and Security
The University of Maryland

Point of Contact.
William Regli
Executive Director
Applied Research Laboratory for Intelligence and Security
Professor of Computer Science, College of Computing, Mathematics & Natural Sciences
The University of Maryland at College Park
regli@umd.edu

- AI becoming More Data intensive.
- HW Not keeping up Because of 'generality'
- Traditional Metrics Not fit-for-purpose.
  - GOPS/W is a start point
  - Think about MOP vs MOE
- Didn't follow their specific HW stuff.
- Good explanation of how they get it all running + generic Metrics.

[1]Jataware
[2]DARPA

- BPT use more C in Future Work.
  - Nothing HW Specific.

Abstract

The evolution of the field of Artificial Intelligence (AI) has been tightly coupled to the evolution of computing machines themselves. Successive generations of advances in AI have proceeded lock-step with advances in hardware, with the capabilities of hardware inspiring new AI techniques and the aspirations of the AI discipline motivating the creation of new hardware technologies. While this has been the case for over 80 years, the pending end of Moore's Law is creating a vast new landscape of computational capabilities available to the AI community. Many of these new computing machines are inspired by the needs of machine learning and deep neural networks; others are making previously intractable problems more manageable via bespoke techniques for hardware-accelerated brute force. The result is what has been called a Cambrian Explosion of hardware technologies capable of radically altering the architecture of how we design and build intelligent software systems. The availability of this expanded hardware ecosystem has significant implications for AI researchers and educators.

```
The time period from the point of the introduction
of a new material idea to the point at which it
is seriously considered for insertion into an
engine involves a long-term process that can
exceed 20 years.

National Academies of Sciences, Engineering, and Medicine. 2011.
Materials Needs and R\&D Strategy for Future Military Aerospace Propulsion Systems.
Washington, DC: The National Academies Press. https://doi.org/10.17226/13144.
```

1 Introduction

This paper presents work undertaken in support of the Defense Advanced Research Projects Agency (DARPA) Software-Defined Hardware (SDH) program. The SDH program was specifically aimed to produce new capabilities to accelerate the symbolic and mathematical processing pipelines associated with machine learning and other AI algorithms. The key insight of this paper is, given new hardware primitives, how does one determine how and where to best apply them? The team has developed a methodology for understanding, for a given AI application processing workflow, the potential impact of new hardware technology and guidance as to how to re-architect the processing pipeline to access these capabilities.

The Software Defined Hardware. Hardware technologies are enabling the development of systems to enhance decisions that are driven by information processed by AI-technologies. That information can come in the form of thousands of sensors providing information, surveillance, and reconnaissance (ISR) data; logistics/supply-chain and personnel performance measurements; or a host of other sources and formats. The ability to exploit this data to understand and predict the world around us is an asymmetric advantage for the Department of Defense (DoD).

Utilizing this data relies on computational algorithms running at a huge scale. Today, developers are limited in their ability to run these algorithms efficiently because they generally have to trade the efficiency of their algorithms with that of the available hardware architecture implementations. To combat this challenge, one solution is to design and fabricate application specific integrated circuits (ASICs)—customized hardware designed to maximize the runtime efficiency of a specific algorithm. However, ASICs typically cost hundreds of millions of dollars and take many years to develop. Once developed, they can perform exactly one class of computation because they were designed and specialized for specific application tasks. Because these systems are so specifically tailored and costly, their creation is often limited to the highest priority algorithms. For problems that cannot afford this level of investment, compute efficiency is sacrificed by implementing solutions such

as software on general-purpose processors or field programmable gate arrays (FPGAs). Often, this results in application implementations that are thousands of times worse than optimal.

The goal of the DARPA's Software Define Hardware (SDH) program was to build runtime-reconfigurable hardware and software that enables near ASIC performance without sacrificing programmability for data-intensive algorithms. Under the program, data-intensive algorithms are defined as machine learning and data science algorithms that process large volumes of data and are characterized by their usage of intense linear algebra, graph search operations, and their associated data-transformation operators. The SDH program aims to create hardware/software systems that allow data-intensive algorithms to run at near ASIC efficiency without the cost, development time, or single application limitations associated with ASICs. If successful, SDH will result in the ability to develop and run data-intensive, data-exploitation algorithms at very low cost, and, consequently, enable pervasive use of big-data solutions for a wide range of DoD applications including ISR, predictive logistics, decision support, and beyond.

The Challenge of Adoption and Transition The history of AI is tightly intertwined with the history of the development of computing machinery itself, and this story has been told in greater detail in a variety of other sources. What is new about the current moment is that the pressures of Moore's Law and the great advances in semiconductor materials have created new incentives and opportunities.

With rapid advances in hardware technologies the challenge has emerged as to how to assimilate these technologies and integrate them into use. Integrating new hardware technologies requires understanding and re-engineering the computational workflows associated with a given user-focused problem. Often these user-focused problems are complex, such as

- Accelerating an image processing pipeline, in which video and other media is processed at great scale and velocity to perform tasks such as automatic target recognition, tracking or object identification. Aspects of this computational workflow will include computing transformation on video streams under real-time constraints.

- Reducing power consumption for a mobile robot, in which the mobile robot is self-contained and constrained by its battery or power generation capability. In this context, such as for small autonomous air vehicles or cube satellites, there is a rigid power budget for the whole system that must be managed. This requires approaches that can vary performance based on power availability as well as hardware technologies that can compute only those specifically needed mathematical functions in as efficient a manner as possible.

- Enhancing the performance of a cloud environment, inside of which processing of data may be occurring at massive scales and under operationally-driven time constraints. An example of such a situation would be a system for real-time fraud analysis in a real-time network of financial transactions. In this scenario, data is large in both volume and velocity and the usefulness of the information produced depends on how quickly the fraud can be identified.

The challenge arises that the "Cambrian Explosion" in new hardware paradigms is creating a challenge for everyone in the field of AI, from educators to practitioners. The explosion of options is offering a vast new possible implementation fabric on which these AI and machine learning systems are trained and execute. These many new options now turn every intelligence system design task into an opportunity for customized and bespoke engineering solutions. Much like in the design of those cubesats or highly optimized UAVs, the hardware selections can influence the choice of implementation and even end-user experience. Engineers and system designers require some approaches to making such choices—and even just guidelines for when those choices are even worth exploring.

Results of this Study As part of the DARPA Software Defined Hardware Program, the team developed a methodology for benchmarking new hardware technologies against common mathematical workloads such as found in the pipelines for machine learning and graph traversal. The authors believe this process is generally

applicable to help practitioners assess the impact for any variety of emerging hardware capability. Further, the team's experience during the course of these studies has yielded several practical lessons learned and identified implications for the AI education community with respect to the exploding role of hardware for all matters of intelligent system building.

This is not a paper surveying emerging hardware technologies—these topics are covered extensively by other literature and sources. Rather the focus on this paper is on on the implications of these technologies to system designer and developers. These technologies offer potentially new means to architect computational solutions to notoriously difficult AI problems—in many cases enabling us to brute force certain problem elements that are difficult on traditional hardware.

Outline of this paper. This paper is organized as follow. Section 2 presents a highly condensed history of the relationship between AI and hardware advances. Section 3 discusses the current landscape of hardware technologies; Section 5 introduces the framework for application workflow redesign. This section documents our methodology for benchmarking new hardware paradigms; breaking down existing application workloads; and estimation of potential yields for applying new hardware to an existing problem. Section 7 presents a discussion of the implications of this program and these techniques. Lastly, Section 8 provides a summary and conclusions.

2 Historical Context

AI research has a sense of purpose and a direction that is embodied in this definition by Nils J. Nilsson, "Artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment" [28]. AI systems have three different paradigms that determine their performance as intelligent systems: algorithms, workflows and hardware. The software and hardware development has progressed in tandem for most of the AI history often driven by the workflows.

In 1945, John von Neumann outlined the architecture of a stored-program computer in the first hand written draft on EDVAC. An exact copy of this draft was published in 1993 [45]. This computer consisted of two system buses with each one having its own CPU and memory. Alan Turing proposed Automatic Computing Engine (ACE). In its first pilot, Turing referred to Neumann architecture as a strong source for understanding the nature and design of general purpose digital computer [4]. Although many computers have been built since the mechanical computer by Charles Babbage or the ABC computer without a CPU, to the modern high end ones, the first programmable computer is considered to be ENIAC invented by J. Presper Eckert and John Mauchly at the University of Pennsylvania in 1946. They also established the first computer company, Electronics computer company in 1949. Small-Scale Experimental Machine (SSEM), executed the first electronically stored program to find the highest proper factor of an integer using subtraction rather than division. EDSAC became the first stored-computer to run a graphical program, an implementation of tic-tac-toe. The workflows further influenced the design of these machines. Manchester Mark 1 became the first computer in 1949 to search for Mersenne prime numbers for nine hours without an error. These computers used a cathode ray tube for the display. The innovation in hardware continued and led to the first transistorized computer invention at MIT in 1956. Prior to that MIT also introduced the first digital computer with a magnetic core RAM and real-time graphics in 1955.

Very early on it was realized that for computational improvements we would need to focus on both hardware as well as software [7]. Fateman identified that the older generation machines had severe address space limitations and lack of flexible pointer manipulation facilities [9]. Rosenblatt proposed perceptron, a system that would recognize patterns analogous to a human brain [35]. Newell and Simon [27] identified that there is a push pull relationship between challenging AI problems and Algorithms that are used to solve them. They proposed the General Problem Solver (GPS), a computer program that is capable of simulating an approximation of human behavior in a narrow domain. The narrow domain is where AI performs the best. Prior to that multiple attempts have been made to solve a single problem [26, 11, 15]. Rosenblatt identified that although innovation in software is critical to AI development, that is not the only factor to understand why humans are so much smarter than computers [36]. According to Rosenblatt, human brain employs a more suitable computational architecture that

is combined with some constraints. They laid out a few workflows, such as, reaching and grasping,the mutual influence of syntax and semantics, interplay of the multiple sources of knowledge etc., to identify and present a parallel distributed processing architecture that would imbibe these constraints. Spinelli studied the reading-in and reading-out of the data affected by the neuronic model of the brain and proposed Omnium-Gatherum Core Content Addressable Memory (OCCAM) [41]. OCCAM was simulated on a small general purpose PDP-8 computer consisting of core memory of 4096 twelve-bit words. Fukushima proposed cognitron, a self-organizing multi-layered neural network and simulated it on a digital computer [10].

