SIP (2017), vol. 6, e9, page 1 of 10 © The Authors, 2017.

doi:10.1017/ATSIP.2017.9

INDUSTRIAL TECHNOLOGY ADVANCES

AI, native supercomputing and the revival of Moore's Law

CHIEN-PING LU

Artificial Intelligence (AI) was the inspiration that shaped computing as we know it today. In this article, I explore why and how AI would continue to inspire computing and reinvent it when Moore's Law is running out of steam. At the dawn of computing, Alan Turing proposed that instead of comprising many different specific machines, the computing machinery for AI should be a Universal Digital Computer, modeled after human computers, which carry out calculations with pencil on paper. Based on the belief that a digital computer would be significantly faster, more diligent and patient than a human, he anticipated that AI would be advanced as software. In modern terminology, a universal computer would be designed to understand a language known as an Instruction Set Architecture (ISA), and software would be translated into the ISA. Since then, universal computers have become exponentially faster and more energy efficient through Moore's Law, while software has grown more sophisticated. Even though software has not yet made a machine think, it has been changing how we live fundamentally. The computing revolution started when the software was decoupled from the computing machinery. Since the slowdown of Moore's Law in 2005, the universal computer is no longer improving exponentially in terms of speed and energy efficiency. It has to carry ISA legacy, and cannot be aggressively modified to save energy. Turing's proposition of AI as software is challenged, and the temptation of making many domain-specific AI machines emerges. Thanks to Deep Learning, software can stay decoupled from the computing machinery in the language of linear algebra, which it has in common with supercomputing. A new universal computer for AI understands such language natively to then become a Native Supercomputer. AI has been and will still be the inspiration for computing. The quest to make machines think continues amid the slowdown of Moore's Law. AI might not only maximize the remaining benefits of Moore's Law, but also revive Moore's Law beyond current technology.

Keywords: Moore's Law, AI, Deep Learning, Supercomputing, Alan Turing

Received 1 December 2016; Revised 21 July 2017

I. AI AND THE UNIVERSAL COMPUTER

What kind of computing machinery do we need to advance Artificial Intelligence (AI) to human level? At the dawn of computing, one of the founding fathers, Alan Turing, believed that AI could be approached as *software* running on a universal computer. This was a revolutionary idea given that during his time, the term "computer" was generally referred to as a human hired to do calculations with pencil on paper. Turing referred to a machine as a "digital computer" to distinguish it from the human one.

In the context of AI, Alan Turing is remembered for his *Imitation Game*, or later referred to as *Turing Test*, in which a machine strives to exhibit intelligence to make itself indistinguishable from a human in the eyes of an interrogator. In his landmark paper, "Computing Machinery and Intelligence" [1], he tried to address the ultimate AI question, "Can machines think?" He reframed the question more precisely and unambiguously by asking how well a machine does in the imitation game. Turing hypothesized that human intelligence is "computable," which has a precise mathematical meaning famously established by himself [2], as a bag of discrete state machines, and reframed the ultimate AI question as

> Are there discrete machines that would do well **(in the imitation game)**? [1]

But what exactly are the discrete state machines to win the imitation game? Apparently, he did not know during his time; but witnessing the extreme difficulty of building a non-human, electronic computer himself [3], he envisioned only one machine, the Universal Digital Computer that could mimic any discrete state machine. Each discrete state machine can be encoded as numbers to be processed by a universal computer. The numbers that encode a discrete state machine become software, and the computing machinery became the "stored program computer"

Novumind Inc, Hardware Engineering, Santa Clara, California, USA

Corresponding author:
C.-P. Lu
Email: cpl@novumind.com

envisioned by John von Neumann in his incomplete report [4]. Thus, Turing concluded:

> Considerations of speed apart, it is unnecessary to design various new machines to do various computing processes. They can all be done with one digital computer, suitably programmed for each case [1].

Thereafter, the history of computing has been mainly the race to build faster universal computers to answer the following challenge:

> Are there imaginable digital computers that would do well **(in the imitation game)**? [1]

AI researchers and thinkers have been advancing AI without worrying about the underlying computing machinery. People might argue that this applies only to traditional rule-based AI. However, even connectionists have to translate their connectionist systems into algorithms in software to prove and demonstrate their ideas. We have been seeing advances and innovations in Deep Learning completely decoupled from the underlying computing machinery. Today, we use terms like "machines", "networks", "neurons", and "synapses", without a second thought about the fact that those entities do not have to exist physically. People ponder about a grand unified theory of Deep Learning using ideas like "emergent behaviors", "intuitions", "non-linear dynamics", believing that those concepts could be adequately represented or approximated by software. According to Turing, any fixed-function Deep Learning accelerator can be simulated in software, and there is always a fallback path to software in applications using such an accelerator. AI has been and will be advanced as software.

II. THE PERFECT MARRIAGE BETWEEN THE UNIVERSAL COMPUTER AND MOORE'S LAW

Turing's Universal Computer inspired von Neumann to come up with a powerful computing paradigm, in which complex functions were expressed in a simple yet complete language, the Instruction Set Architecture (ISA), that computing machinery could understand and execute. It brought us computers, as well as the software industry. The prevailing computing machinery in the era of von Neumann paradigm is the microprocessor, now a synonym of the Central Processing Unit (CPU), designed to run instructions in stored programs sequentially. The CPU, the Graphics Processing Unit (GPU), and the various kinds of Digital Signal Processor and programmable alternatives are all modern incarnations of such a Universal Digital Computer.

But how would such a computer, emulating non-intelligent and non-thinking behaviors of a human, demonstrate human-level intelligence? Turing's answer was this:

> Provided it could be carried out sufficiently quickly the digital computer could mimic the behaviour of any discrete state machine [1].

As far as AI is concerned, Turing's idea was that AI can fundamentally be approached through software running on a Universal Digital Computer. It would be the responsibility of the architects of the computing machinery to make it sufficiently fast. But how would we make it faster and at what rate?

