Rebooting Computing, computing, instruction set-architecture, codesign, architectural simulation, von Neumann architecture, feedback loop, simulation, architecture optimization, computational accelerator, accelerator

Editor: Erik P. DeBenedictis, Sandia National Laboratories; epdeben@sandia.gov

Rebooting Computing

Computer Architecture's Changing Role in Rebooting Computing[1]

Erik P. DeBenedictis, Sandia National Laboratories

Researchers are now considering alternatives to the von Neumann computer architecture as a way to improve performance. The current approach of simulating benchmark applications favors continued use of the von Neumann architecture, but architects can help overcome this bias.

Windows 95 started the Wintel era, in which Microsoft Windows running on Intel x86 microprocessors dominated the computer industry and changed the world. Retaining the x86 instruction set across many generations let users buy new and more capable microprocessors without having to buy software to work with new architectures.

Today's more complex applications and consumers' shift to mobile devices have increased the critical need for energy efficiency and for smoothing the software-upgrade process. This has deeper implications for instruction sets and architectures than many people realize.

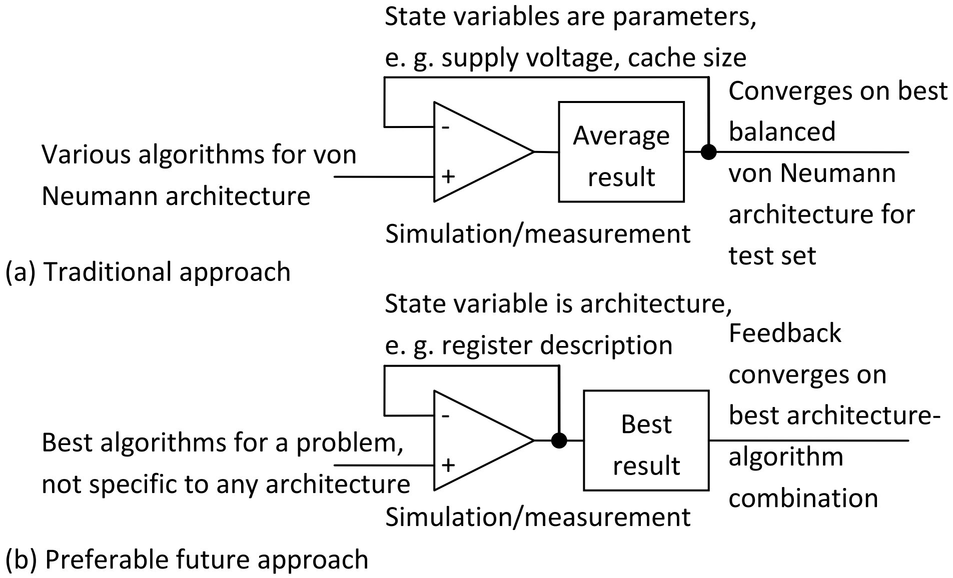
Microprocessor vendors are now able to change architectures even at the instruction-set level if they first remotely upgrade users' software to make it compatible with both the current and new architectures. A computer's architecture has served as a communication interface between architects and programmers. Architects minimized changes to avoid forcing programmers to perform costly rewriting. However, freezing the instruction set reduces the architect's flexibility when trying to increase energy efficiency. It would be more effective for architects to develop more energy-efficient architectures with less regard to instruction-set changes and let the marketplace decide whether the energy savings outweigh the cost of rewriting software.

[1] Sandia National Laboratories approved for unlimited unclassified release SAND2017-9063 J Published as DeBenedictis, Erik P. "Computer Architecture's Changing Role in Rebooting Computing." *Computer* 50.4 (2017): 96-99. **DOI:** 10.1109/MC.2017.99

This illustrates the need to change the process of optimizing architectures for existing software to one that can find new architectures.

Computer Architecture Simulations

Computer architects extensively use simulations to optimize architectures for consistent performance across a suite of benchmarks or test applications. Architects use a feedback loop on architectural parameters such as clock rate and cache size to find their optimal values, as Figure 1a shows. Simulations or measurements on test hardware measure a hypothetical computer’s speed and energy efficiency, yielding a continuous quality value. For example, a simulation might estimate the performance over a suite of benchmarks to be 57.8 computations per second. Architects would then feed the quality value back into the process as they adjust a parameter, such as cache size. A subsequent simulation might improve the rate to 59.3 computations per second. The architect repeats the loop until the values are close to optimal.