Limitations of a combination of software and hardware on specific workflows leads to the design of new architectures and creation of more efficient systems . Fahlman et al. [8] proposed Restricted Boltzman Machines (RBM), a new architecture for tasks that would otherwise perform inefficiently on other existing architectures at that time. However, the first strong relationship of AI software and hardware can be seen in the development of LISP [23] programming language by McCarthy. It is considered to be the first AI specific programming language and to execute it, Lisp specific machines were built. These machines, such as, Symbolics (3600, 3640, XL1200), Explorer, InterLisp-D etc. were the first commercial single-user workstations. Since Lisp none of the languages have been specifically designed for AI. The focus has been algorithmic efficiency and hardware, although, some of the languages like Python, R, Julia, Haskell, Prolog and Scala have been more popular than others among AI developers and researchers. Since the Lisp machines, there has been more focus on chip design and algorithms to perform tasks efficiently.

Lisp was designed for AI and need specific HW.

In 1965 Gordon Moore made a simple observation that the number of transistors on an Integrated Circuit (IC) would double every year, which he had to revise in 1975 to doubling of the transistors every two years. For many years it became not only a guiding force for the industry, but also a reliable method for calculating the future trends [37]. Following this observation world's first microprocessor Intel 4004 was introduced in 1971. Since then we have made tremendous amount of progress and today we have Intel's Xeon Platinum 9200 processors with 56 cores and AMD Ryzen with 64 cores. AI research has relied on generalized chips like CPUs and GPUs as well as some specialized chips like FPGA's, ASICs, and TPUs etc.

CPUs have dominated AI research involving logic or intensive memory requirements and sequential serial processing. They contain an arithmetic logic unit for logic and arithmetic operations, processor registers, and a control unit. Most of these CPUs are implemented on a single Integrated Circuit. Chips with multiple CPUs are called multi-core processors. A microprocessor chip may additionally contain a memory. In addition to the actual multi-core processors on a single chip, these processors can be multi-threaded to create additional logical CPUs. Although we have made a tremendous amount of progress with the speed and the the number of cores in the processor chips, they are still limited by the number of cores. This limit on the number of cores, reduces the effectiveness of a CPU to process large amounts of data in parallel. On the contrary Field programmable gate arrays (FPGAs) and Graphic processing units (GPUs) are designed for intensive parallel processing. Parallel processing has been a hallmark of AI research right from the beginning due to the limitation due to the "von Neumann bottleneck", the serial path used to move instructions and data between memory and the CPU, in serial machines [46]. Kitano and Higuchi demonstrated that in massively parallel machines, memory based translational models perform extremely well in real-time. They introduced IXM2 parallel associative processor that used associative memory for storing and processing memory-base. Toward the end of 1990s GPUs first appeared in the market.

GPUs were designed to handle multiple tasks in parallel, efficiently. More specifically they are designed to process large amounts of data used in rendering video and graphics and in image processing. They started as accelerators for rendering images on a display unit and have now become a competitive alternative to traditional CPUs for high performance computing [30] [34] [22] [3]. GPUs are good for parallel processing of a large number of arithmetic operations. The parallel computations enables the use of additional transistors and therefore achieve higher arithmetic intensity with the same number of transistors as a CPU [31]. These properties also help accelerate applications with repetitive workloads that are required to be performed repetitively and in rapid succession.

Traditionally, GPUs have produced very good results on different stream operations like, map, reduce, scatter and gather, stream filtering, sort, and search [31]. Recently, Deep Neural Networks have formed the bedrock of AI applications [20] and GPUs have become the hardware of choice for such architectures. While GPUs provide a great support to these deep learning AI algorithms, the algorithmic superiority comes at high computational and

memory costs that pose a significant challenge to the hardware platforms executing them [42]. Although, it has been seen that GPUs have performed really well with deep neural networks due to their superior floating point computations, for some customizable data sets Field programmable gate arrays (FPGAs) are more suited [29].

FPGAs are integrated circuits that can programmed and provide an alternative to AI specific applications [24]. FPGA circuitry is comprised of programmable logic circuitry. This implies that they can be programmed for a specific task. This enables design of an FPGA specifically for a particular algorithmic architecture such as a deep neural network. This flexibility of the hardware helps test new AI algorithms. FPGAs have some very useful advantages. Low latency of FPGAs enables faster execution of real-time applications like speech recognition, video streaming and motion recognition. They are also cost-effective since they can be designed and tested at a quick speed and adapted to include more than one AI specific task. FPGAs are also very energy efficient and can be fine tuned to be optimized for frameworks like PyTorch and Tensorflow. However, the flexibility of FPGAs can also be its bane due to the difficulty of programming it. Another significant AI chips are application-specific integrated circuits (ASICs).

ASICs are specialized chips for AI in contrast to the generalized chips like CPUs and GPUs. These have been considered to be the most efficient chips for specific Deep Neural Network computing tasks. They are being used for inference as well as training. They are hardwired for a specific algorithm. One of the examples of such a customized ASICs' for accelerating machine learning algorithms is Google's Tensor Processing Unit (TPU). Due to the specificity of ASICs to an algorithm or an AI task, they maybe easier to design but that is also a reason for them to become obsolete since new algorithms would require a different ASIC. The goal is to design a chip that would be optimized for specific algorithms and at the same time can be applied to solve a much broader problem. One of the measures is to build AI hardware that is as optimized as an ASIC for a single task but at the same time highly configurable so that the same chip can be used for multiple tasks without the loss of optimization.

3 The End of Moore's Law

Traditionally AI has been viewed from the perspective of algorithm development, however, it encompasses a full data life cycle with the focus on process workflows.

AI algorithmic development has been tightly linked with workflows and hardware. Hardware like FPGAs and ASICs has been tuned toward specific algorithms generally focused on speed of execution with some nod towards energy efficiency afforded by the shrinking of hardware process - currently hovering around a 3nm process.

We are currently approaching the time when simply shrinking a die to get better efficiencies and speed won't be possible. Transistors in 5nm processes are approaching 10 atoms. When training for some large Neural Net models is consuming around 900MWh of energy we need to focus on the efficiency of the chips we build. GOPS/Watt is the new Moore's law.

GOPS?

"New Moore's Law"

The goal of the Software Define Hardware (SDH) program is to build runtime-reconfigurable hardware and software that enables near ASIC performance without sacrificing programmability for data-intensive algorithms. Under the program, data-intensive algorithms are defined as machine learning and data science algorithms that process large volumes of data and are characterized by their usage of intense linear algebra, graph search operations, and their associated data-transformation operators.

3.1 Measuring Success

In Spring/Summer 2019, DARPA and subcontractors curated a set of 7 benchmark workloads for SDH benchmarking.

The workloads were chosen to span multiple kinds of computation:

- Dense compute: The kinds of dense linear algebra and elementwise operations commonly found in neural network-based machine learning applications.

- Sparse compute: The kinds of sparse linear algebra or graph traversal operations commonly found in graph analytics applications.
- Mixed compute: Workloads that include both "dense" and "sparse" computational kernels.

The workloads were intended to serve a dual purpose:

- **Performance Benchmarking**: SDH performers were instructed to implement the workloads on their novel software / hardware platforms. They were permitted to do any algorithmic or platform-specific optimizations to attain maximal performance.
- **Programmability Benchmarking**: The SDH BAA states that SDH systems should "enable near ASIC performance without sacrificing programmability". To measure "programmability", UMD ARLIS and Parenthetic conducted a study in Summer 2019 in which they recruited programmers via online coding competition websites to try to implement a subset of the 7 workloads using SDH APIs and simulators.

The primary SDH program metric as defined in the BAA is "compute efficiency (in terms of giga-operations per watt (GOPs/Watt)". The BAA also described "programmability" targets, but to our knowledge those are no longer being pursued.

Table 1: "Program metrics and target" from SDH BAA

	vs CPU	vs ASIC	vs ASIC (sparse math, search, graphs)	Programmability (OBE)
Phase 1	100-300x better	within 50x	1x	within 10x
Phase 2	100-300x better	within 10x	2x better	within 3x
Phase 3	500-1000x better	within 5x	8-10x better	1x

It's unclear whether the SDH BAA contained a typo or whether SDH program management decided to change the target metric after the fact. However, the metric that has been used in all SDH experiments to date is in fact "operations per second per watt (OPS/Watt)". (Note the capital S.)

"Operations per watt (OPs/Watt)" reduces to

$$ops * seconds / joule$$

which would mean – assuming fixed ops and energy usage – systems would be incentivized to maximize runtime.

On the other hand, "operations per second per watt (OPS/Watt)" reduces to

$$(ops/seconds) * (seconds/joule) = ops/joule$$

which removes the incentive for long runtime. Note that it still does not explicitly incentivize fast runtime. This is a more sensible optimization target. For all further discussion, we will use GOPS/Watt, as has been used in the SDH program to date.

OPS per second per watt is the more sensible target because it disincentivizes long run times.

3.2 AI Workflows

We discussed these AI specific hardware, and now it is time to present few representative AI workflows. Although this is not an extensive list of workflows but it is broad enough to include a wide spectrum of AI. These workflows are as follows:

- **ConvNet**: Convolution Neural Networks (CNN) [21] are a class of neural networks that help understand the spatial hierarchies. It employs the use of linear operations that are independent and shift invariant. CNNs are specialized in processing data that has a grid like structure, such as matrix to matrix multiplication 1. Some of the dominant applications where CNNs have been highly successful, include, time-series

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)}$$

Figure 1: An example of matrix-matrix computational kernel.

data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image image data, which can be though of as a 2-D grid of pixels [12]. A representative CNN is shown in Figure 2.