Moore's Law, coined in Gordon Moore's seminal paper [5], has been followed by the semiconductor industry as a consensus and commitment to double the number of transistors per area every 2 years. Based on the technology scaling rule called Dennard Scaling, transistors have not only become smaller, but also faster and more energy efficient such that a chip now offers at least twice the performance at roughly the same dollar and power budgets. The performance growth mainly came from Moore's Law driving the clock speed exponentially faster. From 1982 to 2005, typical CPU clock speed grew by 500 times from 6 to 3 GHz. Computing machinery vendors strived to build more capable CPUs, through faster clock speeds and capacity to do more than one thing at a time while maintaining the sequential semantics of a universal computer. Software vendors endeavored to explore new application scenarios and solve the problems algorithmically. The decoupling of software from the computing machinery and the scaling power of Moore's Law triggered the computing revolution that has made today's smart phones more powerful than supercomputers two decades ago.

However, faster computers have not helped AI pass the Turing Test yet. AI started out as a discipline to model intelligence behaviors with algorithmic programs following the von Neumann paradigm. It had been struggling to solve real-world problems and waiting for even faster computers. Unfortunately, the exponential performance growth of a universal computer has ground to a halt.

III. THE SLOWDOWN OF MOORE'S LAW

The turning point happened in 2005, when the transistors, while continuing to double in numbers, were neither faster nor more energy efficient at the same rates as before due to the breakdown of Dennard Scaling. Intel wasted little time to bury the race for faster clock speed, and introduced multi-core to keep up performance by running multiple "cores" in parallel. A universal computer became a CPU "core". Multi-core has been a synonym of parallel computing in the CPU community. It was expected that there would be a smooth transition from von Neumann paradigm to its parallel heir, and the race for faster clock speed would be replaced with one for higher core count starting from dual and quad cores, to eventually a sea of cores. Around

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

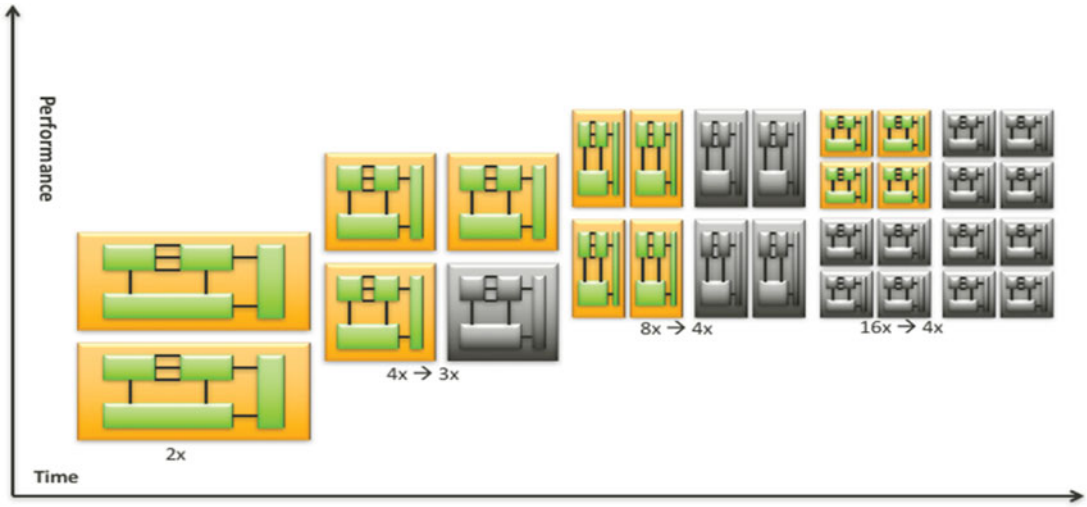

Fig. 1. Dark Silicon phenomenon: diminishing returns with more cores.

the same time, programmers were asked to take on the challenge of writing and managing a sea of programs, or "threads" [6].

Such a race to double core count has not happened. Intel and the CPU industry have been struggling to add cores aggressively due to the issue of lagging improvements in transistor energy efficiency, manifested as the Dark Silicon phenomenon. It implies that while being able to accommodate four times more cores on a die through two generations of transistor shrinking, we could power up only half of the cores. If this does not look serious enough, only one-quarter of the cores can be powered up at the third generation of transistor shrinking. Unless we reduce the core aggressively to compensate for the lagging improvement in energy efficiency, there might be no incentive to go with the fourth generation of transistor shrinking as there will be negligible performance improvement (see Fig. 1). To make the situation even worse, the gap between the speed of memory and that of logic has been widening exponentially.

Such a limit applies to any computing machinery with an ISA legacy to carry, including the GPU. Although the GPU does not need to support ISA compatibility to every bit, it still needs to support higher level standards such as OpenGL and DirectX shading languages and OpenCL, and intermediate-level standards such as SPIR-V. NVIDIA needs to maintain the legacy in their propriety CUDA. For software, managing the threads explicitly for a sea of cores has turned out to be untenable unless we restrict the communications among the threads to some patterns. *Such massive and unwieldy parallelism is not for the computing machinery and software to tackle.*

Some prominent research on Dark Silicon, such as "Dark Silicon and the End of Multicore Scaling" by Hadi Esmaeilzadeh [7], confused the physical limitation in semiconductor with that from Amdahl's Law, and prematurely declared the death of parallelism along with the slowdown of Moore's Law. There is abundant parallelism in AI with Deep Learning as we will see later.

IV. AI AND MOORE'S LAW

Turing was not specific about the performance and energy efficiency of a universal computer. He assumed that computers would always be sufficiently fast, and would not be a gating factor for the quest for human-level AI; but if passing the Turing Test is the ultimate criteria for machine intelligence, he would have suggested that the computers must achieve a certain level of performance and efficiency to exhibit intelligence; otherwise, the interrogator would be suspicious if it takes too long for a computer to respond to questions or consumes too many resources in the effort.

Turing envisioned his digital computer as one that models the slow thinking process of a human doing calculations with a pencil on a piece of paper. The Universal Digital Computer was named to imply that it was designed to model after a human "computer". According to Turing:

> The human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail [1].