(a) Traditional approach

(b) Preferable future approach

Figure 1. Architectural simulation as an analog feedback loop. Simulations or measurements test a computer architecture’s ability to execute software over a range of changes to parameters such as clock rate and cache size, leading to iterative architectural changes. (a) The traditional approach tests architectures over a range of continuous-valued parameters, favoring ongoing use of the system’s original, typically von Neumann, architecture. (b) An alternative approach would work over a range of discrete (noncontinuous) algorithms and architectures, which better enables discovery of non–von Neumann architectures.

Each benchmark or test application originates with a widely used program, so each test embodies the best algorithms for the current architecture. Some simulations go a step further and use instruction traces collected from existing code’s inner loops, meaning the results are based on the instruction set of the computer that created the trace. However, carrying artifacts of the current architecture into simulations inhibits change.

To illustrate the problem, microprocessor benchmark sites such as CPUBoss (www.cpuboss.com) show that comparing the software benchmark performance of a modern laptop's microprocessor with one that is five years old has barely improved. Computer performance has actually increased substantially during this time, but the performance gain is mostly for new code running on new processors. Because benchmarks are usually run only when a processor is first introduced, results for new code are not listed for older processors.

Computer architectures and algorithms include discrete structures that aren't amenable to continuous parameter optimization. A computer's architecture and each algorithm that runs on it can be depicted as a graph with computations in boxes and data movement shown by lines. Typical hardware boxes contain processors, caches, and memory, as well as lines that represent buses or other interconnects. Algorithm graphs are more specific, with boxes containing mathematical calculations, sorting operations, or storage operations for specific data structures. Execution efficiency depends on the way the algorithm's graph maps to the hardware graph.

While Figure 1a's feedback loop can optimize continuous parameters, finding new architectures and algorithms involves higher-level intellectual processes that have resisted automation. For example, Alan Turing and John von Neumann used the exclusively human resource of brainpower to devise the discrete, graph-level representation of today's computers and algorithms.

Figure 1b includes two changes to Figure 1a's process to create an approach that finds new non–von Neumann architectures.

First, simulations should run a broad range of algorithms for a particular problem without the testers downselecting the algorithms to fit specific architectures. For example, a test set could include multiple algorithms for sparse matrix multiplication instead of a specific algorithm or instruction trace. Applying the feedback process to the best algorithm for a dataflow or processor-in-memory architecture should optimize those architectures instead of converting them back to the von Neumann style.

Second, changing the feedback position between Figures 1a and 1b lets the process find the best architecture for a specific problem instead of finding the architectural compromise that yields the best average results over multiple problems without excelling at any single one. Figure 1b's process finds accelerators,[1] rather than general-purpose processors. With the Moore's law–enabled growth in transistor count, today's computers contain accelerators—which are specialized functional units for graphics, encryption, radio, and so forth—in addition to a von Neumann processor. Once Figure 1b's process defines suitable accelerators for important tasks, architects would use Figure 1a's continuous variable optimization process to determine the resources that should be devoted to each accelerator, ultimately yielding a hybrid of a von Neumann processor and several accelerators**.**

