Superstrider Associative Array Architecture

Approved for unlimited unclassified release
SAND2017-7089 C

Erik P. DeBenedictis, Jeanine Cook
Center for Computing Research, Sandia National Labs
P. O. Box 5800 m/s 1319
Albuquerque, NM 87185-1319
epdeben@sandia.gov, jeacook@sandia.gov

Sriseshan Srikanth and Thomas M. Conte
School of Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
seshan@gatech.edu, conte@gatech.edu

***Abstract*—We define the Superstrider architecture and report simulation results that show it could be key to achieving HIVE hardware goals. Superstrider's performance comes from a novel sparse-to-dense stream converter, which relies on 3D manufacturing to tightly couple DRAM to an internal network so operations like merging and parallel prefix can be performed quickly and efficiently. With the ability to use the stream converter as a programming primitive, the memory-bound low-level graph operations that we are aware of speed up substantially. We give special attention to triangle counting in this paper.**

Simulations detailed elsewhere[1] show 50-1,000× improvement in speed and energy efficiency. The low end of the range should be achievable by constructing a custom controller for current High Bandwidth Memory (HBM) where the high end would require fully integrated 3D that is on roadmaps for the future.

Keywords—Superstrider; Moore's law; 3D chips; sorting, processor-in-memory; sparse matrix; backpropagation; associative array; GraphChallenge

I. Introduction

The world went through an information revolution driven by the exponential growth of microprocessor performance and DRAM size popularly called Moore's law. The flat lining of microprocessor performance has led to talk of "Moore's law ending" and dire consequences to the economy. However, the vertical axis on the graph defining Moore's law in Fig. 1 is clearly labeled "number of components per integrated function" (chip) not "microprocessor performance" and some companies find they can maintain growth in component count by using the third dimension to stack more devices per unit surface area. It seems the economy and Moore's law are healthy, but the top-level division of computers into microprocessors and DRAM must change.

Could GraphChallenge and the associated HIVE hardware project have a role in the future direction the industry? The organizers of GraphChallenge make a compelling case that important new classes of applications conflict with von Neumann's division of computers into processor and memory, which we argued in the paragraph above is limiting the industry as a whole. If the HIVE project develops a graph architecture that scales in a way that is compatible with physical-level semiconductor roadmaps, perhaps HIVE could help define the mainstream direction of the industry?

A. Moore's law and 3D

Even though industry wanted to discover a new transistor for logic, it actually developed new memory devices that can be manufactured in 3D. Developments in 3D chips have been compelling enough that there are now roadmaps for further staged development.[3] The roadmaps show a path through intermediate technologies with progressively more features in the third dimension, a greater variety of devices, and more efficient transfer of information along the third dimension.

There are two 3D commercial product categories right now. Stackable DRAM is the first, available as High Bandwidth Memory (HBM[4], illustrated in Fig. 2a) and Hybrid Memory Cube (HMC[5]). HBM transfers 16,384-bit DRAM rows as 128 cycles of 128 bits. 3D flash storage is the second product category.

There are several visions for long term development of fully integrated memory, storage, and logic, one example being N3XT[6] illustrated at the top of Fig. 2b. The tighter integration of the proposed N3XT system would have no reason to limit the width of the memory interface, so for consistency we assume a 16,384-bit interface that operates in 1 cycle. In fact, a scale up path can be defined based on the tightness of logic-memory integration, as shown in Fig. 2c.

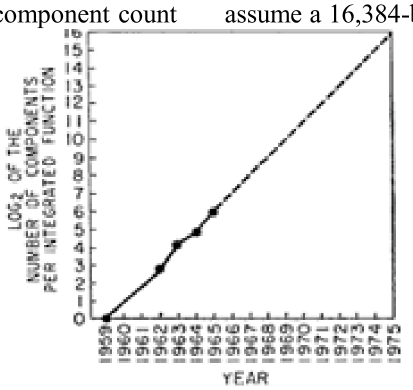
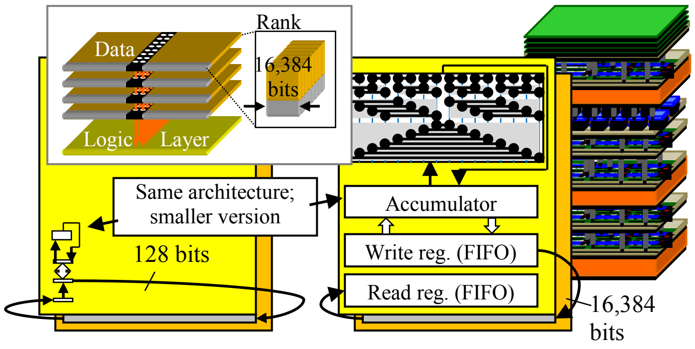


Fig. 1. Graph defining Moore's law from Moore's paper,[2] projecting that device count per chip will rise exponentially for the decade 1965-1975.