In other words, such a Universal Digital Computer does not think, but follows the instructions provided by software. It is the software that makes it think. Following fixed rules strictly requires intensive concentration and is an energy-consuming and slow process for a human brain. Try to multiply 123 by 456 in your head while you are running. It will slow you down. Interestingly, what is energy consuming for human is also for computers. To accomplish a task by executing one instruction at a time takes relatively more energy than doing it natively without the intermediate ISA. Approaching AI as software in the von Neumann paradigm is like mimicking fast and effortless human mental functions, such as intuition, with a machine that is based on the slow mental process of a human.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

Turing did not foresee that a universal computer would run out of steam. If we are to stay with the von Neumann computing paradigm, we need to put an army of universal computers in a machine to continue the quest. These universal computers would have to communicate data and coordinate tasks among them. However, the slowdown of Moore's Law and the legacy of the von Neumann paradigm suggest that we will not be able to supply sufficient energy to keep such an army growing in size. There needs to be a paradigm shift for AI and computing.

Turing did foresee that it would be difficult for human programmers to write all the software to achieve human-level AI. He suggested that we build a learning machine. He said:

> Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain [1].

This idea points to Deep Learning. Although Turing did not predict the emergence of Deep Learning, he was aware of the approach with Neural Networks:

> It is generally thought that there was always an antagonism between programming and the "connectionist" approach of neural networks. But Turing never expressed such a dichotomy, writing that both approaches should be tried [8].

If Turing was alive today and witnessed the emergence of Deep Learning, he would have revised his proposition on the computing machinery for AI. Since we only need to simulate the child's mind, and educate it, the computing machinery can model a child's ability to learn and an adult's capability to leverage the learned knowledge. Such a machine would be different from the universal computer he envisioned.

V. DEEP LEARNING AND THE NEW AI MACHINE

Deep Learning has been transforming and consolidating AI since it came to the center stage of computing in 2012. With Deep Learning, the intelligence is not coded directly by programmers but acquired indirectly by mining training data sets, and then encoded in the various forms of Neural Networks. The acquisition and manifestation of the intelligence can be formulated as computations dominated by a compact set of linear algebra primitives analogous to those defined in BLAS (Basic Linear Algebra Subprograms) [9], the fundamental application programming interface used in supercomputing and high-performance computing (HPC). AI with Deep Learning and supercomputing effectively speak the same language with dialectical variances in numerical precisions, and minor differences in domain-specific requirements.

As mentioned earlier, the massive and unwieldy parallelism under the von Neumann paradigm is not for the computing machinery and software to tackle. On the other hand, the patterns of parallelism in supercomputing can be summarized as collective communications (see Fig. 2) as described in Franck Cappello's "Communication Determinism in Parallel HPC Applications" [10]. Collective communication has been proven to be scalable and manageable in large-scale distributed supercomputing systems. The question is how to leverage the collective patterns on a chip.

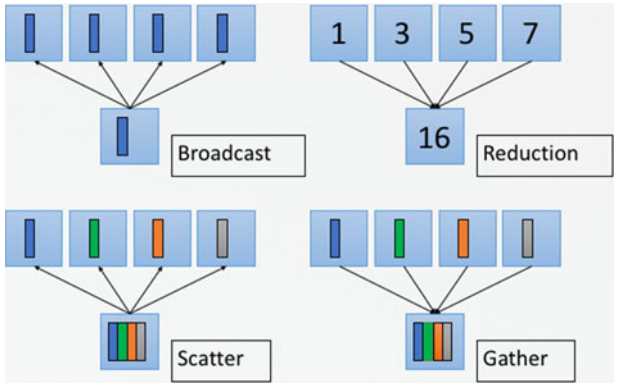

Fig. 2. Four basic collective communication operations.

Through Deep Learning, AI can potentially be liberated from the von Neumann architecture and talk to a native linear algebra machine with massive hardware parallelism, if there is one.

A) Why linear algebra?

The fundamental primitives in Deep Learning are tensors, high-dimensional data arrays used to represent layers of Deep Neural Networks. A Deep Learning task can be described as a tensor computation graph (Fig. 3):

A tensor computation graph is effectively a piece of AI software. Tensors can be unfolded into two-dimensional matrices, and matrix computations are handled through matrix kernels (see Fig. 4). Matrix kernels refer to CPU or GPU programs implementing different types of matrix computations comprising many MAC (multiply accumulate) operations. Such a matrix-centric approach is described by Sharan Chetlur [11]. The MAC operations for matrix multiplication are the most time-consuming part of

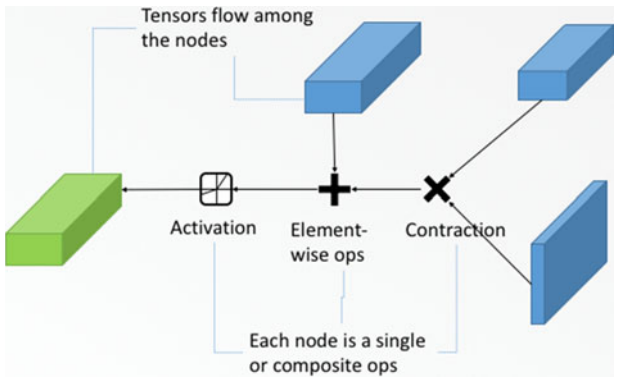

Fig. 3. A tensor computation graph.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

