Memory-System Design Challenges in Realizing Monolithic Computers

Meenatchi Jagasivamani
Candace Walden
Devesh Singh
Luyi Kang
Shang Li
Department of Electrical & Computer Engineering, University of Maryland, College Park, MD

Mehdi Asnaashari
Sylvain Dubois
Crossbar, Inc., Santa Clara, CA

Bruce Jacob
Donald Yeung
Department of Electrical & Computer Engineering, University of Maryland, College Park, MD

ABSTRACT

This paper presents the notion of a *monolithic computer*, a future computer architecture in which a CPU and a high-capacity main memory system are integrated in a single die. Such computers will become possible in the near future due to emerging non-volatile memory technology. In particular, we consider using resistive random access memory, or ReRAM, from Crossbar Incorporated.

Crossbar's ReRAM is dense, fast, and consumes zero static power. Also, *it can be fabricated in a standard CMOS logic process*, allowing it to be integrated into a CPU's die. The ReRAM cells are manufactured in between metal wires and do not employ per-cell access transistors, so the bulk of the transistors underneath the ReRAM arrays are vacant. This means a CPU can be implemented using a die's logic transistors (minus transistors for access circuits), while the ReRAM can be implemented "in the wires" above the CPU. This will enable massive memory parallelism, as well as high performance and power efficiency.

We discuss several challenges that must be overcome in order to realize monolithic computers. First, there is a *physical design challenge* of merging ReRAM access circuits with CPU logic. Second, while Crossbar's ReRAM technology exists today, it is currently targeted for storage. There is a *device challenge* to redesign Crossbar's ReRAM so that it is more optimized for CPUs. And third, there is an *architecture challenge* to expose the massive memory-level parallelism that will be available in the on-die ReRAM. This will require highly parallel network-on-chip and memory controller designs, and a CPU architecture that can issue memory requests at a sufficiently high rate to make use of the memory parallelism.

ACM Reference Format:
Meenatchi Jagasivamani, Candace Walden, Devesh Singh, Luyi Kang, Shang Li, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. 2018. Memory-System Design Challenges in Realizing Monolithic Computers. In *Proceedings of International Symposium on Memory Systems (MEMSYS'18).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

MEMSYS'18, October 2018, Washington, D.C. USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The primary means by which computer hardware improves is via increased integration due to Moore's Law scaling. Over the years, Moore's Law has provided a steady improvement in computer hardware, but along the way, there have been notable points of disruptive change. These have occurred when significant components of a computer system become integrated onto a single die. For example, in the 1970s, the *microprocessor* enabled a single-die central processing unit (CPU). A more recent example is the *multicore processor*, which enabled single-die multiprocessors as well as single-die supercomputers, if one considers GPUs. In all of these cases, the integration of major compute blocks, previously fabricated in discrete chips, yielded significantly faster, less power hungry, physically smaller, and more reliable computer systems. Each of these benefits can be quite significant by itself, but having all of them occur at the same time can cause disruptive change in computing capabilities.

In this paper, we argue that we are on the cusp of another quantum leap in computing capability. This time, it will occur as the last major discrete component of a computer system moves into the CPU die: *main memory*. This will allow a complete computer–including multiple cores and a high-capacity memory system–to exist monolithically (*i.e.*, in a single die, not just across stacked dies). We refer to such computer chips as *monolithic computers*. This form of integration is inherently different from 3D-stacking of discrete dies [1].

Monolithic computers will facilitate physical proximity between the CPU and main memory that is unprecedented, enabling extreme wire density for implementing the CPU / main memory interface. This will allow massive memory parallelism which will improve performance, especially for highly parallel CPUs. The physical proximity between CPU and main memory will also permit lower energy per memory access since all memory transactions will traverse minimally sized wires. This will lead to much higher power efficiency, especially for memory-intensive workloads. And, monolithic computers will exhibit a smaller form factor since they require fewer discrete components, which will benefit embedded applications. Lastly, a reduction in the number of discrete components will also reduce failure rates, improving reliability. We believe all of these benefits will enable transformative change in computer systems' capabilities.