II. Approach

Fig. 3 shows the basis of our approach for exploiting 3D memory to improve graph and other computations. Industry's historic choice to manufacture DRAM and microprocessors as separate chips caused a "data modality" problem. Fig. 3a applies to essentially any multi-chip computer system, showing yellow logic chips, orange memory chips, and connected by gray chip-to-chip interconnect. Algorithms where information

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.



(a) HBM (Now) (b) N3XT (Future)

HBM Row: 16,384 bits =

Interface width ×	Cycles	Possible network size
128 bits (HBM)	128	8 = 4 × 2
256	64	24 = 8 × 3
512	32	64 = 16 × 4
1,024	16	160 = 32 × 5
16,384 (N3XT)	1	4,608 = 512 × 9

(c) Scaling scenario

Fig. 2. (a) HBM physical structure on top and Superstrider layout on bottom. (b) N3XT physical structure on top and functionally similar but scaled Superstrider layout on bottom. (c) Hypothetical scaling sequence based on doubling interface with, halving clock periods, and increasing merge network depth.

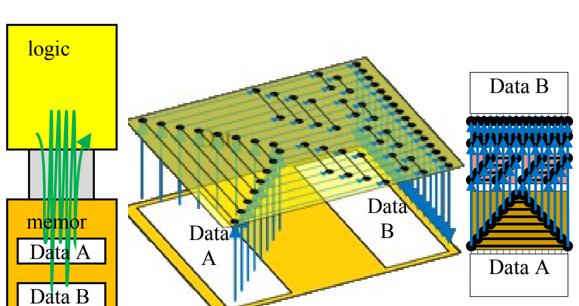
dependencies alternate many times between logic and a large memory (i. e. a memory too large for implementation by cache) inevitably involve moving data across the interface between the two twice per repetition, or shifting modes, as illustrated by the green curve. While the efficiency of electronics entirely within a chip improved with Moore's law, the speed and energy efficiency of chip-to-chip interconnect scaled more slowly, creating a bottleneck.

A. *Sorting networks*

We remove the bottleneck using the additional design flexibility offered by 3D. If memory can be stacked or monolithically fabricated adjacent to logic along the chips' planar faces, not only are wires shortened by the tighter packing allowed by the additional dimension, but the modality problem is relieved because data movement between logic and memory no longer needs to traverse chip-to-chip interconnect.

The shift to 3D offers algorithmic benefit as well, which we will discuss using sorting as an exemplary algorithm class. Sorting has been studied extensively for the structures in Fig. 3a and b, where sorting networks, such as the O($\log^2 n$)-step bitonic sort[7] in Fig. 3c, have been known to be faster for decades – but there is more to the story.

Fig. 3c shows the data dependency diagram for one merge stage of bitonic sort. The diagram shows the merging of two sorted lists of 8 data records, which occurs in 4 stages. The first stage does pairwise comparison of all 16 values, swapping the records so the largest one is on the left. The second and later stages do pairwise comparisons as well, but on more groups of fewer records. The pattern can extend to data that fills one or more rows of DRAM. When implemented with the structure in Fig. 3b, the comparisons and swaps can be laid out on the surface of the logic layer as shown. The interface between



(a) 2D network

Fig. 3. (a) 2D systems comprise logic and memory chips, with the green curve illustrating a mixed logic-memory calculation that is inefficient precisely because of the partitioning into logic and memory. (b) A 3D system with tight coupling between logic and memory avoids high latency paths, bandwidth bottlenecks, and conversion of signals to high energy levels for off-chip interconnects. The blue curve shows a representative data movement step in sorting. (c) The sorting network used for the 3D example.

logic and memory will be via wires crossing the short gap between the large 2D surfaces.

However, the literature for sorting algorithms follows software conventions, which are modeled on a von Neumann computer and hence the structure in Fig. 3a. A von Neumann computer was originally viewed as doing one thing at a time – although recent multicore computers can do up to, say, a dozen things at a time. However, no von Neumann computer comes anywhere close to being able to simultaneously operate on the data in an entire row of DRAM at once. This is the rationale for the Wikipedia page on "Sorting Algorithm" saying "[p]ractical general sorting algorithms are almost always based on an algorithm with average time complexity (and generally worst-case complexity) O($n \log n$)."[8] The best known such algorithm is Quicksort.