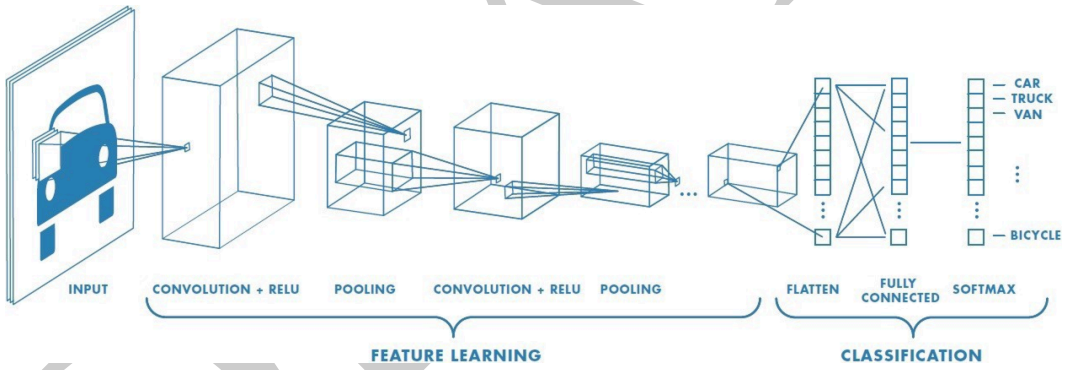

Figure 2: A typical multi-layer CNN network.

- **GraphSAGE**: It is a general inductive framework that efficiently generates node embeddings for each node of a large graph by leveraging node feature information [14]. It learns low dimensional embeddings of nodes in large graphs that is very useful for a wide variety of prediction tasks. A visual illustration of the GraphSAGE sample and aggregate approach as presented in the paper by Hamilton et al. is shown in Figure 3. The node embeddings can be fed to machine learning algorithms downstream wherein they help node classification, clustering and link prediction [13, 33, 43].

- **ip-NSW**: Inner-product navigable small world (ip-NSW) is a 'hierarchical similarity graph'-based retrieval algorithm which outputs approximate solutions to Maximum Inner Product Search (MIPS) [1]. MIPS is a computational bottleneck in machine learning applications. Historically graph similarity measurement has been associated with the metric scenarios such as the vector based representations and ip-NSW is an approach extended to non-metric scenarios. Its key applications include, image search and retrieval, document search, comparison and social media analysis, and recommender systems.

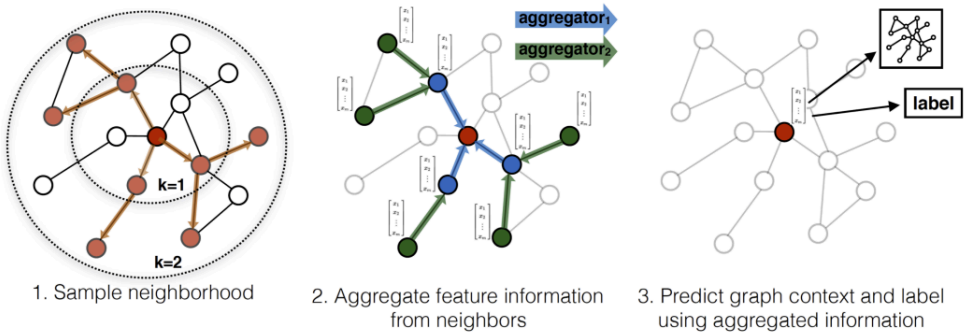

Figure 3: Visual illustration of the GraphSAGE sample and aggregate approach [14].

- **Parallel Local Graph Clustering**: Local graph clustering algorithms diffuse probability mass from a seed vertex and return an approximate PageRank vector [38]. Parallelization of the local graph algorithms is achieved by iteratively processing subsets of the vertices and their edges until a termination criteria is met. Each iteration process is parallelized. Some of the popular local graph algorithms that have been ietartively parallelized, include, Nibble algorithm of Spielman and Teng [39, 40], the PageRank-Nibble algorithm of Andersen et al. [2], the deterministic heat kernel PageRank algorithm of Kloster and Gleich [17], and the randomized heat kernel PageRank algorithm of Chung and Simpson [5].

- **RecSys**: The deep autoencoder based recommender system (RecSys) predicts ratings. it takes a list of items a person has liked and predicts a score of how likely a person is to like all other items. An autoencoder is a neural network that implements two transformations: encoder, $\mathbb{R}^n \rightarrow \mathbb{R}^d$ and decoder $\mathbb{R}^d \rightarrow \mathbb{R}^n$ [18]. The goal of the autoencoder is to obtain d dimensional representation of data such that an error measure between x and $f(x) = decode(encode(x))$ [16]. Figure 4 depicts a five layer autoencoder network. Beyond traditional applications like recommending the next YouTube video or Netflix movie,

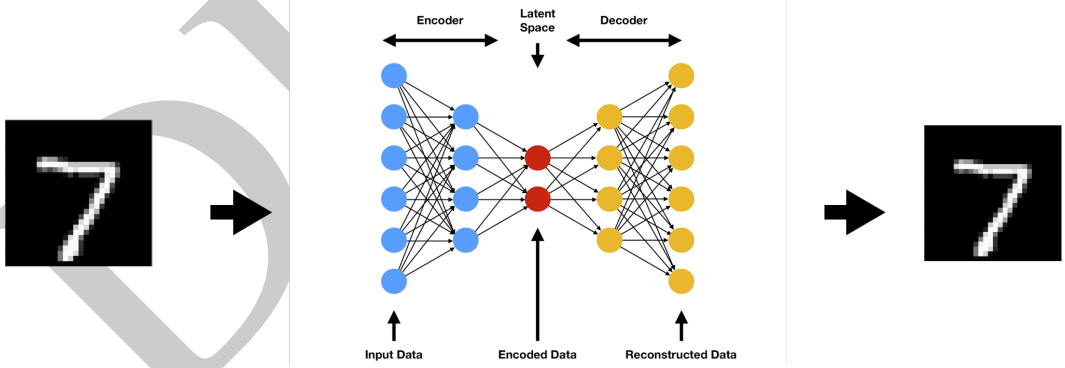

Figure 4: Encoder-decoder representation of an autoencoder.

recommender systems are used in many analytics pipelines such as recommending next target for analysts to examine based on historical data, and fault-/anomaly-detection.

- **Sinkhorn WMD**: Word Mover's distance (WMD) is used to measure distance between two documents. It measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to be modified to arrive at the embeddings of the second document [19]. The word embeddings are computed using word2vec [25]. The WMD distance is measured using the Sinkhorn-Knopp algorithm [6]. An example of using Sinkhorn-WMD is shown in Figure 5.

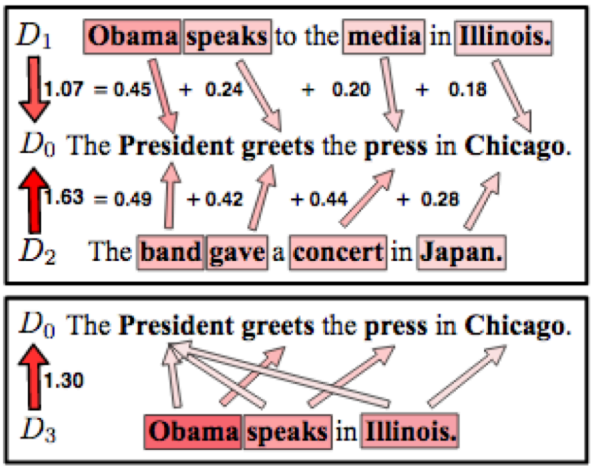

Figure 5: An illustration of finding Sinkhorn-WMD metric.

3.3 AI Workflow Kernels

Kernel Results Smaller algorithms, here referred to as "kernels" were selected to allow implementation on FPGA hardware (to predict ASIC performance). FPGA measurements and ASIC estimates from the University of Dayton Research Institute are included here.

A key problem in evaluating hardware solutions is that the hardware doesn't exist in a configuration that supports large scale workflows. It was hoped that some components, from the workloads listed above, might be extracted to run, either in simulation or emulation or in actual hardware.

DARPA and ARLIS identified key algorithms that would lend themselves to evaluation on FPGA devices.These kernels were identified as candidates:

- FFT – used in many DSP applications
- Dense matrix multiplication – example General Matrix Multiply (GEMM) from BLAS
- Sparse matrix multiplication – example SpGEMM
- 2D Convolution
- Shortest Path algorithm – SSSP, Dijkstra's algorithm
- Auction Algorithm – Linear Assignment problem

4 Baselines

4.1 CPU/GPU baseline

Both workflows and kernels were implemented on CPU and GPU

4.2 FPGA and ASIC baseline

The SDH workflow kernels considered for the DARPA SDH program to evaluate hardware are:

- **Fast Fourier Transform (FFT)**: This kernel employs a parallel FFT. The parallelism is achieved on a single FFT as opposed to processing multiple FFTs at once. The implemented FFT cores supported FFT sizes from 1K to 64K.