1.1 CPU - Main Memory Integration

A modern computer's main memory system is comprised of many discrete DRAM chips, integrated either into the CPU package via a silicon interposer or in separate packages on the system motherboard. While DRAM is an extremely mature technology, its continued scaling to advanced technology nodes is proving difficult. As a result, researchers have been investigating new memory technologies that may eventually replace DRAM. One promising candidate is resistive-memory based nonvolatile technology, such as resistive random access memory or *ReRAM*. A notable example is the 3D ReRAM from Crossbar, Incorporated. ReRAM is extremely dense, surpassing DRAM densities; it consumes zero power when idle; and it is relatively fast, currently able to achieve latencies in the 100s of nanoseconds. ReRAM is also much more scalable than DRAM, so its capacity and power advantages (along with its speed) will only improve over time.

Another interesting characteristic of ReRAM is that it can be fabricated in a standard CMOS logic process,[1] allowing it to be integrated into a CPU's die. Figure 1 illustrates how ReRAM can exist monolithically with the CPU. As shown in Figure 1a, the ReRAM memory cells are manufactured in between metal wires–*i.e.*, at the intersection of wires laid out perpendicularly in adjacent VLSI layers. Crossbar ReRAM bitcells are impervious to data-disturbance from access to adjacent column bit-cells as well as any logic activity from transistors beneath them. This gives rise to the so-called "crosspoint architecture." Rather than isolate individual ReRAM cells using access transistors, isolation is provided by per-cell "selector devices" integrated in the crosspoint arrays. The use of selector devices instead of access transistors enables extremely small memory cells. It also enables stacking of ReRAM cells vertically across multiple metal layers (see Figure 2), leading to very dense memory arrays. (We envision 100–200 GB of ReRAM could fit in a CPU die today, with even higher capacities possible as technology scales). But, it also means *the silicon area underneath the ReRAM are free for implementing non-memory circuits, like CPU logic*. Some logic is still needed per ReRAM "sub-array" for access circuitry (*i.e.*, decoders and sense amplifiers) but the bulk of the transistors are unused, as shown in Figure 1b. Hence, a CPU can be implemented in the die's logic transistors (minus those needed for ReRAM access circuits), while the ReRAM can be implemented "in the wires," floating above the CPU across the entire die.

The extreme physical proximity between the CPU and its monolithically integrated main memory system in Figure 1 will potentially enable massively parallel memory access. An important goal for monolithic computers is to realize this memory parallelism capability. This will require developing a highly parallel as well as a high throughput ReRAM memory structure. It will also require developing a CPU architecture that can sustain a high enough memory access rate to exploit all of the extant memory parallelism. We believe *tiled CPU architectures*, as illustrated in Figure 1c, are a good starting point for this purpose. Tiled CPUs, such as Intel's Phi [2] or Tilera's Tile processor [3], are comprised of multiple compute tiles interconnected across a 2D mesh network-on-chip (NOC), with each tile incorporating a NOC router that connects the tile to its nearest neighbors. (See Figure 1d). Due to their scalability, tiled architectures can support a large volume of memory traffic across the on-die NOC.

Of course, memory requests must eventually access main memory, which on existing tiled CPUs requires going off-die or off-package to discrete DRAM chips. Typically, a small number of DRAM memory controllers located at the periphery of the 2D mesh service such memory requests, which can become a bottleneck. For a monolithic computer, we propose to incorporate the memory controllers (MCs) into the compute tiles themselves, as shown in Figure 1d, with each MC responsible for servicing the requests to the ReRAM sub-arrays directly above the MC's compute tile.

To realize our monolithic computer concept, several challenges spanning multiple system abstraction layers, including *physical design, devices, and architecture*, must be overcome. In this paper, we discuss these challenges. The discussion proceeds by abstraction layer: Section 2 discusses physical design challenges, Section 3 discusses device challenges, and Section 4 discusses architecture challenges. Then, Section 5 concludes the paper.

2 PHYSICAL DESIGN CHALLENGE

Although the ReRAM memory cells of a monolithic computer are fabricated within the metal layers of a VLSI process, their access circuits must be integrated with the CPU's logic circuits. Hence, there is a challenge of how best to layout the access circuitry and CPU logic together. We refer to the process of physically laying out the access circuitry and CPU logic as "CPU-ReRAM stitching."