Which sorting algorithm is fastest thus depends on whether Fig. 3a or b is the model for implementation. A von Neumann computer does the compare and swap operations one at a time, raising the bitonic sort's complexity from O($\log^2 n$) to O($n \log^2 n$). Thus, Quicksort's O($n \log n$) complexity makes it faster than bitonic sort's O($n \log^2 n$) complexity on a von Neumann computer, but the winner shifts to bitonic sort for implementation in 3D using Fig. 3b, allowing sorting O($\log^2 n$) steps, which is much faster.

B. *Superstrider block diagram*

Superstrider comprises a DRAM memory bank connected to a functionally enhanced butterfly network and a control system, as illustrated in Fig. 4. Functionally, Superstrider is in a never-ending loop reading and immediately writing back DRAM rows. However, the control system selects the row and also uses functional features added to the butterfly network like reduction and parallel prefix to modify the data between read and write back.

Here are a few more details: A small portion of the DRAM row width (a few dozen bits) is dedicated to control fields as indicated in Fig. 4, but the rest comprises *K* data records. There is also an accumulator that, in conjunction with the DRAM's

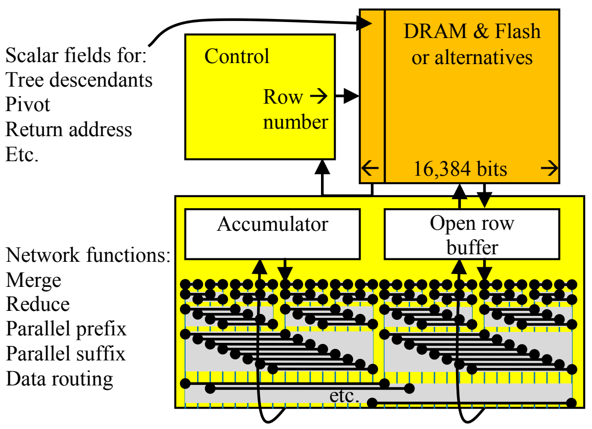


Fig. 4: Superstrider block diagram. A memory bank connects to the logic across its entire width, typically 16,384-bits. The logic comprises a butterfly network with functions listed, connected to both the memory and an accumulator (in the historical sense of the word). A control system implements a basic cycle where DRAM is read and written back continuously as fast as possible. Superstrider function change data between read and write, including storing data in the accumulator.

open row buffer, form a 2*K*-record buffer; the butterfly network actually operates on this double-length buffer. While not shown, the memory is accessible to an external, von Neumann-type, processor.

C. The sparse-to-dense stream converter

Imagine the pyramidal Champagne tower in Fig. 5a to be a physical analogy of element insertion into a binary tree data structure, where each Champagne glass corresponds to a DRAM row holding *K* red or green records. A regular tree insertion subroutine searches from the root downward through descendant nodes until the proper one has been found, after which the record is stored in its proper place and the subroutine returns. However, pouring Champagne into the tower has a feature not found in any computer algorithm (as far as we know). Most of the Champagne poured into the top glass does not end up in its final glass after just one pouring, but will temporarily reside in glasses further up the tree until a later pour causes it to move downward.

The problem is resistance to parallelization. If *K* records are added at the root all at once, like pouring a glass of red and green records into the top glass, there will be more groups of fewer records moving at each level, as shown in Fig. 5a. Since each glass is analogous to a DRAM row, almost all the DRAM rows in the entire system will be accessed for even modest values of *K*, which is why the structure in Fig. 3a experiences a lot of inefficient data movement between logic and memory chips for many problems.

