Toward a Memory-Centric, Stacked Architecture for Extreme-Scale, Data-Intensive Computing

John D. Leidel
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409
Email: john.leidel@ttu.edu

Xi Wang
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409
Email: xi.wang@ttu.edu

Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409
Email: yong.chen@ttu.edu

Abstract—One of the primary concerns of performing efficient data-intensive computing at scale is the inherent ability to exploit memory bandwidth on a local and global scale. The traditional computer architecture inherently decouples the processing interconnect from the memory interconnect, thus preventing efficient, parallel utilization of both at scale. Further, the orthogonal nature of these board-level and system-level interconnects force users to build messaging libraries to bridge the gap between local and global memory access. The result is a system architecture that fails to efficiently expose the necessary locality of non-deterministic data patterns for algorithmic exploitation at scale.

In this position paper, we present a theoretical architecture based upon processing near memory using a 3-dimensional stacked device component as the central processing and networking component. We utilize the second generation, Hybrid Memory Cube device architecture, packet specification and interconnect as the basis for our architecture in order to dramatically increase the compute-to-bandwidth relationship by augmenting the logic layer of the HMC devices with actual processing elements. We also utilize the same core component to serve as the interconnection mechanism between multiple processing elements such that memory and communication become a homogeneous function of the platform. Finally, we explore two potential algorithmic exploitation methodologies that could be used to efficiently implement non-deterministic data analytics problems at scale.

1. Introduction

Given the inherent memory bandwidth and latency constraints of large-scale, data-intensive applications and algorithms, modern computational infrastructures fail to provide an efficient platform for execution. Traditional large-scale computational systems fail to provide the fundamental expression of data locality at the architectural level necessary to build minimal runtime libraries and efficient programming models. Further, the communication overheads associated with orthogonal memory and I/O infrastructures further exacerbates the inefficient execution of data intensive algorithms when a disproportionate number of data blocks are

not found in the physically attached main memory. The end result being a system architecture that operates drastically below its theoretical peak processing and I/O performance.

In this work, we present a theoretical system architecture that couples space and energy efficient RISC-V cores to a near-memory logic layer based solely on the Hybrid Memory Cube (HMC) device specification. We analyze the potential use of the HMC link layer as the basis for a scalable system interconnect that utilizes addressing as the core routing table and present sample power and performance numbers for a scalable system based upon existing measurements. The end result being a tightly coupled system and memory infrastructure that utilizes the same I/O pathways for memory operations and communication operations with a simple central processing design that lends itself well to compiler and runtime optimization. As such, this combined near-memory compute and communication infrastructure has the fundamental structure to efficiently execute data-intensive algorithms and applications in both traditional compute-communication and data flow programming models.

The remainder of this work is organized as follows. Section 2 presents the proposed system architecture devices, device organization, potential performance and the inherent risks associated with the proposed architecture. Section 3 describes two potential algorithmic methodologies by which we may exploit the proposed system architecture. Finally, Section 4 summarizes our efforts.

2. Memory Centric System Architecture

2.1. Memory Architecture

The memory architecture is constructed using 8Link-8GB Hybrid Memory Cube devices [3]. We build two custom logic layers to represent compute nodes (1a) and routing nodes (1b), respectively as shown in Figure 1. The logic layer is grown from the original $34mm^2$ as documented in the HMC specification [1], [3] to $68mm^2$ in order to accommodate additional processing or routing functions in the respective device configuration. While this doubling of die

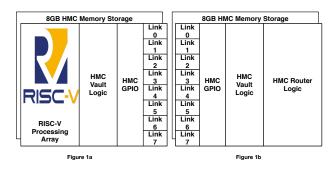


Figure 1. HMC Device Architectures

area may seem egregious, this falls well within the capability of the state of the art in CMOS fabrication technology.