- **Dense Matrix Multiplication**: For matrix multiplication, an energy efficient FPGA is used. An on-chip memory design is shown in Figure 6. This design consists of a linear systolic array which minimizes the

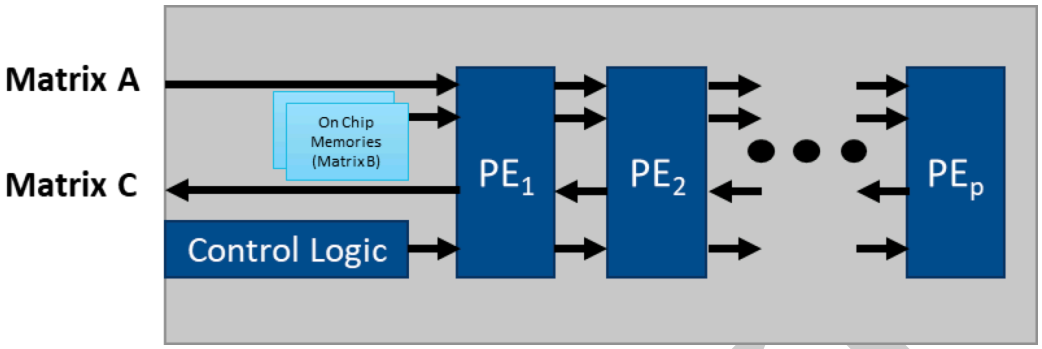

Figure 6: On-chip memory design.

number of long interconnects which can lead to improved energy efficiency. Each processing element only communicates with its nearest neighbor. This design also leverages both parallel and pipelined processing. Parallel processing can be achieved by increasing the number of resources in each processing element which leads to higher throughput. The systolic design allows for pipelining which increases the resource utilization.

N $\times$ N matrix multiplication requires only N hardened DSP floating point blocks for the previous design. This DSP reduction will allow the SDH data sizes to fit within a Stratix 10 device. Theoretically, the largest matrix multiplication that can be done is 5760 $\times$ 5760. This does not consider the on-chip memory requirements for each processing element. A more detailed view of a processing element can be seen in Figure 7. Using the architecture in Figure 7, N $\times$ N matrix multiplication can be completed in n2+2n cycles with each processing element using 1 DSP. Each processing element also consists of 3 IO ports, 4 registers, and 2 FIFOs of N word capacity.

To improve throughput, more resources can used for each processing element as shown in Figure 8. The throughput is increased by a factor of r, the number of processing elements is reduced by a factor of r, while the number of DSP for each processing element increases by a factor of r2.

- **2D Convolution**: 2D Convolution was implemented using Intel's DSP builder tool. Early literature searches showed that depth-wise convolution can perform the standard convolution algorithm with significantly fewer operations as illustrated in Figure 9. The convolution algorithm was implemented as a systolic array. However, it was IO limited as only 4-5 convolution cores exhaust the IO bandwidth. To overcome this problem an MX variant of Stratix 10 FPGA with up to 16 GB of high band-width memory as shown in Figure 10.

- **Sparse Matrix Multiplication**: Sparse matrix multiplication was implemented with some assumptions up-front. The first being that the A matrix is in compressed sparse row (CSR) format. Second being that the size of the A matrix is (<1024, 10K to 100K). Third, the matrix B is stored in memory because it usually does not change. And last, the B matrix is of size (10k-100k,32-15). Sparse matrix multiplication is depicted below in Figure 11. The number of cycles until complete is dependent upon the sparsity of the matrix. If there are more non-zeros than rows, the latency is approximately 1.0005 the number of non-zeros. If more rows than non-zeros, latency is 2.5x the number of rows. The FPGA design that was implemented using DSP Builder is shown in Figure 12.

- **Dijkstra Algorithm**: Dijkstra's algorithm solves the shortest path problem. The key to parallelizing Dijkstra-like algorithms is eliminating the von Neumann bottleneck. In this research, we have achieved

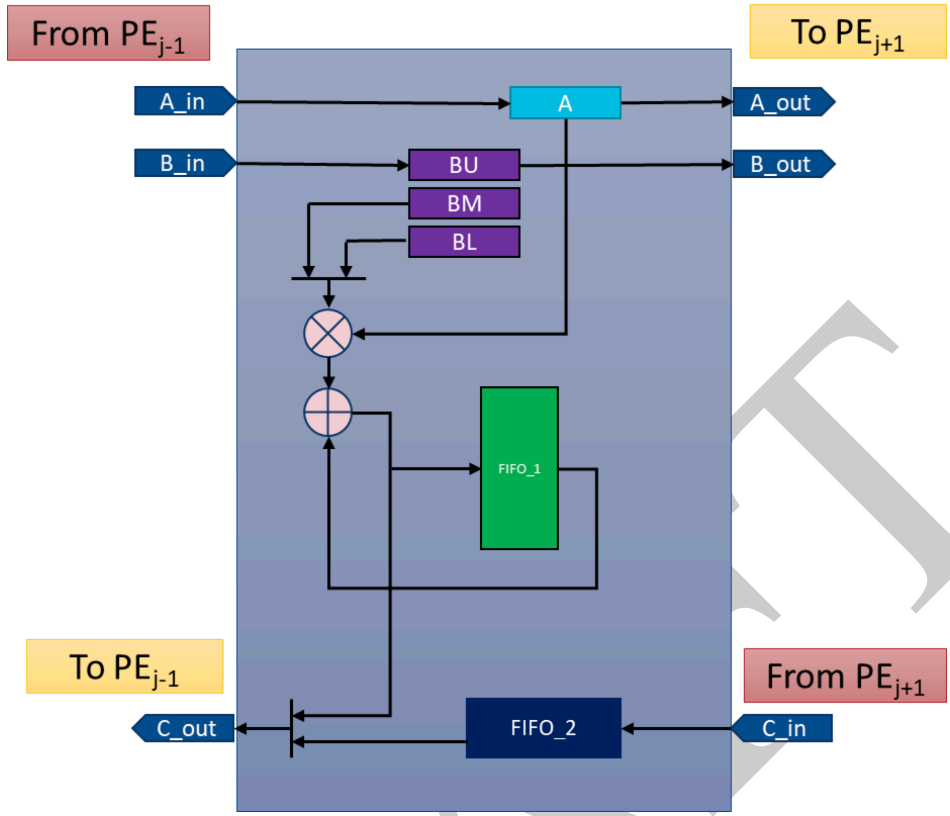

Figure 7: Processing Element Architecture.

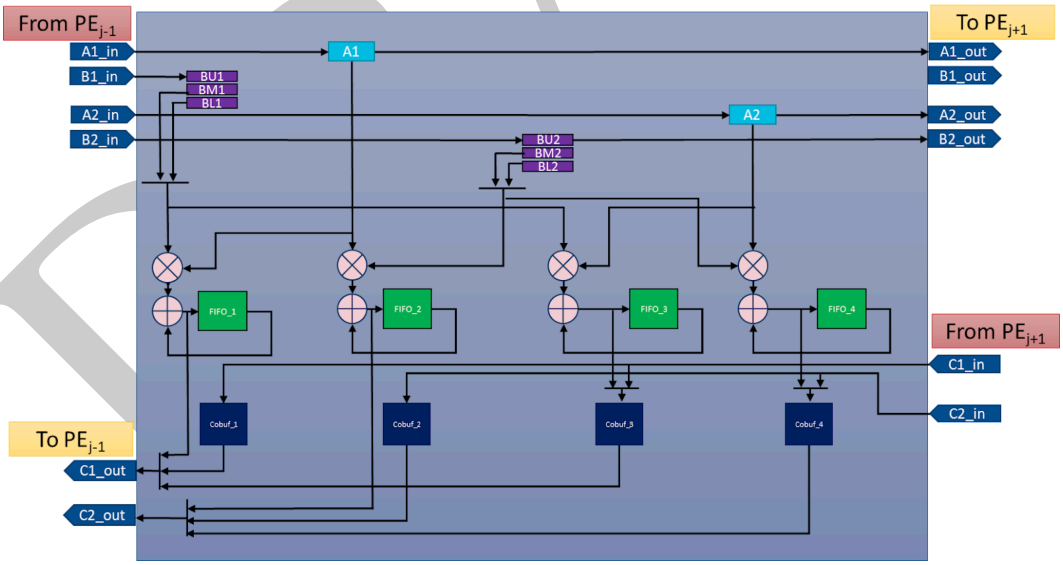

Figure 8: Matrix Multiplication using Parallel Architecture.

This is Cou

that by designing a Processor In Memory (PIM) RAM and PIM Set that take the work of finding unserviced nodes and determining the best solution from the kernel. This architecture is scalable and can process graphs with many millions of nodes. Unless otherwise noted, the words kernel and Processing

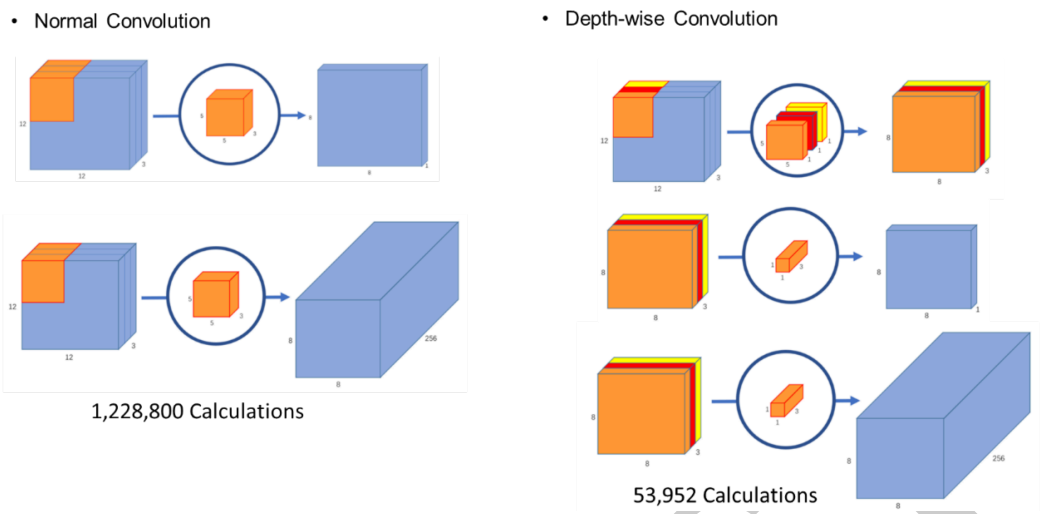

Figure 9: Standard Convolution vs Depth-Wise Convolution.