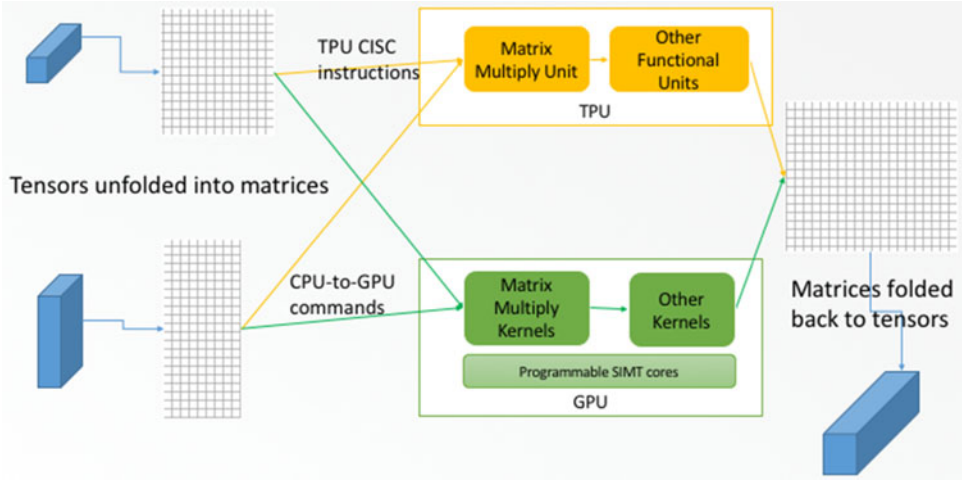

Fig. 4. Matrix-centric platforms on the GPU and the Tensor Processing. Unit (TPU).

Deep Learning. One might ask if computations in Deep Learning are predominantly MACs in matrix computations, why don't we simplify a core all the way to a MAC unit that does nothing but a MAC operation? In fact, why does a MAC unit need to keep the legacy of being a core at all?

B) The Tensor Processing Unit and systolic arrays

In the highly anticipated paper, "In-Datacenter Performance Analysis of a Tensor Processing Unit" [12], Google disclosed the technical details and performance metrics of the Tensor Processing Unit (TPU). The TPU was built around a matrix multiply unit based on systolic arrays. What is eye-catching is the choice by the TPU design team to use a systolic array. A systolic array is a specific spatial dataflow machine. A Processing Element (PE) in a systolic array works in lock step with its neighbors. Each PE in a systolic array is basically a MAC unit with some glue logic to store and forward data. In comparison, a computing unit equivalent to a PE in a mesh-connected parallel processor is a full-featured processor core with its own frontend and necessary peripherals, whereas a PE equivalent in a GPU is a simplified processor core sharing a common frontend and peripherals with other cores in the same compute cluster. Among the three solutions, the density of MAC units is the highest in a systolic array. These differences are shown in Fig. 5:

A systolic array claims several advantages: simple and regular design, concurrency and communication, and balancing computation with I/O. However, until now, there has been no commercially successful processor based on a systolic array. The TPU is the first, and it is impressive, arguably the largest systolic array implemented or even conceived. Their design is reminiscent of an idea introduced by H.T.

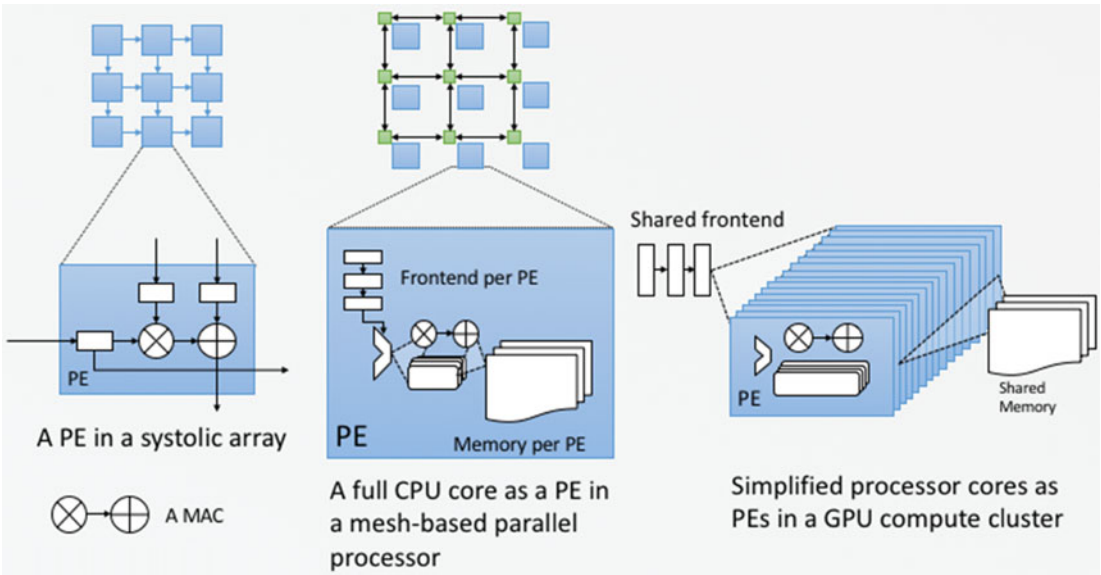

Fig. 5. PEs in a systolic array, mesh-connected parallel processor and a GPU.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

Kung [13]. However, due to the curse of the square shape, it suffers from scalability issues as elaborated in the LinkedIn article, "Should We All Embrace Systolic Arrays" [14].

VI. SPATIAL DATAFLOW ARCHITECTURE

Like a systolic array, the building block of a generic spatial dataflow machine is often referred to as the PE, which is typically a MAC unit with some glue logic. Mesh topology is a strikingly popular way to organize PEs, for example, Google's TPU [12], the DianNao family [15], MIT's Eyeriss [16] (see Fig. 6).

It seems logical to use a mesh topology to organize the PEs on a two-dimensional chip when there are lots of PEs and regularity is desirable. Such an arrangement leads to the following two *mesh-centric assumptions*:

(1) The distance for a piece of data to travel across the mesh in one clock period is fixed as that between two neighboring PEs, even though it could be much further;
(2) A PE depends on the upstream neighboring PEs to compute even though such a dependency mainly comes more from the spatial order, rather than from true data dependency.