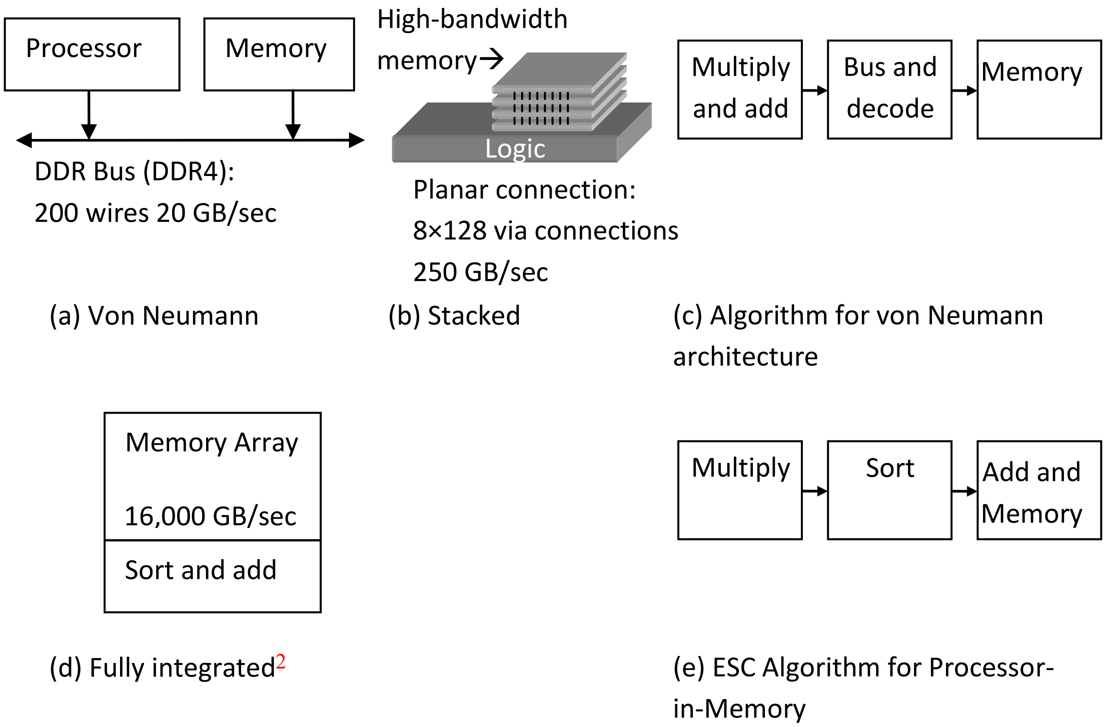
Example: Sparse Matrix Multiply

Realizing the benefits of the process illustrated in Figure 1b requires a technology advance best suited to a new architecture. Otherwise, the process will unhelpfully rediscover an existing architecture. A suitable advance is 3D stacked memory, whose best known examples are hybrid memory cube (HMC) and high-bandwidth memory (HBM). Academic visionaries see stacked memory as an intermediate step toward systems with fully integrated logic and memory.[2] We'll use HBM's second generation—HBM2—in subsequent examples.

First, we'll illustrate the performance potential of stacked memory on a sparse matrix multiply problem. Sparse matrix multiply is the most important step in some important scientific codes, such as multigrid solutions to partial differential equations. The task is to compute $C = AB$, with A, B, and C being sparse matrices.

In matrix notation, a matrix M is comprised of elements m_{rc}, where r and c are the row and column indices, respectively. Elements of the matrix product c_{ij} are a sum of products, sometimes called the dot or inner product, of a row of A multiplied by a column of B, $c_{ij} = \sum_k a_{ik}b_{kj}$. This dense matrix multiply is executed efficiently as vectors.

If the matrix is sparse, perhaps 99.999 percent of the products $a_{ik}b_{kj}$ are zero, due to one or both variables being absent and assumed to be zero. As a result, the terms used to compute a specific c_{ij} occur at different times interleaved with the calculation of other c_{ij} instances, which requires looking up the partially summed cij instances in memory before adding to them. While this may sound easy, it leads to billions of essentially random memory updates out of a gigabyte-size pool of memory, as Figure 2a illustrates. This makes sparse matrix multiplication on a von Neumann computer inefficient due to resource-intensive random-access memory activity.



(a) Von Neumann

(b) Stacked

(c) Algorithm for von Neumann architecture

(d) Fully integrated[2]

(e) ESC Algorithm for Processor-in-Memory

Figure 2. (a) and (b) Two von Neumann architecture implementations for sparse matrix multiply. (c) The corresponding discrete architectural representation of the architecture and algorithm. (d) An integrated processor-memory architecture. (e) A different discrete representation, using the Expand-Sort-Compress (ESC) algorithm, that is more suitable to the integrated architecture.