Fig. 5b illustrates the idea behind Superstrider's sparse-to-dense stream conversion. When *K* records are poured into a glass that contains another *K* records, Superstrider sorts these 2*K* records immediately and puts them back in the glasses, but it carefully picks which glass gets the red versus green records. Using the terminology of tree algorithms, each row has a pivot. We'll color records red if their key is less than the pivot and green otherwise. There are now two scenarios corresponding to the bottom halves of Fig. 5b. If sorting reveals *K* or more red

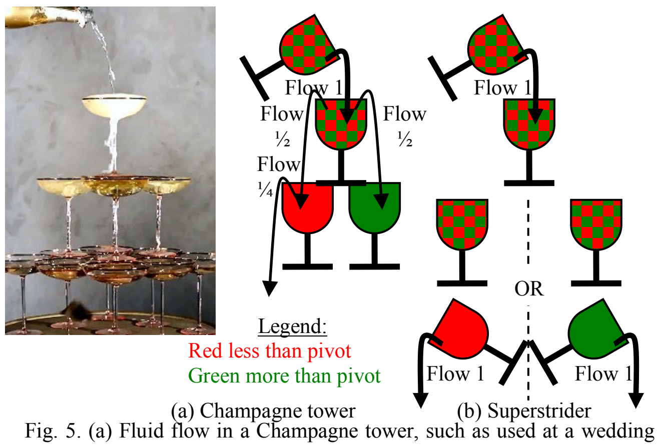


Fig. 5. (a) Fluid flow in a Champagne tower, such as used at a wedding reception, and a tree insertion algorithm. Imagine a Champagne glass holds *K* red or green records in lieu of Champagne molecules. Pouring *K* records into a glass or tree node recurses to two similar operations of size *K*/2, ultimately disturbing nearly every glass in the tower. (b) Superstrider can sort based on comparing keys with a pivot, where the comparison result is illustrated by color, yielding asymptotically fewer steps. If the amount poured at one time is the same as the capacity of a glass tree node (*K* records), sorting the 2*K* items by color is guaranteed to produce at least *K* items of the same color. Recursion will be needed only down the branch with the predominate color (i. e. tail recursion) and the algorithm will have just O(log *N*) steps for *K* records. (Photo https://vimeo.com/114254175, labeled for noncommercial reuse.)

records, one glass will be entirely red and the other glass will have mixed colors. If there are less than *K* red records, there must be more than *K* green ones, so the previous statement will be true with the colors interchanged. Now pour the glass where all the records are the same color down just one subtree and leave the other glass in its position in the tree with both colors.

The catch is that the records are not properly added to the subtree that is not visited. So does Superstrider go back and visit the subtree with incompletely processed records? No. That subtree is ignored for the time being. Graph problems typically do long sequences of additions, so subsequent sorts of that tree node will eventually produce enough records of the other color that the subtree will be visited naturally. Simulation shows this to be very efficient for long sequences of additions, yet a cleanup phase, which we call normalization, is required at the end to move laggard records to their proper destination.

The step count reduction is significant and arguably an "order reduction." Adding a single element to a tree is an O(log *N*) operation in standard algorithm theory, where *N* is the number of elements in the tree. Superstrider can add *K* elements in this amount of time, making the step count per record added O(log *N*)/*K*, which captures parallelism but is not an order reduction. However, DRAMs are designed to be refreshed in 8,192 cycles, meaning they actually have 8,192 rows at some low level of the DRAM electronics. If the number of rows is fixed, $K \propto N$, and the step count reduces to O((log *N*)/*N*), which is an order reduction. This argument is rather abstract. If the reader prefers, Ref. 1 shows speedups of 50-1,000×.

D. Connection to associative arrays

The preceding description spoke of a 16,384-bit DRAM row being divided into a few dozen control bits and the remainder into K records. In the notation of associative arrays,[9] each record is of the form $\{ k, v_k \}$, where k is a key and v_k is a value. If the record is an int and a float, the record is 64 bits and K is about 256 records. The DRAM may contain multiple arrays such as **A**, **B**, and **C** where $\mathbf{A}[k] = v_k$.

Previous sections of this paper only referred to sorting records, but the butterfly network also uses the network functions listed in Fig. 4 to identify collisions and compress records. This means the sparse-to-dense stream converter performs accumulation part of sparse matrix multiply.

The Superstrider concept is generally compatible with generalizations of associative arrays, but low-level hardware would need to process diverse data types and aggregations beyond what we have actually studied. The Superstrider concept should support associative arrays defined as $\mathbf{A}[\mathbf{k}_i] = v_i$, where **A** is the associative array, $\mathbf{k} = (k^1 \ldots k^d)$ is an array of indices of primitive types, and v may be a data structure of primitive types. Furthermore, even values that are data structures must be an element of a semiring, in which case there will be application-specific operators corresponding to multiplication and addition ($\otimes$ and $\oplus$).

Given the previous definitions, Superstrider could execute graph algorithms, like shortest path, where the values are floats or integers (albeit extended to include ∞) and the operation is $\oplus.\otimes \equiv$ min.+.