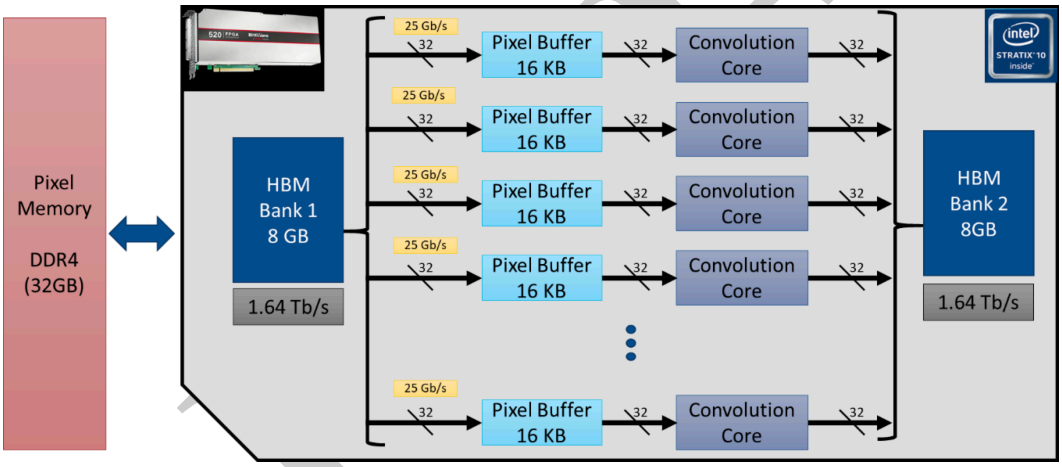

Figure 10: Convolution using Stratix 10 with High Band-Width Memory.

Element (PE) are used interchangeably. A kernel or PE is the ASIC version of a CPU thread.

This algorithm is implemented in VHDL. This is different from straightforward C-like implementations. C-like implementations mean single threaded implementations in a C-like language (C++, Java, Python, etc.) on a Von Neumann architecture. C-like implementations sometimes use concurrency to give the illusion (and in some cases, the substance) of parallelism. This VHDL implementation offers true parallelism. The difference also matters for proving algorithm correctness. The difference also demands a different memory model than C-like languages have relied on. C-like implementations can sometimes achieve an 8x speedup using 8 cores. But 8 cores would be small by VHDL standards. Because the number of threads in a VHDL can grow far larger than the number of threads in any C-like language (per cm^2), Processing-In-Memory (PIM) is not merely the next horizon for parallelization; it is a necessity.

- **Auction Algorithm**:The auction algorithm solves the assignment problem. Many instances of the kernel described here can be parallelized in a scalable architecture which increase throughput beyond what CPUs can achieve.

  Auction algorithms solve Computer Science & Engineering problems analogous to a real life auction. In

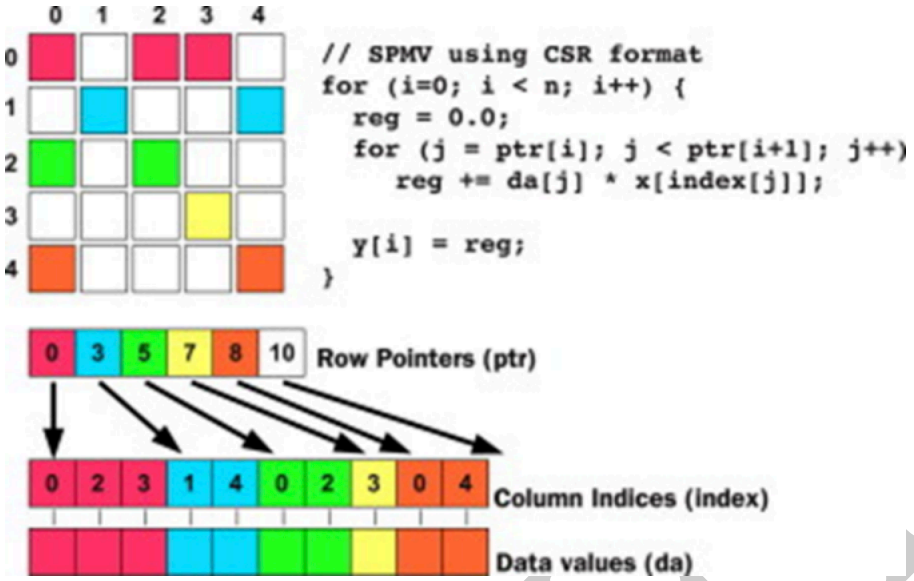

Figure 11: Sparse multiplication using CSR Format.

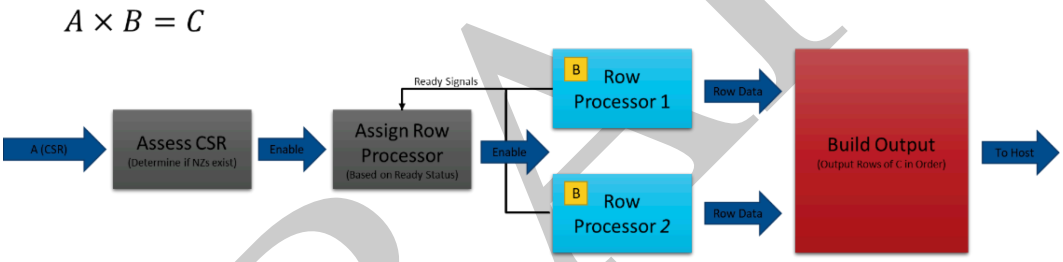

Figure 12: Sparse Matrix Functional Design.

an auction, there are buyers and sellers, users and assets. Auction algorithms match each buyer with a resource according to how useful a resource is to a buyer. Auction algorithms are useful for optimizing a wide variety of problems that involve matching up some user with some resource. For example, passengers with taxis, servers with clients, or soldiers with satellites. Auctions provide a solution to complex, multi-variate optimization problems.

There are two fundamental approaches for implementing the auction algorithm:

- Jacobi Architecture: 1 actor bids on every resource
- Gauss-Seidel (GS) Architecture: Every actor bids on 1 resource.

GS makes for a more efficient FPGA implementation. At their core, auction algorithms are simple. They require:

- A merit function which decides how valuable an asset is to a particular buyer, and
- A list of buyers,
- A list of sellers.

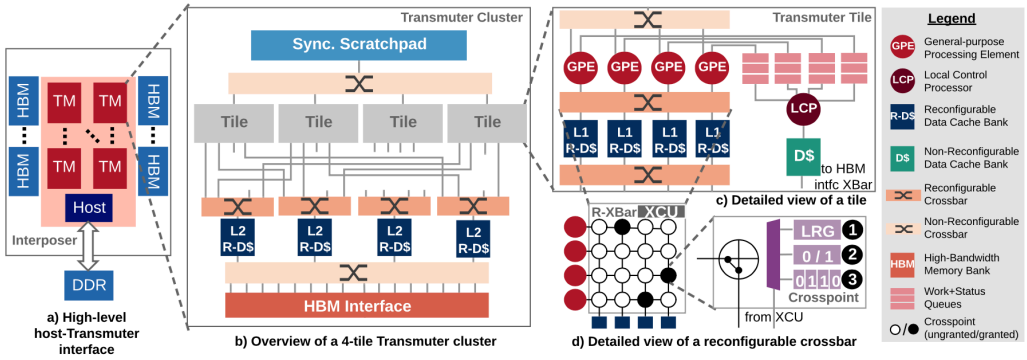

Figure 13: Transmuter Architecture.

An auction algorithm outputs a list of tuples, such that for each tuple, (Actor, Resource), the resource is the most valuable among available resources to the actor – with the constraint that actors who have already won an asset do not bid again. This is an important difference between auction algorithms and real life ones, where a single organization can buy all of a given resource. In a computer auction, we typically do not let one server to snatch up all the clients or vice versa.

5 A Framework for Application Redesign

In order to build Software Dependent Hardware (SDH), three performers in collaboration with ARLIS and DARPA proposed three solutions. The purpose of this collaboration is to create a balanced ecosystem of configurable architecture, runtime and software that outperforms off-the-shelf commercial CPUs and GPUs. This new design is within an order of magnitude of the performance of a custom ASIC on dense computational kernels, while providing better than ASIC performance on sparse workloads, or workloads that exhibit phases of behavior that can be rapidly reconfigured to obtain near optimal efficiency. These solutions are:

5.1 Transmuter: A Reconfigurable Computer

This research has been performed at the University of Michigan, University of Edinburgh, University of Arizona, and ARM. The transmuter architecture is a highly introspective design so software can monitor and adapt. It comprises of a massively parallel fabric of simple cores in a tiled architecture [32].

The full transmuter is displayed in Figure 13.

A two-level hierarchy of crossbars and on-chip memories allows for fast reconfiguration of the on-chip memory type (cache/scratchpad/FIFO), resource sharing (shared/private) and dataflow (demand-driven/spatial). There are three different Transmuter configurations:

- Shared Cache (Trans-SC): It uses L1 and L2 shared caches. The cores to the L1 memory banks are connected by crossbars which connect the L2 banks to the tiles. This resembles a many core system with a larger compute-to-cache ratio and it is efficient for regular accesses with high inter-core reuse.

- Private Scratchpad (Trans-PS): It retains L2 as cache and reconfigures the L1 cache banks into SPMs. The crossbars are reconfigured to privatize the L1 SPMs to their corresponding cores and L2 SPMs to their corresponding tiles, respectively. This is an ideal configuration for workloads with high intra-core but low inter-core reuse of data, prone to cache-thrashing. The caching of secondary data, such as spill/fill variables is enabled by the private L2 banks.