An important characteristic of ReRAM affecting CPU-ReRAM stitching is overhead. Access circuits, consisting of address decoders, wordline drivers, sense amplifiers, and mechanisms for driving locally sensed data to global buffers, are required for every ReRAM sub-array. The area overhead of these access circuits depends on the *sub-array size*. Larger sub-arrays permit greater amortization of the access circuits' area, and leave more area for CPU circuits. For the largest sub-arrays, Crossbar estimates that roughly *one quarter* of the area underneath each sub-array is needed to implement the access circuits, leaving *three quarters* of the area unused and available for compute logic. In other words, the access circuits incur non-trivial overhead, but the majority of die area is still available for building CPU logic.

We conducted a preliminary physical design study to illustrate the CPU-ReRAM stitching problem. In particular, we tried stitching ReRAM with a small synthesized core–*i.e.*, the RISC-V VSCALE core from U.C. Berkeley [4]. We performed the Auto-Place-and-Route (APR) using the open-source NCSU FreePDK 45nm process design kit and the Nangate open source digital library for the 45nm process node. We first generated the APR layout for the original RISC-V core. Then, we added a constraint in the form of 4 ReRAM sub-arrays located above the core. Each array has its access circuitry clustered along two of its sides. To minimize the interference with the core, we tried arranging the sub-arrays' access circuits back-to-back via rotations of 0, 90, 180, and 270 degrees. The total blocked-out area of the abutted access circuits is roughly 25% of the combined sub-array areas. (The area for the storage cells is not masked out because it lies in the metal layers directly above the array's access circuitry as well as the transistors of the core). After adding the constraints, we re-generated the RISC-V core layout.

[1] In contrast, DRAM requires specialized memory fabrication processes

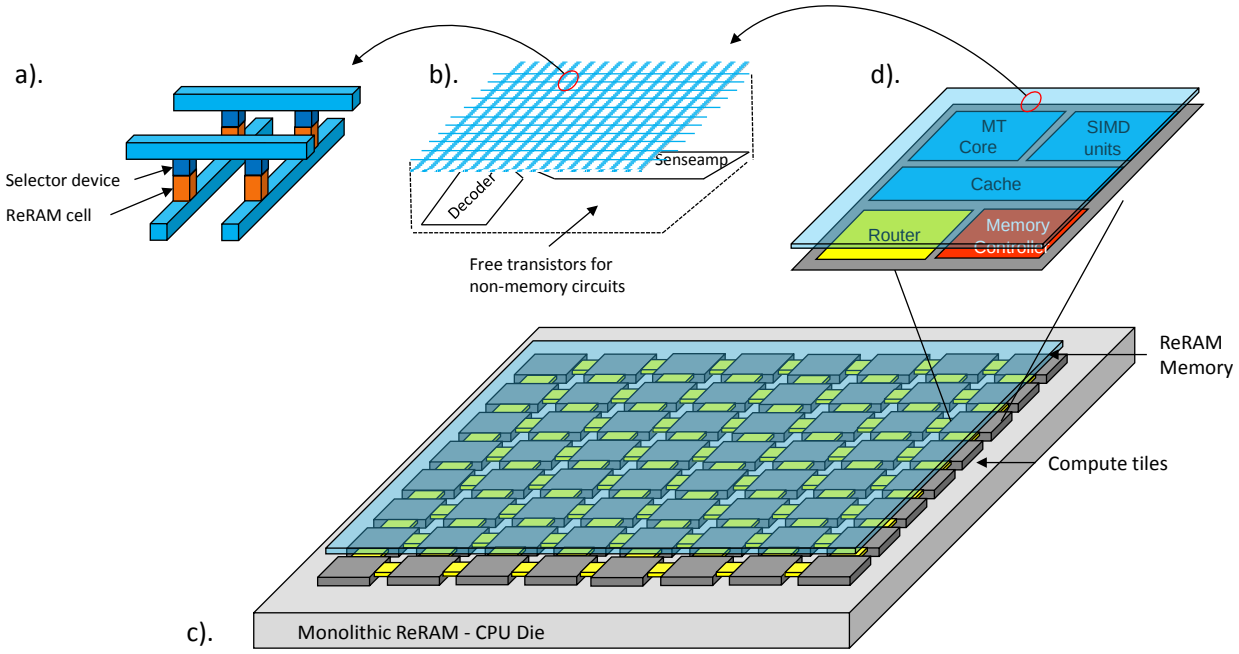

Figure 1: A ReRAM-based monolithic computer. a). ReRAM memory cells and selector devices. b). Integration over logic. c). A tiled CPU plus ReRAM main memory. d). Each compute tile integrates its own memory controller.