While certain simple kernels can be expressed in associative array syntax like $\mathbf{C} = \mathbf{A}$ +.* **B**, many of the more complex algorithms involve hundreds or thousands of lines of conventional computer code interspersed with calls to compute-intensive associative array operations. Reproducing an example from Ref. 9, the Bellman-Ford algorithm for finding the shortest distance from a node s to all other nodes in a graph **A** appears below, where only the red characters would be performed by Superstrider.

```
Bellman-Ford ($\mathbf{A}$, $s$)
$\mathbf{d} = \infty$
$\mathbf{d}(s) = 0$
for $k = 1$ to $N$-1
   do $\mathbf{d} = \mathbf{d}$ min.+ $\mathbf{A}$
if $\mathbf{d} \neq \mathbf{d}$ min.+ $\mathbf{A}$
   then return "A negative-weight cycle exists"
return $\mathbf{d}$.[9]
```

E. Superstrider programming strategy

The sparse-to-dense stream converter becomes a programming primitive of sorts, so we owe the reader a brief lesson on how to program with it. This introduction will lead up to the discussion of triangle finding requested by the Graph Challenge.

Sparse matrix multiply $\mathbf{C} = \mathbf{AB}$ can be performed with multiplications on a host or entirely stand alone. For the first option, Superstrider is initialized with an empty **C** matrix. The host then sends Superstrider vectors of K records of the form $[i, j] = c_{ij}^{(k)}$, where i and j are matrix indices, $c_{ij}^{(k)}$ is a contribution to a matrix element, with superscript $^{(k)}$ distinguishing between the contributions. After sending all the records, the host would command Superstrider to do the normalize function, which cleans up the representation so it could be read out by the host in a packed lexicographic order.

Alternatively, Superstrider could compute $\mathbf{C} = \mathbf{AB}$ autonomously, where matrices **A** and **B** are already stored in Superstrider's memory. This requires first transposing **A** from $\mathbf{A}[i, k] = a_{ik}$ to $\mathbf{A}^t[k, i] = a_{ik}$. This is accomplished by Superstrider accessing **A** internally, interchanging the indices, and sending the records into the sparse-to-dense converter as $\mathbf{A}^t$. This entire process is performed by the hardware in Fig. 4. Reading $\mathbf{A}^t$ later on will produce the records in lexicographic order by what was originally the second index. The actual multiply is then performed by streaming $\mathbf{A}^t$ and $\mathbf{B}[k, j] = b_{kj}$, each in lexicographic order, whereby the products $\mathbf{C}_{ij}^{(t)} = \sum_{k=1,t} a_{ik}b_{kj}$ can be formed by data close together in the streams. The products are then sent to the sparse-to-dense stream converter to creates **C**, using collision of the keys to perform the addition required in sparse matrix multiplication.

The delta rule in backpropagation of neural network learning, is defined as $\mathbf{C} = \mathbf{C} + \mathrm{ab}^t$, which we will perform with no fill-in in two steps. First, $\mathbf{C}[i, j] = c_{ij}$ is streamed along with $a[i] = a_i$ to produce a temporary matrix $\mathbf{T}[j, i] = \{ c_{ij}, a_i \}$. Note that **T**'s indices are reversed, so its records contain the elements of $\mathbf{C}^t$. In addition, **T**'s records are a data structure with both an element of $\mathbf{C}^t$ and an element of a. Secondly, $\mathbf{T}[j, i] = \{ c_{ij}, a_i \}$ is streamed along with $b[j] = b_j$ to produce $\mathbf{C}[i, j] = \mathbf{T}_{ji}.c_{ij} + b_j * \mathbf{T}_{ji}.a_i$, which naturally adds to the existing **C** when sent to the sparse-to-dense converter due to collisions. Since $\mathbf{T}_{ji}$ is a data structure, the notation $\mathbf{T}_{ji}.a_i$ represents the second element of $\{ c_{ij}, a_i \}$ given above (sorry, we need better notation for data structures).

F. 3D Streaming and triangle counting

Superstrider is essentially a linear algebra machine, so it should be able to execute any triangle-counting algorithm that can be expressed as linear algebra. The authors' attention was directed to the semi-streaming triangle-counting approach of Becchetti in Ref. 10, a paper from 2007. In short, we would put Becchetti's entire algorithm into Superstrider, but this simple statement has more to it than may be immediately apparent.