A Sandia National Laboratories study benchmarked a series of sparse matrix multiply algorithms on various types of hardware, including a system based on Intel Knight's Landing processors. This system had double data rate synchronous DRAM (DDR SDRAM) channels and 3D HBM stacked memory, as illustrated in Figure 2b. HBM's wider data buses, resulting from the use of high-density vias running through the silicon rather than PC board traces, increases the processor-memory bandwidth about 10 times, from DDR SDRAM's 25 GBps to 250 GBps. Both systems used the same discrete structure illustrated in Figure 2c.

A New and Different Approach

While 3D memory increases bandwidth and eases the memory bottleneck, fully integrating logic and memory like the structure shown in Figure 2d eliminates it entirely. [2]

DRAM refresh requires each bit to be read and rewritten every 64 ms. DRAM is organized as banks of 8,192 rows, requiring about 0.25 ms for a full memory refresh. The DRAM's internal data rate of reading and rewriting HBM2's maximum 8-Gbyte memory stack in 0.25 ms is equivalent to 32,000 GBps, which is 1,200 times faster than DDR SDRAM and 120 times faster than the stacked memory shown in Figure 2b.

The recent Sandia Labs study benchmarked the Expand-Sort-Compress (ESC) algorithm,[3] whose block diagram is shown in Figure 2e. In lieu of executing the statement $c_{ij} = c_{ij}+a_{ik}b_{kj}$ all at once, the ESC algorithm streams records { i, j, $a_{ik}b_{kj}$ } produced after the multiplies are performed into a memory array. The array is then sorted using *i*, *j* as the key, arranging records so that the ones with the same index are next to each other. The additions are then performed in an efficient array scan. This alternative algorithmic approach uses more bandwidth yet is more efficient because it has a regular access pattern. The approach is not very effective on the standard von Neumann architecture shown in Figure 2a because of its memory bottleneck, but is more effective with the increased bandwidth available from the 3D memory in Figure 2b.

Current DRAM architectures can't alter data during refresh and as a result cannot perform sorting or addition for these algorithms. However, the fully integrated architecture of Figure 2d would support the algorithms and yield higher performance.[2]

Comparing Approaches

The type of continuous variable optimization in Figure 1a can iteratively adjust clock rate or cache size to achieve an architecture that is well balanced across an entire application suite. However, optimizing continuous variables can't separate the addition and multiply operations found in a single box in Figure 2c into the two boxes shown in Figure 2e, thus preventing discovery of the new architecture in Figure 2e, which could offer higher performance. Only humans can create such block diagrams because designing architectures and algorithms uses higher-level intelligence. Genetic programming is perhaps the only artificial-intelligence method that could approach this task, but it currently works only for trivial problems due to poor scaling.

3D memory as shown in Figure 2b could evolve into integrated logic and memory as shown in Figure 2d in a decade or more, repeatedly optimizing the architecture for rebooting computing by creating a path to continued scaling. However, this approach's credibility is based on 3D memory, which became available only in the last few years. Applying architecture tools to any such recent system would just point back to the von Neumann architecture.

Continued computing advances depend on finding energy-efficient alternatives to the von Neumann architecture. Current methods and simulation tools miss the target because they retain artifacts of the von Neumann architecture. However, I suggest the method can and should be adapted to test new architecture–algorithm combinations. This would create a hybrid human–computer system in which humans create architectures and algorithms using brainpower, and then computer simulation assesses the quality of each combination.

Acknowledgments

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corp., a wholly owned subsidiary of Lockheed Martin Corp., for the US Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

1. E.P. DeBenedictis and R.S. Williams, "Help Wanted: A Modern-Day Turing," *Computer*, vol. 49, no. 10, 2016, pp. 76–79.
2. M.M. Sabry Aly et al., "Energy-Efficient Abundant-Data Computing: The N3XT 1,000X," *Computer*, vol. 48, no. 12, 2015, pp. 24–33.
3. Dalton, L. Olson, and N. Bell, "Optimizing Sparse Matrix—Matrix Multiplication for the GPU," *ACM Trans. Mathematical Software*, vol. 41, no. 4, 2015, pp. 1–20.

***Erik P. DeBenedictis** is a technical staff member at Sandia National Laboratories' Center for Computing Research. Contact him at epdeben@sandia.gov.*