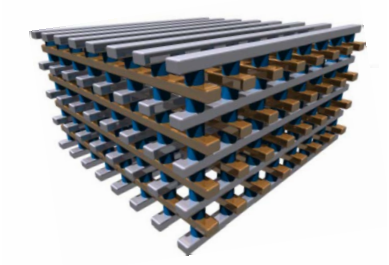

Figure 2: ReRAM arrays can be stacked vertically using additional metal layers to increase memory density and parallelism.

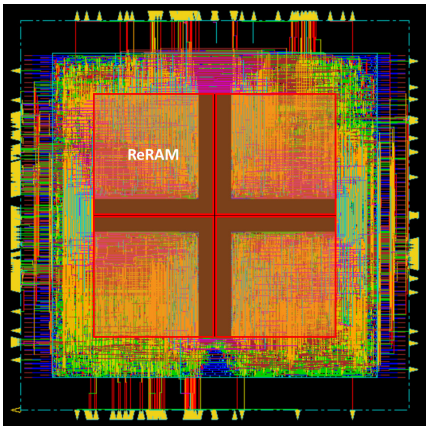

Figure 3: Placed & routed RISC-V VSCALE (integer) core and four ReRAM sub-arrays.

Figure 3 shows the final layout, including the routed wiring for the RISC-V core as well as the 4 mirrored ReRAM sub-arrays. Despite introducing a non-trivial blocked-out area for the ReRAM access circuits, the design tools were still able to successfully route the design. Overall, we found that the area of the core expands by 11.3% due to the introduction of the ReRAM. Given the area for the access circuits, the total area impact of introducing the ReRAM is about 36%. We believe this is an acceptable cost considering that we are integrating an entire main memory system into the CPU die.

This preliminary study illustrates the physical design problem of CPU-ReRAM stitching that designers will need to address in order to realize monolithic computers. But eventually, designers will need to consider much larger and more realistic cores. Given larger cores, there will be many more ReRAM sub-arrays stitched into the core (not just the 4 sub-arrays in Figure 3). There will also be a need to consider stitching ReRAM into regular structures, such as SRAMs. (The RISC-V core in Figure 3 does not include any cache). In addition, while we have only looked at area impact, it will be necessary to quantify the impact in terms of delay and power consumption as well.

3 DEVICE CHALLENGE

Crossbar, Incorporated has been manufacturing ReRAM commercially for over 3 years. The left side of Figure 4 shows a wafer

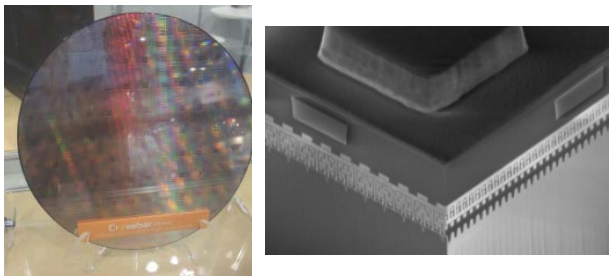

Figure 4: Crossbar ReRAM wafer and chip cross-section.

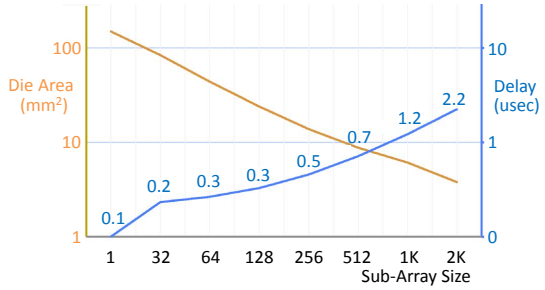

Figure 5: ReRAM read latency and area vs. sub-array size.