We propose going a step beyond Becchetti's view of semi-streaming.[10] Becchetti sees a computer as a CPU and DRAM in one unit and "disks" (yes, the article uses the word for rotating storage) on the other side of the cabinet. The large edge-containing matrices are on the disk but DRAM is only deemed big enough to hold vertex-containing matrices. If one were to implement Becchetti's system today, the disk storage would probably be 3D SSD. So when we say we would "put the entire algorithm into Superstrider," we envision 3D chips, such as N3XT in Fig. 2b,[6] with the ability to integrate both RAM-equivalent and storage-equivalent storage right on top of the same logic base chip. This leads to a second interpretation, illustrated in Fig. 4, where Superstrider's memory includes both DRAM and Flash in different ranges of row addresses, yet all appearing to Superstrider as a memory bank. This would allow Superstrider to, for example, multiply a matrix physically stored in Flash by one physically stored in DRAM just by properly specifying the row addresses. Let's call this

"3D steaming" because it works like streaming, but moves data just a few microns in the vertical direction.

Ref. 10's method is based on tagging vertices with random numbers or permutations of node numbers (π(.) in the document). With the possible exception of having a von Neumann host processor generate the permutations π(.), Superstrider could implement the algorithms in Ref. 10's using the linear algebra functions discussed above.

We therefore propose supporting triangle finding with a change in algorithm. Line 15 of figure 5 in Ref. 10 is executed when a triangle has been found, although in Ref. 10 it only triggers a counter increment. We propose that this line additionally create a 3-field record containing the triangle and send it to a sparse-to-dense stream converter designated to hold the output triangles.

We believe the algorithm as outlined so far will function, but it will still be probabilistic, generating the same triangle many times (the duplicates will be removed by the stream converter) and it may take a lot of runs to generate the very last triangle. So we propose modifying the tagging to effectively remove nodes once it has been determined that they have produced all the triangles they will ever produce. This can be done by tagging nodes with ∞ in lieu of π(.).

III. Experiments

We created a cycle accurate Superstrider simulator or 1,500 or 3,500 lines of C++, depending on whether the user interface code is counted. The simulator compares the accumulation phase of sparse matrix multiply against a von Neumann architecture baseline.

A. *Memory banks*

To make reasonable comparisons with a conventional processor, the simulator illustrated in Fig. 6 models 8 identical instances of Superstrider, each connected to an HBM channel. Each HBM channel comprises 8 16,384-bit wide physical DRAM banks. To consider the extreme range of algorithmic efficiency, each Superstrider row is set as wide as possible, i. e., $K = 2{,}048$ records or 131,072 bits wide.

The physical configuration of a bank and the timing parameters were obtained from the High Bandwidth Memory (HBM) JEDEC standard,[4] however we speculated on the timing of hypothetical HBM successors with wider interfaces per the scale up path in Fig. 2c.

B. *Front end*

A front end generates vectors of K sorted records to feed both Superstrider and the von Neumann baseline. The front end does not reorder the sparse matrices and the baseline does not simulate caches because there is negligible spatial locality in sparse matrices. The baseline traverses the HBM banks as rectilinear memory, i. e., without the binary tree format of the Superstrider algorithm. The front end supports matrices downloaded from the University of Florida[11] sparse matrix collection as well as pseudo-random sparse matrices generated on the fly. However, the results in this paper are for pseudo-random matrices with 27 million records where the keys are randomly chosen from a space of 27 million possibilities.

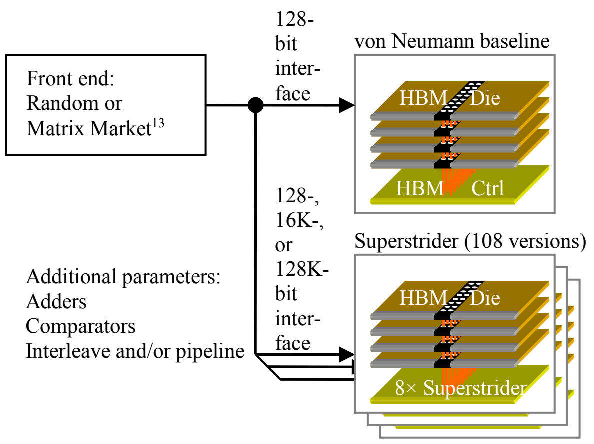


Fig. 6: Experimental framework. Baseline is an HBM stack with controller (8 channels) but no cache anywhere. For compatibility, Superstrider is implemented as 8 instances, one per HBM rank or channel, with a parameter sweep over additional parameters.