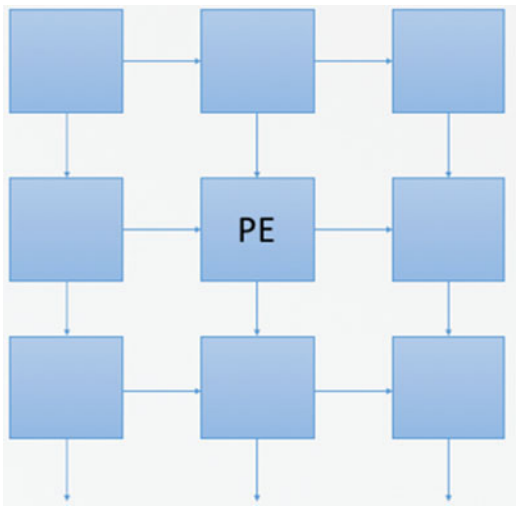

Fig. 6. A PE and its neighbors in a mesh.

The first assumption is a legacy inherited from distributed parallel processors comprising many compute nodes. Each compute node has to communicate among themselves through intermediate nodes. It is analogous to the situation when a high-speed train stops at every single station on the way to the destination, as shown in Fig. 7. Within one clock period, a piece of data could travel over a distance equal to hundreds of the width of a MAC unit without having to hop over every single MAC unit in between. Restricting dataflows to PE hopping in a mesh topology causes an increase in latency by several orders of magnitude.

The second assumption is another legacy inherited from distributed parallel processors. Each compute node not only handles computations but also plays a part in the distributed storage of the data. The nodes need to exchange data among them to make forward progress. For an on-chip processing mesh, however, the data come from the side interfacing with the memory. The dataflow through the mesh and the results are collected on the other side as shown in Fig. 8. Due to the local topology, an internal PE has to get the data through the PEs sitting between it and the memory. Likewise, it has to contribute its partial result through the intermediate PEs

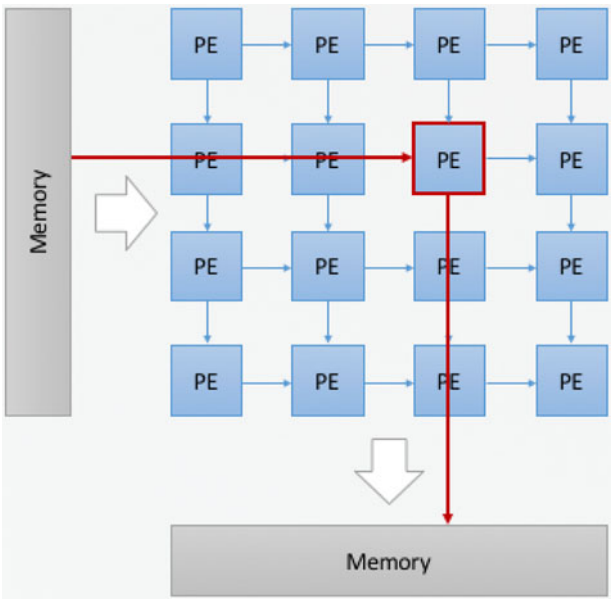

Fig. 8. Mesh-centric assumption 2.

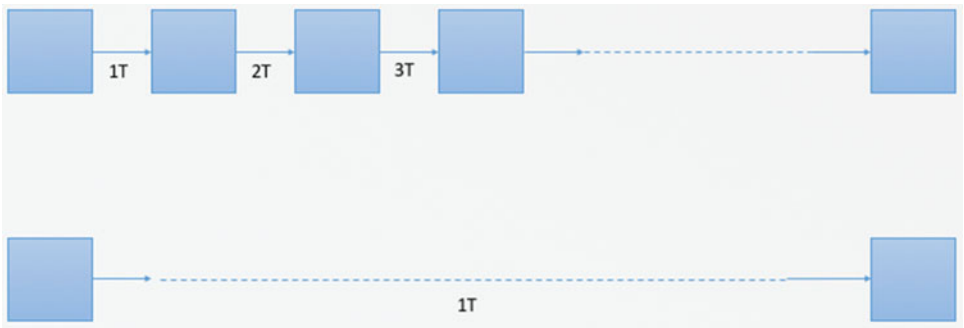

Fig. 7. Mesh-centric assumption 1.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

before reaching the memory. *The resulting dataflows are due to the spatial order of the PE in the mesh, not as a result of true data dependency*.

Given the two mesh-centric assumptions, no matter how many PEs and how much bandwidth you have, the performance to solve a problem on a d-dimensional mesh is limited by the dimensionality d of the mesh, not the number of the PEs, nor the IO bandwidth. Suppose a problem requires I inputs, K outputs, and T computations, then the asymptotic running time to solve the problem on a d-dimensional mesh is given by Fisher's bound [17]:

$$t = \Omega\left(\max\left(\sqrt[d]{I}, \sqrt[d]{K}, \sqrt[d+1]{T}\right)\right).$$

Fisher's bound implies there are upper bounds on the number of PEs and bandwidth beyond which no further running time improvement is achievable.

Applying Fisher's bound to the inner product, the running time to do an inner product is $\Omega(n)$ on a one-dimensional mesh. If you can afford to have a two-dimensional mesh, the running time is $\Omega(\sqrt{n})$ Can we do better? Instead of using one- or two-dimensional mesh, we can feed the input to n PEs and add the products in pairs recursively. A $\Omega(\log(n))$ running time can be achieved. However, it is not possible to achieve such a performance on an either one- or two-dimensional mesh unless we organize the PEs in the way shown in Fig. 9.

The reasons for such a *superoptimal* result compared with the theoretical limits on a mesh is that there is no PE hopping, and it uses links of different lengths assuming that it takes the same time for a piece of data to travel over links with different lengths. If the distance is too long for a piece of data to travel in one clock period, we can add flops in the middle. It should be an implementation issue, not an architectural one.

VII. MATRIX MULTIPLICATION ACCORDING TO SUPERCOMPUTING

Let us look at the most time-consuming part of Deep Learning: matrix multiplication, which has always been at the heart of supercomputing. State-of-the-art parallel matrix multiplication performance on modern supercomputers is achieved with the following two major advancements:

(1) Scalable matrix multiplication algorithms,
(2) Efficient collective communications with logarithmic overhead.