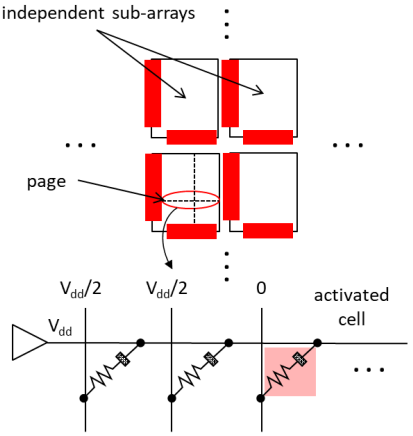

Figure 6: ReRAM sub-arrays have a small access granularity.

manufactured by Crossbar, and the right side shows a cross-section of one of their memory chips. As these photos demonstrate, the memory technology needed for monolithic computers largely exists. But Crossbar's current ReRAM cells are designed for storage devices, not CPUs. As such, there are drawbacks with using existing ReRAM for CPU main memory. One drawback is high access latency.

ReRAM's access latency depends on the sub-array size. Larger sub-arrays incur higher latencies whereas smaller sub-arrays incur lower latencies due to the impact of wire length on delay. For example, the blue curve in Figure 5 plots latency versus sub-array size (square-root of the number of ReRAM cells) for existing Crossbar ReRAM implemented in a 28*nm* technology node. Latencies of 200-300*ns* are achievable, which while high, at least begin to approach DRAM latencies. But these latencies are only possible given very small ReRAM sub-arrays. As discussed in Section 2, larger sub-arrays (corresponding to the right-hand side of Figure 5) are needed for area efficiency. Unfortunately, for larger sub-arrays, latencies are in the 1-2μs range, which are too high for CPUs. Worse yet, the latencies reported in Figure 5 are for reads. Write latencies are another 2x higher.

An important challenge in realizing monolithic computers is to re-design existing ReRAM cells so that they are better suited to CPUs. The design space for ReRAM memory cells is complex, involving numerous device parameters. Up until now, Crossbar has made tradeoffs within this design space that are inappropriate for CPUs because the design target has been storage. One example of this is current Crossbar memories exhibit a retention of *10 years*, which is over-kill for CPU main memory. By making different design tradeoffs, more CPU-friendly characteristics can be achieved. For example, Crossbar estimates that a re-design of their ReRAM cells can result in latencies between 200-700*ns*, even for the largest sub-arrays in Figure 5.

In addition to latency, another drawback of using existing ReRAM for CPU main memory is its low endurance. Current ReRAM memory cells can sustain 10^6 writes. While this is quite good for non-volatile memory, it is not good enough for CPU main memory. Again, as with latency, the endurance of ReRAM is currently tuned for storage devices. Much higher endurance could be achieved if ReRAM were to be redesigned with CPU main memory in mind. As mentioned above, current Crossbar ReRAM exhibits an extremely high retention. This can be traded off for better endurance–for example, by reducing how strongly bits are written into the ReRAM cells.

Overall, developing ReRAM device technology so that it is better matched to CPU requirements is crucial for realizing monolithic computers. The goal is to create a low-retention ReRAM cell that exhibits both high speed and high endurance.

4 ARCHITECTURE CHALLENGE

At the architecture level, a key challenge is exposing the massive memory-level parallelism that will be available in the on-die ReRAM. We anticipate that on a large CPU die, there could be 10s of thousands of ReRAM sub-arrays integrated in the die, each providing an independent memory access point. What's more, each sub-array may contain up to 8 vertically stacked ReRAM layers (Figure 2), providing additional memory access parallelism. (Not all 8 layers will be independent since adjacent layers share wires, but every other layer could be accessed independently). All told, there could be over 100K independent memory access points in an on-die ReRAM memory system.

The actual parallelism available to the CPU, though, will depend on how many sub-arrays are activated per memory transaction. Each ReRAM sub-array provides very little data per access, so multiple sub-arrays–*i.e.*, a *ReRAM bank*–must be activated together to

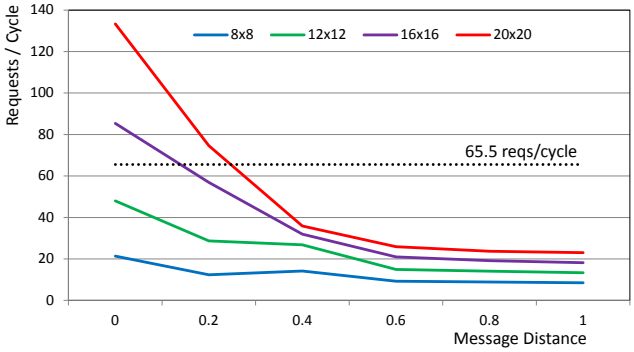

Figure 7: NOC throughput as a function of message distance for different mesh network sizes.