C. *Simulated parameters*

In the simulator, we vary the following five parameters to explore the design space: (1) the width of the logic/memory interface, (2) the size of butterfly network, (3) the number of adders comprising the adder network for collisions, (4) effects of adder partitioning schemes across Superstrider instances, and (5) resource pipelining.

IV. Results

A. *Data transfers*

The principal advantage of Superstrider is that it mitigates the von Neumann bottleneck by reducing the number transfers between the processor and memory, and the number of internal memory accesses. In conventional processors, cache-line utilization (including hardware prefetching) for sparse matrices is extremely low. In contrast, Superstrider makes effective use of an entire row.

The simulator counts the number of DRAM row accesses to estimate energy saved due to reduced memory traffic. Superstrider accesses 1/121 as many physical memory rows as the von Neumann baseline on the random test case, although this fraction is highly application dependent.

B. *Performance*

The vertical axis of the 3D bar charts in Fig. 7 is the speedup of Superstrider over the von Neumann baseline. Two messages are clear even at the low-level of detail in this paper.

1. Even the thinnest bars show a speedup of 50× or more, making Superstrider a candidate for implementation as a controller for off-the-shelf HBM memory.

2. Both the 16K and 128K interface widths include speedups of 1,000× or more, although only with 256 or more comparators. In simple terms, Superstrider can make very good use of tight memory-logic integration, but the interface can be

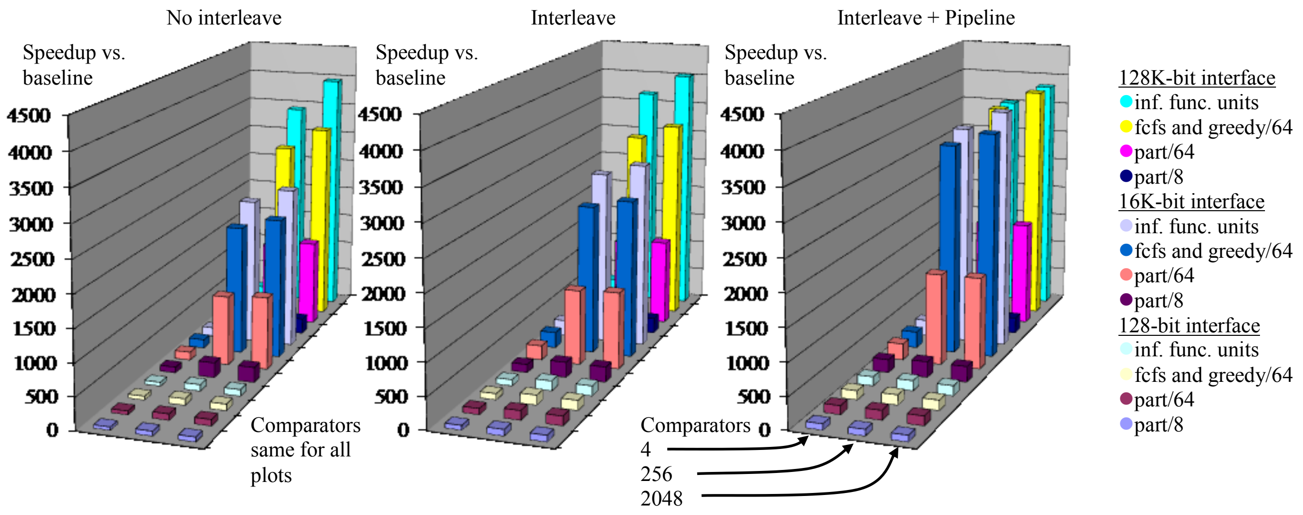


Fig. 7. Superstrider simulation results over the parameter sweeps discussed in the text. The vertical axis represents speedup over a non-cached sequential HBM driven by a microprocessor. The conspicuous result is that performance is high in the back, right hand row. This area corresponds to tight processor memory coupling (wide HBM burst width) and a large number of ports on the merging network. The other parameters are well-known techniques from microprocessor architecture, such as pipelining and interleaving. While these other parameters yield the expected benefit, it appears small in the context of the benefits of the non-von Neumann architectural components.

"choked" by insufficient computational resources. The resource is the comparators in this case, but see Ref. 1 for an explanation of this component and the other parameters.

V. Conclusions

In an environment rife with computer technology efforts to redesign computers from the devices up, we propose to address graph problems through carefully chosen computer architecture and algorithms yet based on just roadmaps of 3D chips. Our result of 50× immediate benefit and 1,000× or more in the long term is good enough to drive the industry and is on par with the more radical approaches.