- Systolic Array (Trans-SA): It is suitable for data parallel applications where the work is balanced between the cores. It employs systolic connections between the cores within each tile.

workload	Platform	Execution time (s)	GOPS	GFLOPS	GOPS / W	GFLOPS / W	Total Energy (J)
graphsage	Transmuter (Phase-II)	19.375	5.30	0.41	165.83	12.86	0.6195
	Intel Xeon (CMU)	119.2	N/A	38.0	N/A	0.25	18356.8
		(6.15x)		(0.01x)		(51.4x)	(29632x)
ipnsw	Transmuter (Phase-II)	1.05	2.82	0.33	85.72	10.15	0.0346
	Intel Xeon (CMU)	16.5	0.5	N/A	0.005	N/A	1560.9
		(15.7x)	(5.64x)		(17144x)		(45113x)
recsys	Transmuter (Phase-II)	1091.04	6.23	1.43	197.49	45.32	34.4
	Intel Xeon (CMU)	188.6	13	N/A	0.08	N/A	29044.4
		(0.17x)	(0.48x)		(2469x)		(843x)
sinkhorn_wmd	Transmuter (Phase-II)	1.620	0.82	0.12	26.99	4.04	0.049
	Intel Xeon (CMU)	1.987	11	N/A	0.11	N/A	200.7
		(1.23x)	(0.07x)		(245x)		(4096x)
LGC/ pr_nibble	Transmuter (Phase-II)	0.708	3.14	0.07	99.45	2.12	0.0203
	Intel Xeon (CMU)	42.7	0.9	N/A	0.01	N/A	4073.6
		(60.31x)	(3.49x)		(9945x)		(200670x)
LGC/ISTA	Transmuter (Phase-II)	14.5	7.9	1.6	125.5	26.3	0.91
	Intel Xeon (CMU)	15.4	2.87	N/A	0.03	N/A	1470.7
		(1.07x)	(2.75x)		(4180x)		(1615x)
convnet	Transmuter (Phase-II)	13.86	3.44	0.83	135.49	32.50	0.35
	Intel Xeon (CMU)	0.07	70	N/A	0.4	N/A	12.39
		(0.005x)	(0.05x)		(338.73x)		(35.40x)

Table 2: Transmuter benchmark summary.

Tiles are made of multiple ARM M-class cores as general-purpose processing elements (GPE). The cores are programmed using the TransPy software stack (TA2). The afrchitecture of Transmuter is dsiplayed in Figure It has a configurable interconnect and memory with a reconfigurable data cache (R-DCache) and crossbar (R-XBar). It also has register-to-register inter-core communication that avoids SRAM accesses by passing data directly between registers of adjacent cores. Reconfiguration in transmuter is cheap and most configuration changes take 1-2 cycles. The workflow performance of Transmuter is presented in Table 2

5.2 DECADES Platform

In Decades platform architecture computations are mapped onto core tiles or accelerator tiles. It is a vertically integrated approach that enhances data locality, optimizes spatial mapping of threads, and enables in-memory computing. It contains a reconfigurable interconnection network with in-memory computing and embedded machine learning accelerators. The chip prototype contains in-memory computing hardware with a scalable full-system simulation with a multi-FPGA infrastructure. The DECADES chip contains 2.2B transistors with 60 OS capable RISC-V cores, 24 accelerators, 23 intelligent storage tiles with embedded FPGA.

The DECADES software stack contains flexible frontend API to handle multiple frontends, and flexible backend API to handle multiple backends. It is highly programmable with automated parallelization, and decoupling. It also allows for flexible design cycle through a simulator, MosaicSim. This simulator supports multiple clock frequencies, and inter-tile communication. MosaicSim also supports asynchronous memory requests used by MAPLE decoupling, multi-level cache hierarchy, lightweight DRAM modeling with simpleDRAM. It has an LLVM-based front end that enables flexible programming models, APIs, and custom/specialized instructions. It abstracts away ISA / hardware compexity. The workflow performance of Decades is presented in Table 3.

workload	Execution time (s)	GOPS	GOPS / W	Total Energy (J)
graphsage	14.768	1248.079	807.849	22.816
ipnsw	0.018	486.585	103.170	0.086
recsys	566.029	228.109	1051.180	122.830
sinkhorn_wmd	3.100	1233.276	32.561	117.414
pr_nibble	0.0035	520.792	90.058	0.02
LGC/ISTA	9.467	883.968	141.331	59.213
convnet	122.76	238.967	244.656	119.9

Table 3: Decades benchmark summary.

workload	Execution time (s)	GOPS	GOPS / W	Total Energy (J)
graphsage	79.70	56.76	29.89	151.35
ipnsw	0.038	7.37	1.28	0.222
recsys	40.61	67.55	28.77	95.36
sinkhorn_wmd	0.134	14.98	12.29	0.163
pr_nibble	0.123	1.91	1.38	0.170
LGC/ISTA	1.08	18.00	9.84	1.98
convnet	12.05	545.14	375.90	17.48

Table 4: HammerBlade benchmark summary.

5.3 Hammerblade

Hammerblade is a data-processing architecture. Each node in this architecture is a single system-on-chip. Multiple nodes are connected with high bandwidth links. Each node is architected from an array of tiles connected by a 2-D mesh network, called the manycore accelerator network, and is attached to a reconfigurable high-bandwidth memory system and the I/O system. Tiles within a HammerBlade node are processing elements, memory, or IO interfaces.

HammerBlade processing tiles fit into one of the three categories: throughput-optimized RISC-V tile, a Linux-capable RISC-V core, or a specialized accelerator. The throughput-optimized cores in this system are called Vanilla-5 (V5) cores. Each V5 core supports floating-point operations, contains a scratchpad for local data storage, and an instruction cache.

HammerBlade accelerators are diverse, flexible and performant. There is an effort through this program to develop a CGRA fabric for accelerating large sequences of branch-free code with instruction-level parallelism and a sparse-tensor architecture for accelerating sparse linear algebra operations. Together with dense-tensor accelerators these will provide efficient computation for high-value workloads in this program.

Cache tiles provide the interface to external memory. The cache is reconfigurable and introspective. These tiles are typically located on the top and bottom edge of a tile array and are called column-caches. These column cache tiles are connected to memory controllers that interface to multiple parallel memory channels of a DRAM-based memory system system - high bandwidth memory (HBM), DDR4, GDDR5, etc. Applications will be able to measure performance and reconfigure the memory system parameters depending on the demands of the phase of an application.

A collection of tiles is called a Pod, and a HammerBlade node contains multiple Pods. Pods impose a hierarchy in communication: Intra-Pod communication will have lower latency and higher-bandwidth than inter-Pod communication. Likewise, inter-Pod (Intar-Pod) communication will have lower latency and higher bandwidth than inter-Node communication. The Pod dimensions, distribution of tile types, cache sizes, number of memory DRAM interface channels, and mapping from channels to caches is flexible and an active area of research. The workflow performance of Hammerblade is presented in Table 4.

6 Benchmarking

6.1 Methodology

The SDH CPU baseline machine was an Intel E7-8894 v4 / Broadwell, as specified in the SDH BAA. This specific CPU was specified in the SDH BAA but proved difficult for DARPA to procure. In the end, Intel provided the SDH benchmarking teams with access to one of their in-house instances.

... Note: Intel Advisor didn't work for the graphsage workload, so they used VTune to measure FLOPS instead.

After implementing and optimizing the four workloads as described above, we measured

- Runtime of region-of-interest (ROI), using std::chrono::high_resolution_clock
- Operation counts (integer and floating point), using a combination of manual op-counting and the Intel Advisor tool
- Average power consumption of the ROI, using ARM MAP

Intel Advisor and ARM MAP were chosen to keep the benchmarking methodology as similar to CMU-SEI's process as possible.

Power and runtime measurements are averaged over 64-256 runs of the workload.

We benchmarked the workloads using 1, 2, 4, 8, 16 and 24 threads on one socket of the Intel SDH CPU baseline machine. (Threads are pinned to a socket using MP_HW_SUBSET=1s).

The benchmarks were run with the Intel baseline machine in the "powersave" power governor mode. This means that the runtimes we report may be slightly inflated, compared to benchmarking with the "performance" governor. (In particular, the first iteration tends to be much slower than subsequent iterations). However, Intel was unable to change the power governor mode in time for our benchmark runs.

We validated the output of Intel Advisor's op counting procedure by

- Compiling our code with compiler optimization disabled
- Instrumenting our code with manual op counting functions
- Reverse-engineering what Advisor is counting as an "op".

With this procedure, we were able to reproduce Advisor's op counts to within a ~5% margin. This gives us confidence to use Advisor when -O3 compiler optimization is turned on, and manual op counting is less straightforward.

6.2 Benchmarking Results

A benchmark is defined as a "Standard tool for the competitive evaluation and comparison of competing systems or components according to specific characteristics, such as performance, dependability, or security [44]". The ARLIS T&E team has researched and benchmarked CPU and GPU implementations of workflows and kernels in order to baseline existing microprocessor performance. The benchmark results for workflow performance is provided in Table 5 in GOPS/watt. Smaller algorithms, referred to as "kernels", were selected to allow implementation on FPGA hardware (to predict ASIC performance). FPGA measurements and ASIC estimates from the University of Dayton Research Institute (UDRI) are included here in Table 6.

	convnet	graphsage	recsys	LGC (ista)	LGC (prn)	Sinkhorm WMD	ipnsw
CPU	0.4	0.25	0.08	0.277	0.102	0.613	0.279
GPU	11.18	2.46	0.00	0.711	0.245	3.29	

Table 5: Workflow Performance Measurements.

UDRI utilized different devices and speeds for their FPGA and ASIC estimates. The table below lists the various devices 7.

	FFT	2D Convolution	Dense MatMul	Sparse MatMul	Dijkstra	Auction
FPGA	347	540	610	2.5	0.430	0.290
ASIC	700	1100	1200	5	0.860	0.580

Table 6: Kernel Performance Measurements.

Kernel	Device	FMAX(FPGA) (Mhz)
FFT	Stratix10	750
2D Convolution	Stratix 10 MX	750
Dense MM	Stratix 10	600
Sparse MM	Stratix 10	182
Dijkstra	Kintex 7	100
Auction	Kintex 7	100

Table 7: FPGA device and speed.