increase the fetch granularity. Figure 6 shows why. When activating a sub-array, an entire row or "page" of memory cells is selected, but only the cells at crosspoints of interest are accessed. Due to limitations in the word-line current, the number of crosspoints that can be active in the same page is quite small, *e.g.* 4 – 8 bits. This means fetching a 64-byte cache block requires activating at least 64 ReRAM pages, requiring banks of 16 sub-arrays each (assuming 4 ReRAM pages can be accessed in parallel per 8-stack sub-array). Banks with a smaller number of sub-arrays are possible if the memory system supports fine-grained accesses. For example, fetching 8-byte double words would only require activating 8 ReRAM pages, allowing banks to contain 2 sub-arrays each. Depending on the bank size, the memory system could support from thousands to small 10s of thousands of independent memory requests across the 100K+ sub-arrays that we anticipate having in the CPU die.

Supporting all of this memory parallelism will place significant stress on the network-on-chip and memory controllers in our tiled architecture that was discussed in Section 1.1. Moreover, being able to *drive* all of this memory parallelism will require CPU cores that can source a tremendous number of simultaneous memory requests. In this section, we discuss the architecture-level challenges we expect in the NOC, MC, and cores of a monolithic computer.

4.1 Network-On-Chip

We begin by looking at challenges in the NOC design for a monolithic computer. Most existing tiled CPUs employ simple NOC topologies, such as the 2D mesh. Unfortunately, we find that simple 2D meshes will have trouble sustaining the levels of memory traffic that may be possible in monolithic computers.

To quantify, we built a 2D-mesh NOC and ReRAM simulator. We configured the simulator to model a large number of ReRAM banks, 32K, grouped into tiles that each contain a single mesh router and MC, just like in Figure 1. At each NOC router, we injected memory requests as fast as possible, thus driving the NOC to its maximum throughput. (We only simulated the request messages, omitting the replies from the ReRAM for these experiments). Each injected memory request is destined to a random ReRAM bank on a random tile some maximum distance away from the originating tile. We varied the number of tiles, thus changing the number of routers, and hence, the messaging capacity of the NOC.[2] We also varied the maximum distance that messages can travel, thus changing the physical locality of the memory requests. Figure 7 plots the throughput achieved in memory requests per cycle as a function of message distance ("0" means all messages only go to the local tile, while "1" means messages can travel to any tile on the NOC). Results are plotted for four different NOC sizes: 64, 144, 256, and 400 tiles.

Figure 7 shows throughput is high as long as there is good locality (*i.e.*, distance = 0). But as memory requests are allowed to travel farther and farther away, throughput drops precipitously. This is due to the inherently low messaging capacity of 2D meshes, resulting in network contention. For all-to-all communication (distance = 1.0), contention becomes severe, and the throughput drops as much as 6x below its peak.

An important question is: what throughput does the NOC need to support for a monolithic computer? While actual throughput requirements will be application dependent, one possible benchmark is the messaging rate needed to keep all of the banks in the on-die memory system busy. For example, given 32K banks, and assuming each bank incurs a 500*ns* access latency, the theoretical throughput that would saturate all of the banks is 32K ÷ 500*ns* = 65.5 memory requests per cycle (at a 1 GHz clock rate). This is an extremely high throughput. As Figure 7 shows, even for modest levels of contention (*i.e.*, at a message distance of just 0.3), *none* of the 2D meshes can achieve a 65.5 requests per cycle throughput (dotted line in Figure 7). Even for large 20x20 NOCs, the 2D mesh is unable to support the memory traffic needed to keep all of the on-die ReRAM banks busy.

A major challenge is to design a NOC that can support the high memory request rates of monolithic computers. Given the limitations of 2D meshes shown in Figure 7, alternative network topologies, possibly with higher dimensionality, will likely be needed. Along with dimensionality, it may be helpful to scale the number of network nodes as well. This may result in designs for which there are more routers than cores (as opposed to the single-router per core baseline in Figure 1). Designing such high-throughput NOCs must be done with area and power limitations in mind. One of the great challenges of monolithic computers is that the entire computer system must fit on a single die. Hence, area and power of the NOC design will be crucial.