A) Scalable matrix multiplication algorithms

See Fig. 10 for the demonstration of matrix multiplication in outer products. The computations are two-dimensional, but both the data and the communications among them are one-dimensional.

The width of a block column and a block row can be a constant and is independent of the number of nodes. On a systolic array, the computations are also broken down into outer products. However, the width of the block column/row must match the side length of the systolic array to achieve optimal performance. Otherwise, the array is poorly occupied for problems with low inner dimension.

Outer product-based matrix multiplication algorithms, such as Scalable Universal Matrix Multiplication Algorithm (SUMMA) [18], have been proven to be very scalable both in theory and in practice in distributed systems.

B) Efficient collective communications with logarithmic overhead

The communication patterns in SUMMA or similar algorithms are based on collective communications defined for

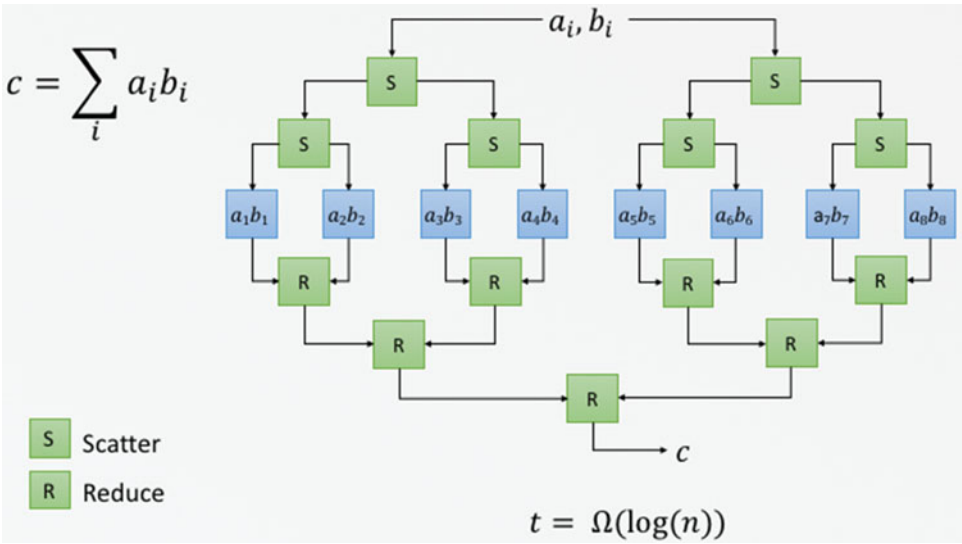

Fig. 9. A faster inner products than Fisher's bound.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

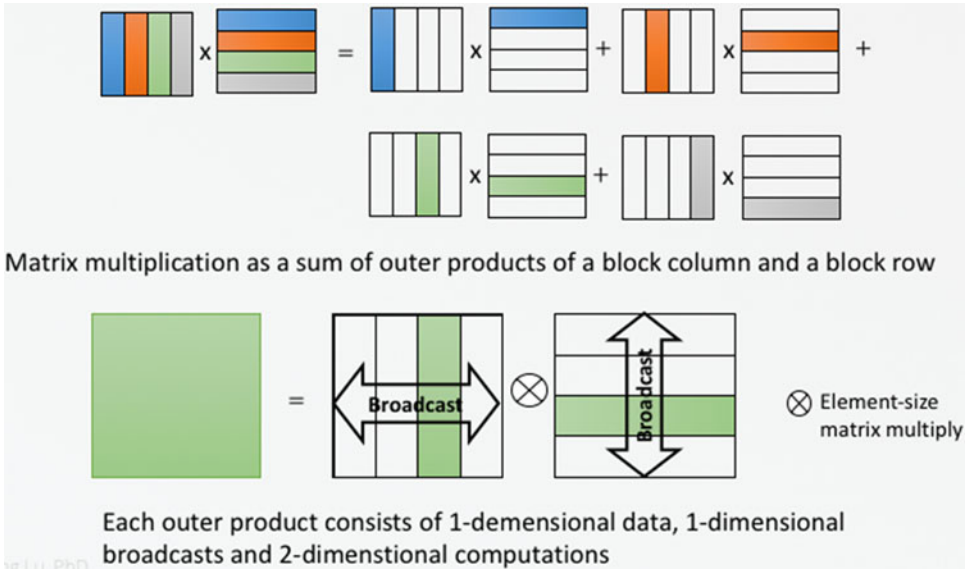

Fig. 10. Matrix multiplication with outer products.

parallel computing on distributed systems. Advances in collective communication for HPC with recursive algorithms [19] reduce the communication overheads to be proportional to a logarithmic of the number of nodes and have been instrumental in the continuing performance growth in supercomputing.

VIII. NATIVE SUPERCOMPUTING

It is interesting to compare how matrix multiplication is achieved with a systolic array and a supercomputer, even though they are at completely different scales: one is on-chip and each node is a PE; the other is at the scale of a data center and each node is a compute cluster (Fig. 11).

Broadcasts are implemented as forwarding data rightward, and reductions (a synonym of "accumulate" in the terminology of collective communications) are implemented as passing partial sums downward in a systolic array and accumulate along the way.

In comparison with an algorithm like SUMMA, broadcasts on a supercomputer happen in two dimensions among the nodes, while reductions are achieved in place at each node. There is no dependency, thus no dataflow but collective communication among the participating nodes. Since the reduction is in place, the number of nodes in either dimension is independent of the inner dimension of the matrices. As a matter of fact, the nodes do not even have to be arranged physically in a two-dimensional topology as long as collection communication can be supported efficiently.