6.2.1 Workflow Evaluation / Performance

The workflows were evaluated for performance on CPU and GPU using PyTorch implementations.

ConvNet: Cifar-10K, 32× 32 pixel images were trained to 95% accuracy in five epochs. Baseline Python runs in about

- 5s on P100 gpu
- 40s on CPU (8 thread, Intel Xeon, CPU E5-2698 v4 2.20 GHz.

The implementation results are presented in Table 8

graphsage is evaluated on a social networking data set to predict the exact age. It would take 2-3 minutes on an Intel Xeon E5-2698 v4 2.20GHz CPU with 8 threads and 30 seconds on a P100 GPU. The performance measurements are displayed in Table 9.

RecSys Recommender systems have also been evaluated in Pytorch on CPU and GPU. The performance results are in Table 10.

LGC Three implementations were evaluated:

- Baseline Python - written in numpy / scipy
- Optimized Python - written in numpy / scipy / numba
- Optimized C++ - written with multithreading using openmp

The performances are presented in Tables 11 and 12

SinkhornWMD is evaluated using baseline Python written in NumPy / Scipy and optimized Python written in Numpy / SciPy / Numba, and optimized C++ written with multithreading using OpenMP. The baseline Python implementation on 8 threads runs in about a minute, optimized Python version runs in < 1 second, and optimized C++ runs in < 0.1 milliseconds. These results are presented in Table 13.

ipNSW is evaluated by only using the query section of the algorithm. Construction of the indexing can be handled offline, while scalability and performance of the query kernel (returning similar objects) is far more critical. The following three implementations were evaluated on Music100 dataset:

implementation	threads	μsec	Watts	GOPS	GOPS watt
pytorch CPU	24	27480000	177	70.00000	0.40000
pytorch GPU	V100	3980945	119.271	1333.60295	11.18128

Table 8: ConvNet Performance Measurements using PyTorch implementation.

implementation	threads	μsec	Watts	GOPS	GOPS watt
pytorch CPU	24	98810000	154	38	0.25
pytorch GPU	V100	41649533	44.446	109.67710	2.46765

Table 9: Graphsage Performance Measurements using PyTorch implementation.

implementation	threads	μsec	Watts	GOPS	GOPS watt
pytorch CPU	24	151160000	154	13	0.08
pytorch GPU	V100	22369443	136.218	0.0	0.0

Table 10: RecSys Performance Measurements using PyTorch implementation.

implementation	threads	μsec	Watts	GOPS	GOPS watt
C++/OpenMP	1	664621	94.266	4.20691	0.04463
C++/OpenMP	2	433426	102.941	6.45093	0.06267
C++/OpenMP	4	242072	112.277	11.55028	0.10287
C++/OpenMP	8	127945	123.736	21.85314	0.17661
C++/OpenMP	16	75461	143.983	37.05225	0.25734
C++/OpenMP	24	56210	179.548	49.74204	0.27704
CUDA/Thrust	V100	22998	170.79	121.57579	0.71184

Table 11: LGC ISTA Performance Measurements.

implementation	threads	μsec	Watts	GOPS	GOPS watt
C++/OpenMP	1	138739	95.253	1.25415	0.01317
C++/OpenMP	2	77333	104.343	2.25001	0.02156
C++/OpenMP	4	41192	114.701	4.22412	0.03683
C++/OpenMP	8	22520	130.252	7.72647	0.05932
C++/OpenMP	16	12773	158.418	13.62248	0.08599
C++/OpenMP	24	9225	184.926	18.86179	0.10200

Table 12: LGC PRN Performance Measurements.

implementation	threads	μsec	Watts	GOPS	GOPS watt
C++/OpenMP	1	319689	95.962	8.18921	0.08534
C++/OpenMP	2	175177	104.402	14.94488	0.14315
C++/OpenMP	4	99813	111.428	26.22905	0.23539
C++/OpenMP	8	55574	120.869	47.10836	0.38975
C++/OpenMP	16	34570	138.455	75.73040	0.54697
C++/OpenMP	24	26731	159.764	97.93872	0.61302
CUDA/Thrust	V100	6990	113.74	374.53505	3.29291

Table 13: Sinkhorn WMD Performance Measurements.

implementation	threads	μsec	Watts	GOPS	GOPS watt
C++/OpenMP	1	162806	95.747	3.19399	0.03336
C++/OpenMP	2	94137	103.882	5.52386	0.05317
C++/OpenMP	4	49299	114.385	10.54788	0.09221
C++/OpenMP	8	25149	132.376	20.67677	0.15620
C++/OpenMP	16	13584	155.851	38.28033	0.24562
C++/OpenMP	24	10077	184.344	51.60266	0.27993

Table 14: IPNSW Performance Measurements.

- Baseline Python
- Optimized Python
- Optimized C++

The results of the evaluations are presented in Table 14.

6.2.2 Kernel evaluation / performance

Kernels are smaller pieces of functionality that were selected specifically to be implemented for FPGA evaluation as well as CPU and GPU. UDRI performed FPGA implementations and documented the performance. ARLIS identified optimized algorithms and to capture specific performance metrics (runtime, operation count, energy consumption) for the 6 selected kernels. Intel's MKL for the CPU and NVIDIA's CUDA for the GPU were selected as libraries that provide highly optimized, freely redistributable implementations of many workloads. Specifically, these libraries provide the fast fourier transform (FFT), sparse matrix multiplication, and dense matrix multiplication algorithms. Both libraries also com- bine the advantages of having good documentation and being widely used, so there is relatively good technical support.

C was selected as our primary programming language, as Intel MKL and NVIDIA CUDA offer support for C and C offers superior efficiency to higher-level programming languages. For application profilers, we selected Intel Advisor for collecting CPU operation count, NVIDIA NSight Compute for collecting GPU operation count, and Arm MAP for collecting CPU and GPU energy consumption. These were selected for providing the desired metrics and for being compatible with the CPU and GPU implementations and hardware we used. For auxiliary code, such as data conversion and aggregation scripts, we wrote in high-level languages (Python and Bash scripts) to save time and keep the code concise. The hardware used to measure kernel performance was:

- Azure NC6s_v3 instance.
- GPU: NVIDIA Tesla V100 GPU
- CPU: Intel Xeon E5-2690 v4 (Broadwell), 6 vCPUs

Lol, Same idea as me.

As the kernels were selected specifically to be implemented on FPGA, UDRI's FPGA results and ASIC estimates are included in Table 15.

7 Discussion and Implications

The dominance of a common underlying computing paradigm is fast giving way to a large ecosystem of hardware technologies that, when cleverly arranged, can create new, disruptive possibilities. We have seen this happen before. In 2005-2006, between the first DARPA Grand Challenge and the second a little over a year later, it was the integration of new sensor hardware (specifically LIDARs) that created the leaps in capability that empowered the winning self-driving vehicles. The adoption of GPUs in the late 2000s enabled the innovations

Kernel	Device	FMAX(FPGA) MHz	Core Power (FPGA) Watts	FPGA Efficiency GF LOPsW	ASIC Estimates GF LOPsW
FFT	Stratix10	750	10.2	347	700
2D Convolution	Stratix 10 MX	750	9.6	540	1100
Dense MM	Stratix 10	600	25	610	1200
Sparse MM	Stratix 10	182	15	2.5	5
Dijkstra	Kintex 7	100	12	0.430	0.860
Auction	Kintex 7	100	9.5	0.290	0.580

Table 15: SDH FPGA and ASIC proxy Results.

in deep learning, etc. What is different today is the diversity of emerging innovations and the bespoke nature of the solutions that they enable. Rather than a broad capability provided by a GPU, a technology may become readily available that is highly suitable for a certain type of matrix computation (i.e., the sparse-dense case discussed earlier). For specific applications that have a sufficient set of these cases, the new technology can offer considerable advantages but in the general case it may not outperform a traditional approach. The challenge for any of us who build, test, and evaluate intelligent systems will be to understand the design tradeoffs across this landscape and how to best optimize their creation.

Given these experiences, the we offer several observations along with possible prescriptions for future research.

What are the metrics? The AI community needs a broader and more expansive notion of what to measure and how to measure it. For those of us grounded in theoretical computer science, one might think in terms of "Big O" and the number of states examined, nodes expanded, etc.—the traditional measures of performance[**?**]. We also talk about measures of effectiveness[], which would include assessments of completeness, quantification of solution accuracy, and the like. We use these to talk about run time or storage space required, or solution quality—and the trade offs across these measures such as one might find in real-time systems, contract algorithms, and anytime algorithms. It has become clear that these approaches are not fully adequate to understand this new ecosystem and its trade-offs.

For example, only recently have those in the multi-agent systems community begun to characterize communications (represented as messages among agents) as a metric for consideration when building distributed algorithms. In some cases such work draws on concepts from the networking community, and we believe there are analogies to be drawn here with understanding the metrics we need to develop around emerging hardware. Taking insights from the high-performance computing community, hardware literature and related areas one must begin to understand measures such as GOPS/PER/WATT, energy cost per operation, computation accuracy associated with the method (if the method is analog for example).

REWRITE THIS WITH THE EXAMPLES WE STUDIED!

The result of a renewed focus on the science and application of metrics would be to create better approaches to understanding system performance and effectiveness—and improve our overall engineering processes for building such systems that can leverage emereging hardware.

Accessing future hardware. The emerging diversity of hardware paradigms and capabilities has created a new set of options that systems designers must now include and account for. Hence, systems designers and software engineers cannot simply assume that the compiler and the run-time optimizer is going to be able to lay out the execution of a system in the manner that best takes advantage of the available hardware for the given problem or application. The compiler does not know, for example, that you are expecting a dataset that could require lots of processing of convolutions or that the energy budget for your system is constrained by the draw on a solar panel and a battery. There are system design parameters that might include options across the hardware ecosystem that, while available to many, are not the domains of expertise by those that could most benefit.