4.2 Memory Controller

As with the NOC, a significant challenge for the MC is how to balance circuit size against need for parallelism. And similar to the NOC, one way to strike the best balance may be to decouple the MCs from the cores (and possibly the NOC too), allowing a different number of MCs compared to cores. At one extreme, a system design can have one MC per ReRAM bank (*e.g.*, 32K); at the other extreme, one could have a single controller capable of queueing up that many requests. The disadvantage of the first scenario is the large amount of circuitry required (the die would be almost entirely memory controllers); the disadvantage of the second is the limited parallelism provided by the arrangement: a

[2]The MCs were configured with 0 latency and infinite bandwidth to remove MC contention effects.

controller can only handle one request at a time, so traffic spikes result in long delays.

We seek a system of controllers that can accomplish several very important tasks that pull the design in opposite directions:

- The number of controllers is minimal, to minimize both the number of access points on the NoC and the amount of controller circuitry required chip-wide
- The number of controllers is optimized to handle the number of simultaneous memory requests originating from all the CPU's cores, including scenarios of all-to-all communication, as is considered in Figure 7

These design constraints pull in opposite directions: having fewer controllers reduces circuitry, but having more controllers increases available parallelism, thus lowering queueing delays and end-to-end latency.

Because the number of controllers may or may not equal the number of cores, one of the primary design considerations will be intertwined with the design of the NOC: should controllers lie "behind" the core, as in traditional distributed shared memory architectures, or should the controllers be first-class endpoints on the network-on-chip? This is a very important consideration, because initial calculations show that the number of banks could exceed the number of cores by 2–3 orders of magnitude, and thus the number of controllers could easily be ten times the number of cores. Arguably, raising the memory subsystems (the controllers and their banks, each representing a subset of the total memory system) to the level of "peer" on the network would provide a seamless design space covering application behaviors ranging from a large number of independent, non-cooperating processes, to large-scale multi-threaded applications with all-to-all communication and sharing patterns. Note that DRAM systems such as Rambus [5] and Hybrid Memory Cube [6] have explored this type of main memory architecture, and Storage Area Networks have explored the same architecture at the file-system level. Similar approaches can be used for organizing the on-die ReRAM of a monolithic computer.

4.3 Cores

To realize the massive memory parallelism that will be available in monolithic computers, not only do we need the NOC and MC techniques discussed in Sections 4.1 and 4.2, but we also need compute cores that can inject memory requests into the memory system at a sufficiently high rate. Modern tiled CPUs employ different techniques in the cores to boost memory parallelism. At a minimum, each CPU core can execute a single thread capable of issuing one memory request at a time. So, assuming a tiled CPU with C cores, we can sustain C-way memory parallelism. Higher memory parallelism is possible if the cores can either issue multiple memory instructions from each thread (superscalar) or switch between hardware-managed threads to issue multiple memory requests in an interleaved fashion (multi-threading). And, SIMD instruction extensions can further boost memory parallelism. SIMD pipelines have recently begun supporting scatter-gather, allowing each sub-word from a single scatter-gather operation to generate a separate memory request (*i.e.*, to an arbitrary memory location). For example, Intel's AVX-512 ISA [7] has incorporated such flexible scatter-gather instructions just in the past year.

In combination, all of these techniques can permit CPUs to sustain a very large amount of memory parallelism. Assuming $C = 64$ cores, 4-way multi-threading, and 8-way SIMD with scatter-gather, a total of 2K memory requests can be in flight simultaneously. While this represents a significant amount of memory parallelism, it is actually not enough to utilize all of the ReRAM banks in a monolithic computer. As mentioned earlier, we anticipate potentially having 10s of thousands of ReRAM banks. To more fully utilize the on-die ReRAM, additional techniques within the cores may be necessary to further boost memory parallelism.