Today's distributed supercomputers are descendants of "Killer Micro" [20], which were considered aliens invading the land of supercomputing in the early 90s. As a matter of

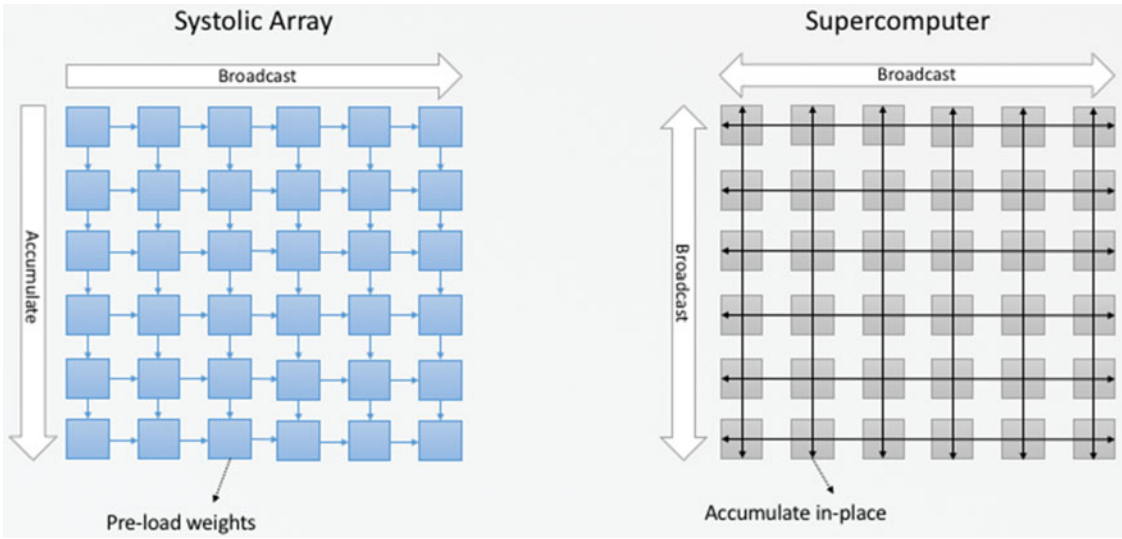

Fig. 11. Matrix multiplication on a systolic array and a supercomputer.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

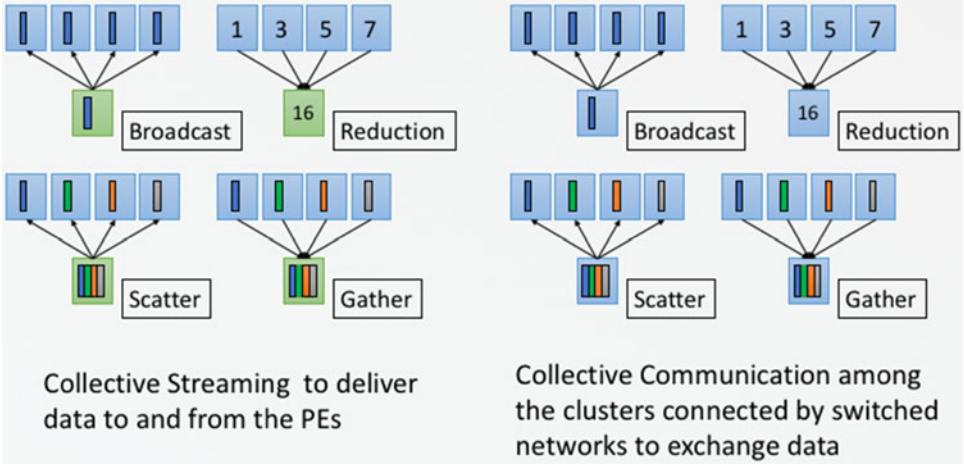

Fig. 12. Collective streaming versus collective communication.

fact, early supercomputers were purposely built to do matrix computations. Imagine that we build a supercomputer-on-chip by

(1) Shrinking a compute cluster to a PE with only densely packed MAC units,
(2) Building on-chip data delivery fabric to support collective streaming, reminiscent of collective communication in supercomputing.

Just as efficient collective communication can be achieved recursively, efficient collective streaming can be accomplished recursively through the building block, collective streaming Element (CE). The CEs are inserted between the PEs and the memory to broadcast or scatter the data hierarchically to the PEs, and to reduce or gather the results recursively from the PEs. The four operations are analogous to the counterparts in collective communication in supercomputing for the compute nodes to exchange data among themselves as shown in Fig. 12. Compared with systolic arrays, the PEs do not have to be interlocked in a two-dimensional grid and the latency can be within a constant factor of a logarithm of numbers of PEs. Building a supercomputer-on-chip can be considered as an effort to return to the matrix-centric root of supercomputing. It is effectively a *Native Supercomputer* (see Fig. 13).

IX. WHY COLLECTIVE STREAMING?

In many *conventional parallel processors*, including the GPU, a core, as a universal computer, not only has to support many functions other than MAC, but also needs to retrieve data from the memory, expecting the data to be shared through memory hierarchy. As a result, it requires a significant investment in area and energy for generic functions, multiple levels of caches, scratch memory, and register files. Collective streaming allows the computing units to comprise only MAC units without a memory hierarchy.

In a *spatial dataflow machine*, such as a systolic array, a PE still keeps the legacy of a core having to communicate with other PEs. This causes latency and makes it difficult to scale. Collective streaming allows orders of magnitude more MAC units without sacrificing latency.

A *programmable dataflow machine* is expected to resolve the dependencies among fine-grain data items. Given that dependencies among data items are collective, the efficiency of a programmable dataflow machine to handle generic

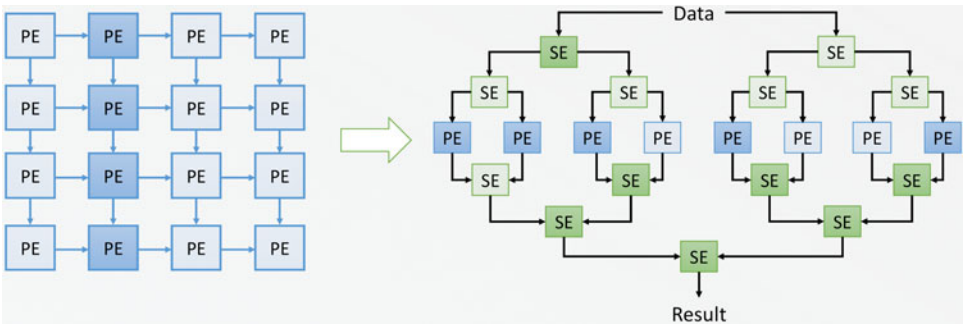

Fig. 13. From a mesh to a hierarchically organized PEs.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press

data dependencies will be worse than a spatial dataflow machine.