The community would benefit by having more methods and tools for system design that enable non-experts to assess the potential utility and tradeoffs among different hardware-software designs. For example, such co-

design approaches for intelligent systems might take example from the networking community in the following manner: the OSI reference model[] provides a framework for understanding the different elements of the communications stack and networking hardware and software solutions can define their capabilites with respect to this stack. A reference model around emerging AI hardware, while it would take a different form than a strict stack, might provide guidance around the mathematical operations, energy requirements, or data landscapes in which intelligent systems reside and enable developers to guide their design decisions (or eventually automate them).

The limits have moved. As with the introduction of GPUs into widespread use, new hardware means that the concept of what is fundamentally difficult is changing as certain problems can be increasingly brute-forced. For example, hardware architectures such as Intel's PIUMA enable rapid acceleration of graph traversal algorithms such as breadth first search, A*, and related algorithms—changing the calculus about the cost in time and energy for these techniques and expanding their potential applications to new areas and improving their performance in others. This creates the potential to create new capabilities by simply moving the limits on what scale and nature of problem were previously intractable. Historically, this is similar to the innovations of IBM's Deep Blue and Watson (both customized hardware, one for search machine for chess and one for machine question-answering); the Connection Machine (enabling parallel search); and other hardware-driven innovations. The process of GPU adoption also fits this pattern—providing everyone with custom processor for large scale linear algebra. The difference today is that these innovations are now available to many more people across a wider variety of computational workloads. The research question emerges to determine how to design and implement systems that use these processors to brute force with optimal impacts on performance and effectiveness.

Fundamental limits will remain. Certain problems will remain evergreen. Matters of scale, issues of size-weight-power, problems created by the velocity of data or time constraints on processing will persist as hardware elements improve. What appears likely to happen is that emerging hardware will enable application developers to attack these fundamental limits in various niche applications. Much like the matrix processing kernels for GPUs, there will be a growing library of hardware accelerators for many other mathematical or combinatorial operations that are central to AI algorithms. Improved theoretical and empirical techniques for understanding these limits will aid in understanding how to best apply new hardware to specific mission problems.

Programming Abstractions and Optimizations for AI In executing this analysis and conducting these case studies, the team invariably had to break apart the application workflows into their foundational functions to determine if and where the new hardware elements might be used. This process appears to bear resemblance to the changes that occurred in the field of Computer Graphics with the advent of OpenGL and customized Graphics Processing Units. OpenGL offered abstraction for functions that previously would have been implemented directly on the CPU. Even more historically, the appearance of the personal computer in the 1980 originally did not include chips for direct processing of floating point calculations—requiring the use of approximation code or exact arithmetic libraries in order to perform operations on real numbers. The introduction of floating point co-processors, and their eventual integration onto the main CPU, changed how those data objects were used and manipulated.

Across the broad research communities looking at AI and ML, this transformation is being seen today in the widespread use of use of common libraries and software frameworks: AI Gyms, Python, and the AI cloud frameworks offered by commercial vendors. These efforts all offer higher abstractions for programming AI applications. These abstractions increasingly will connect with the emerging hardware elements to provide direct access to bespoke computational capabilities without developers having to become hardware experts.

AI "Systems Engineering". For much of the history of AI and ML, the field has focused at the level of the development of algorithmic techniques and mathematical tools. These activities typically exist above the specifics of the hardware and are safe assuming a Von Neumann architecture and the widely shared goals of understanding the fundamental limits of space and time complexity. Time complexity many areas of AI is viewed as an abstract measurement of discrete steps or operations or "nodes" or something similar. With the explosion of

hardware paradigms, the creation of AI "systems" is becoming an increasingly engineering exercise. In real-time control or robotics applications, the all system elements are interdependent and need to be viewed collectively. For examples, energy consumption of the processor can affect networking performance or the actuation of motors as they share power sources. Going forward, systems thinking for AI/ML design and deployment of applications is beginning to require better tools and methods for understanding tradeoffs in hardware design and the costs of the computational fabric used to execute the whole system application.

Implications for AI Education. At present, any developer with Internet has access to CPU, GPU and cloud-based resources. More advanced organizations have access to their own customized clusters and hardware capabilities. While this diversity is growing, the typical undergraduate computer science major developing AI/ML applications has little experience with hardware. As the diversity of the discipline has grown, the core computer science curriculum has largely moved away from teaching concepts of hardware and most programs do not have a software or systems engineering component. The interdisciplinary nature of this new tradespace of hardware-software-application codesign invites a new type of interdisciplinary thinking. To some degree these choices may be able to be automated, but beyond the easily automated developers of AI applications will need new tools to help them understand tradeoffs and make choices.

8 Conclusions

The field of robotics has always been driven by hardware advances and many successes can be viewed as triumphs of hardware-software co-design. Invoking the historic custom computing created during World War II, we are now in a vast landscape of computationally extreme problems that necessitate inventive new ways to compute. While our programming abstractions may eventually subsume the hardware distinctions, we are all becoming codesigners and systems engineers.

Bibliography

[1] Non-metric similarity graphs for maximum inner product search.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486, 2006.

[3] L. M. Ang and K. P. Seng. Gpu-based embedded intelligence architectures and applications. *Electronics*, 10(8), 2021.

[4] B. E. Carpenter and R. W. Doran. A. M. Turing's ACE Report of 1946 and Other Papers. 1986.

[5] F. Chung and O. Simpson. Computing heat kernel pagerank and a local clustering algorithm. *European Journal of Combinatorics*, 68:96–119, 2018. Combinatorial Algorithms, Dedicated to the Memory of Mirka Miller.

[6] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2292–2300, Red Hook, NY, USA, 2013. Curran Associates Inc.

[7] M. F. Deering. Hardware and Software Architectures for Efficient AI. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence*, AAAI'84, page 73–78. AAAI Press, 1984.

[8] S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski. Massively parallel architectures for al: Netl, thistle, and boltzmann machines. In *proceedings Association for the Advancement of Artificial Intelligence (AAAI) conference*, pages 109–113, 1983.

[9] R. J. Fateman. Is a LISP Machine Different from a Fortran Machine? *SIGSAM Bull.*, 12(3):8–11, aug 1978.

[10] K. Fukushima. Cognitron: a self-organizing multilayered neural network. *Biological Cybernetics*, 20(3-4):121–136, 1975.

[11] H. Gelernter, J. R. Hansen, and D. W. Loveland. Empirical explorations of the geometry theorem machine. In *Papers Presented at the May 3-5, 1960, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '60 (Western), page 143–149, New York, NY, USA, 1960. Association for Computing Machinery.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016.

[14] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[15] L. Hiller and L. Issacson. *Experimental music : composition with an electronic computer*. McGraw-Hill, 1959.

[16] G. E. Hinton and R. Zemel. Autoencoders, minimum description length and helmholtz free energy. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993.

[17] K. Kloster and D. F. Gleich. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 1386–1395, New York, NY, USA, 2014. Association for Computing Machinery.

[18] O. Kuchaiev and B. Ginsburg. Training deep autoencoders for collaborative filtering. *arXiv preprint arXiv:1708.01715*, 2017.

[19] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 957–966. JMLR.org, 2015.

[20] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(6):436–444, 2015.

[21] LeCun, Yann. Generalization and network design strategies., 1989.

[22] G. Li, L. Liu, and X. Feng. Accelerating gpu computing at runtime with binary optimization. In *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 276–277, 2019.

[23] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Commun. ACM*, 3(4):184–195, apr 1960.

[24] O. Mencer, D. Allison, E. Blatt, M. Cummings, M. J. Flynn, J. Harris, C. Hewitt, Q. Jacobson, M. Lavasani, M. Moazami, H. Murray, M. Nikravesh, A. Nowatzyk, M. Shand, and S. Shirazi. The history, status, and future of fpgas. *Commun. ACM*, 63(10):36–39, sep 2020.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[26] A. Newell and H. Simon. The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956.

[27] A. Newell and H. A. Simon. Computer simulation of human thinking. *Science*, 134(3495):2011–2017, 1961.

[28] N. Nilsson. The quest for artificial intelligence: A history of ideas and achievements. *Isis*, 102:588–589, 2011.

[29] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, page 5–14, New York, NY, USA, 2017. Association for Computing Machinery.

[30] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[31] Owens, John D. and Luebke, David and Govindaraju, Naga and Harris, Mark and Kruger, Jens and Lefohn, Aaron E. and Purcell, Timothy J. A Survey of General-Purpose Computation on Graphics Hardware. In *Proceedings of Eurographics 2005*, page 21–51, 2005.

[32] S. Pal, S. Feng, D.-h. Park, S. Kim, A. Amarnath, C.-S. Yang, X. He, J. Beaumont, K. May, Y. Xiong, K. Kaszyk, J. M. Morton, J. Sun, M. O'Boyle, M. Cole, C. Chakrabarti, D. Blaauw, H.-S. Kim, T. Mudge, and R. Dreslinski. Transmuter: Bridging the efficiency gap using memory and dataflow reconfiguration. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, page 175–190, New York, NY, USA, 2020. Association for Computing Machinery.

[33] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[34] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 873–880, New York, NY, USA, 2009. Association for Computing Machinery.

[35] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[36] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books, 1962.

[37] R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, 1997.

[38] J. Shun, F. Roosta-Khorasani, K. Fountoulakis, and M. W. Mahoney. Parallel local graph clustering. *Proc. VLDB Endow.*, 9(12):1041–1052, aug 2016.

[39] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 81–90, New York, NY, USA, 2004. Association for Computing Machinery.

[40] D. A. Spielman and S.-H. Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1):1–26, 2013.

[41] D. Spinelli. *OCCAM: A computer model for a content addressable memory in the central nervous system*. Academic Press New York, 1970.

[42] 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.

[43] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[44] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao. How to build a benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, page 333–336, New York, NY, USA, 2015. Association for Computing Machinery.

[45] J. von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

[46] Waltz, David L. Massively Parallel AI. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI'90, page 1117–1122. AAAI Press, 1990.