One promising technique is non-blocking memory operations. Normally, when a core misses in its cache and initiates a long-latency main memory access, the thread running on the core stalls. In contrast, non-blocking memory operations [8, 9] would allow the thread to continue executing and reach additional memory operations, thus boosting memory parallelism. Writes are easily made non-blocking by adding a buffer to stage the stored data before it is written to memory. Reads are more challenging to make non-blocking since the memory operation doesn't actually complete until the read data returns from main memory. By adding presence bits to the register file, or a scoreboard, the instructions that are dependent upon a non-blocking load can be identified, allowing a stall to occur when a dependent instruction tries to issue.

To our knowledge, such non-blocking loads and stores have been applied to scalar memory operations only [8, 9]. We will investigate their use in SIMD instructions, and in particular, for scatter-gather operations. Because non-blocking SIMD memory operations are orthogonal to the above techniques, they can be used in concert with those previous techniques. Doing so will further increase the memory parallelism by the number of outstanding non-blocking scatter-gathers allowed. As with the NOC and MC designs, a major consideration will be to achieve this parallelism without incurring exhorbit area and/or power such that the entire CPU and its ReRAM memory system can fit in a single die.

5 CONCLUSION

This paper considers the notion of a *monolithic computer*, a future computer architecture in which a CPU and a high-capacity main memory system are integrated in a single die. We believe such computer chips will become possible in the near future due to emerging ReRAM technology that can be integrated over CPU logic within top-level metal layers of a VLSI process.

We identify several challenges that should be addressed in order to realize such new architectures. First, the ReRAM's access circuits must be merged with the CPU's logic. We presented a preliminary physical design study using a simple VSCALE core from the Berkeley RISC-V architecture. In addition to the area overhead of the access circuits (roughly 25%), we find the layout of the CPU core increases by 11.3%. In the future, we hope to perform other similar physical design studies, but with larger and more realistic CPUs.

Second, while the basic ReRAM memory technology exists today at Crossbar, the existing technology is targeted for storage devices. An important challenge is to redesign Crossbar's ReRAM technology so that it is more suitable for use in CPU main memory. In particular, trade offs should be made to sacrifice some retention in order to improve access latency and endurance.

Finally, the CPU architecture should be designed to effectively exploit the massive memory-level parallelism that will potentially be available in the on-die ReRAM. We propose to use a tiled CPU as a starting point, but to integrate the memory controllers into the compute tiles. A crucial part of the design is the NOC. Unfortunately, the most common NOC topology, a 2D mesh, may incur significant contention. It may be necessary to explore higher-dimensional networks that can support greater messaging capacity. In addition to the NOC, the memory controller design is crucial as well. An important issue is the number of controllers that should be provided relative to the number of cores. And, the CPU cores themselves should be capable of sustaining a large memory request rate. Besides multi-threading and wide SIMD scatter-gather memory operations, we believe non-blocking SIMD memory operations can help achieve the desired levels of memory parallelism.

6 ACKNOWLEDGEMENT

This work was supported by the Department of Defense under Contract FA8075-14-D-0002-0007, TAT 15-1158.

REFERENCES

[1] G. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," *International Symposium on Computer Architecture*, 2008.
[2] Intel, "Intel Xeon Phi Product Family, http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html," no. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html, 2014.
[3] A. Agarwal, L. Bao, J. Brown, B. Edwards, M. Mattina, C.-C. Miao, C. Ramey, and D. Wentzlaff, "Tile Processor: Embedded Multicore for Networking and Multimedia," in *Proceedings of the 19th Symposium on High Performance Chips*, (Starford, CA, USA), 2007.
[4] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA," UCB/EECS 2011-62, University of California, Berkeley, May 2011.
[5] "Rambus, http://www.rambus.com." 2014.
[6] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube New DRAM Architecture Increases Density and Performance," in *Proceedings of the 2012 Symposium on VLSI Technology*, (Honolulu, HI), June 2012.
[7] Intel, "AVX 512 Instruction Extensions, http://software.intel.com/en-us/blogs/2013/avx-512-instructions." 2017.
[8] K. I. Farkas, N. P. Jouppi, and P. Chow, "How Useful are Non-blocking Loads, Stream Buffers, and Speculative Execution in Multiple Issue Processors?," WRL Research Report 94/8, Western Research Laboratory, December 1994.
[9] A. Rogers and K. Li, "Software Support for Speculative Loads," in *ASPLOS-V*, pp. 38–50, ACM, October 1992.