X. CONCLUSION

As mentioned earlier, Turing envisioned a universal AI machine modeling a human computer hired to do calculations with a pencil on paper. According to Turing, it is the software that tells it step by step what to do to make it think. With Deep Learning, the software will be like a CEO provisioning the resources and planning for the workflow to educate the machine. It involves formulating the time-consuming tasks through a software stack, running on legacy universal computers, into the most efficient computing resources.

The new AI machine will be built on top of the achievements of the previous computing revolution. However, the workhorse will be the computing resources that perform linear algebraic tasks natively.

AI was the inspiration behind the previous computing revolution. It shaped computing as we know it today. The history of computing now comes full circle. AI is coming back again to inspire computing. The quest to make machines think continues amid the slowdown of Moore's Law. AI might not only maximize the remaining benefits of Moore's Law, but also revive Moore's Law beyond the current technology.

REFERENCES

[1] Turing, A.: Computing machinery and intelligence. Mind, 50 (1950), 433–460.

[2] Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. Ser. 2 42 (1936), 230–265.

[3] Turing, A.: Programmers' Handbook for the Manchester Electronic Computer, *Manchester University*, Manchester, England, 1950.

[4] Neumann, J.v.: First Draft of a Report on the EDVAC. 1945. [Online]. Available: https://sites.google.com/site/michaeldgodfrey/vonneumann/vnedvac.pdf?attredirects=0&d=1.

[5] Moore, G.: Cramming more components onto integrated circuits. Electronics, 8 (1965), 82–85.

[6] Sutter, H.: The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software. 2005. [Online]. Available: http://www.gotw.ca/publications/concurrency-ddj.htm.

[7] Esmaeilzadeh, H.; Blem, E.; St. Amant, R.; Sankaralingam, K.; Burger, D.: Dark silicon and the end of multicore scaling, in The 38th International Symposium on Computer Architecture (ISCA), 2011, 365–376.

[8] Hodges, A.: Alan Turing. 30 Sep 2013. [Online]. Available: https://plato.stanford.edu/entries/turing/#Unc.

[9] BLAS (Basic Linear Algebra Subprograms). [Online]. Available: http://www.netlib.org/blas/.

[10] Cappello, F.; Guermouche, A.; Snir, M.: On communication determinism in parallel HPC applications, in Int. Conf. on Computer Communication Networks, Zurich, Switzerland, 2010.

[11] Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E.: cuDNN: Efficient Primitives for Deep Learning, 18 Dec 2014. [Online]. Available: https://arxiv.org/abs/1410.0759.

[12] Jouppi, N.P.: In-Datacenter Performance Analysis of a Tensor Processing Unit, Google, Inc, 2017. [Online]. Available: https://drive.google.com/file/d/0Bx4hafXDDq2EMzRNcy1vSUxtcEk/view.

[13] Kung, H.T.: Why systolic architecture? IEEE Comput., 15 (1) (1982), 37–46.

[14] Lu, C.-P.: Should We All Embrace Systolic Arrays? 28 April 2017. [Online]. Available: https://www.linkedin.com/pulse/should-we-all-embrace-systolic-arrays-chien-ping-lu.

[15] Luo, T.; Liu, S.; Li, L.; Wang, Y.; Zhang, S.; Chen, T.; Xu, A.; Temam, O.; Chen, Y.: DaDianNao: a neural network supercomputer. IEEE Trans. Comput. 66 (2017), 73–88.

[16] Sze, V.: Efficient Processing of Deep Neural Networks: A Tutorial and Survey. 27 Mar 2017. [Online]. Available: https://arxiv.org/1703.09039.

[17] Fisher, D.C.: Your favorite parallel algorithms might not be as fast as you think. IEEE Trans. Comput., 37 (2) (1988), 211–213.

[18] Robert A. van de Geijn, J.W.: SUMMA: Scalable Universal Matrix Multiplication Algorithm. Technical Report UT CS-95-28, pp. 255–274, Vol. 9. Department of Computer Science, The University of Texas at Austin, 1997.

[19] Thakur, R.; Rabenseifner, R.; Gropp, W.: Optimization of Collective Communication Operations in MPICH. 2005. [Online]. Available: http://www.mcs.anl.gov/~thakur/papers/ijhpca-coll.pdf.

[20] Brooks, E.: The attack of killer micros, in Supercomputing 1989, Reno, NV, 1989.

Dr. Chien-Ping Lu is currently the VP Hardware Engineering at NovuMind Inc. He is responsible for the R&D of hardware acceleration of Deep Neural Networks. Prior to NovuMind, Dr. Lu was a senior director of advanced graphics development at Intel. From 2011 to 2015, Dr. Lu was a senior director at MediaTek, where he successfully led the in-house GPU project from ground up, and co-founded HSA (Heterogeneous System Architecture) Foundation with AMD, ARM, Imagination, TI, Qualcomm and Samsung to push Heterogeneous Computing. From 2002 to 2011, Dr. Lu was GPU Architect and Senior Architecture Manager at NVIDA. Dr. Lu participated and delivered several important GPU products. Dr. Lu got his PhD in Computer Science from Yale University in 1995. He was one of the early researchers in Neural Networks in 90s. His Orthogonal Iteration algorithm for pose estimation has been widely adopted, cited and improved upon in Vision, Robotics and Augmented Reality communities.

https://doi.org/10.1017/ATSIP.2017.9 Published online by Cambridge University Press