Instead of writing a specific algorithm or buying or building a specific instance of hardware, we considered a computer whose design is parameterized by the date of manufacture and running software that adapts to the hardware as it changes. The principle parameter is the "tightness" of the logic-memory interface, where roadmaps can be consulted for a rough schedule of this parameter over time. We then created a cycle-accurate simulator and applied it to the most challenging computational kernel, which is sparse matrix multiplication and specifically the accumulation phase.

Algorithm theory showed that the tightening interface between memory and logic will make hardware implementation of sort, merge, parallel prefix, and some other network operations progressively more effective, where the "von Neumann bottleneck" makes them poor choices today. Using the principle of Occam's razor, we packed Superstrider with as much of the new hardware as possible and as few traditional components as absolutely necessary .

A. *Loose ends*

Superstrider is a chip-scale solution, but we need a way to glue *n* Superstriders together have the same effect as an *n*× bigger one.

We need to turn the simulator into a low-level linear algebra library and then run triangle counting algorithms, as requested by GraphChallenge.

It should be possible to configure Superstrider's internal state machine to do functions besides a sparse-to-dense stream conversion, although it will be a challenge to figure out the right functions.

Superstrider generalizes beyond DRAM.

B. *Crossing the valley of death*

We designed Superstrider to cross the "valley of death." In our view, an innovation needs immediate product potential and a long term vision to cross the valley:

For an immediate product, Superstrider could be created in an Intel/Altera Stratix 10 MX or Xilinx Virtex Ultrascale+ FPGA, both of which will come with on-package High Bandwidth Memory when they available (for engineering samples) in 6-12 months. They would implement Superstrider in the programmable logic of the FPGA and test the result. Simulations suggest about 50× improvement in speed for an HBM implementation, but the simulations do not account for the speed and energy efficiency penalty inherent in FPGA implementations. Of course, an FPGA test would reduce risk for an ASIC implementation.

Superstrider is part of a project to roadmap the continuation of Moore's law, which recently went down a dead end by limiting itself to the microprocessor and better transistors. The project's goal is to devise and promote architectures like Superstrider that can advance emerging applications areas like graph problems, and then roadmap these approaches in the IRDS to facilitate their funding. The plan is to roadmap a scale-up path like "Moore's law" based on progressively tightening logic-memory interfaces through 3D packaging, as shown in Fig. 2. This paper is one element of the strategy.

REFERENCES

[1] Srikanth, Sriseshan et. al. The same authors with author names in a different order, have submitted a paper to ICRC with details of the archtiecture. If not accepted, this other paper could be published as a Sandia tech report. "[the superstrider paper]." *IEEE ICRC* xx.y (2017): pp. If a reviewer wants to see this paper prior to its publication, it will be avialable temporarily at http://www.debenedictis.org/erik/ComputerPapers/GraphChallenge/ICRC_Superstrider_v3.4.pdf. Also, the simulator is online, open source, at http://www.debenedictis.org/SuperStrider.cpp.

[2] G.E. Moore, "Cramming More Components onto Integrated Circuits, Reprinted from Electronics, Volume 38, Number 8, April 19, 1965, pp. 114 ff," *IEEE J. Solid-State Circuits Newsletter*, vol. 11, no. 5, 2006, pp. 33–35.

[3] International Roadmap for Devices and Systems, http://irds.ieee.org

[4] High Bandwidth Memory (HBM) DRAM (JESD235), JEDEC, October 2013 http://www.jedec.org/standards-documents/results/jesd235

[5] Official website of the Hybrid Memory Cube Consortium http://www.hybridmemorycube.org/

[6] M.M. Sabry Aly et al., "Energy-Efficient Abundant-Data Computing: The N3XT 1,000X," Computer, vol. 48, no. 12, 2015, pp. 24–33.

[7] Batcher, Kenneth E. "Sorting networks and their applications." *Proceedings of the April 30-May 2, 1968, spring joint computer conference*. ACM, 1968.

[8] See https://en.wikipedia.org/wiki/Sorting_algorithm.

[9] Kepner, Jeremy, and John Gilbert, eds. *Graph algorithms in the language of linear algebra*. Society for Industrial and Applied Mathematics, 2011.

[10] Becchetti, Luca, et al. "Efficient semi-streaming algorithms for local triangle counting in massive graphs." Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.

[11] T. A. Davis and Y. Hu, "The university of Florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, 2011.