We modify the core routing approach utilized by the HMC device specification. Several proposals have been published proposing a similar utilization of HMC links as a system interconnect [4], [5]. However, we extend these initial proposals to harness the HMC protocol as the basis for a scalable, partitioned global address space methodology. In this manner, the core address format is directly coupled to the decomposition of data blocks across the parallel set of compute nodes. This platform optimization is key to our algorithmic exploitation as it removes the requirement to host any ancillary static or dynamic routing tables and it exploits the memory-centric nature of these analytics workloads as a first order design principle.

2.2. Processing Architecture

In order to provide efficient near-memory computing capabilities, we utilize a variant of the RISC-V ISA with an included HWACHA SIMD unit [2]. The RISC-V ISA is a BSD licensed infrastructure that provides excellent extensibility without sacrificing core performance and throughput. The proposed unit will be packaged as a set of 28 cores, each with local SIMD compute and small instruction caches. The cores are documented to offer 34 GFlops of peak computing throughput using roughly $1.2mm^2$ of die area on a 28nm process. We utilize the previously published throughput and performance values as quoted by other RISC-V developers [2]. These values may differ depending upon the logic density and clock frequency achieved in an actual device tape out.

Each of the individual cores is configured with an associated instruction cache and no data cache unit, as shown in Figure 2. We deliberately forego utilizing local data caches in order to avoid the associated control mechanisms required to maintain coherency between all the cores within a given logic die. Further, incoming external requests from adjacent dies are directly served by the vault control units, outside the prevue of the in-situ processing resources. As a result, maintaining data cache coherency would be cost prohibitive in terms of logic layer area and external link bandwidth. As such, all data member requests are fulfilled by equivalent, raw memory requests formulated as HMC packets.

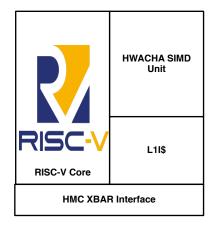


Figure 2. Core Architecture

As a result, any atomicity required to exist between cores must be maintained via the in-situ HMC atomic memory request packets or other custom memory cube atomicity mechanisms [18].

The initial node architecture consists of 28 individual cores that lack any inherent on-chip connectivity such as multi-level caches. While this may appear naive, we focus on utilizing the memory-centric nature of the device to perform any on-chip communication tasks. As we begin to further explore the physical design and layout of the compute device logic layer, we may find that we have sufficient die area to interconnect multiple cores into a more complex system-on-chip (SoC) with mechanisms such as shared scratchpad memories. At which point, we would make use of a scalable on-chip SoC fabric such as the OpenSoC Fabric [9] in order to interconnect the individual components in the logic layer.

2.3. Full System Architecture

As a sample implementation, we construct a full system architecture using our aforementioned approach using a three-dimensional network topology. As the basis for our system architecture, we define an individual node building block as four compute devices and two router devices as shown in Figure 3. The HMC physical links are broken into four categories. First, links 0 and 1 for each compute device are connected to routers 0 and 1, respectively. In this manner, we construct a redundant, or dual-rail, network. This permits us to alleviate communication congestion and balance the incoming and outgoing routing queues. Next, we connect two links from each compute device to all adjacent compute devices. In this manner, 25% of the link bandwidth is dedicated to adjacent compute devices. This provides us the ability to exploit compute-level and partition-level data decomposition and data locality within our target algorithms. Three of the router links are then connected to the X, Y and Z torus axes. Finally, Link 7 from each router is optionally connected to a periphery I/O device. This provides the opportunity to connect external

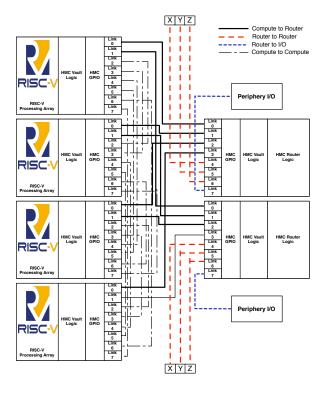


Figure 3. Stacked Memory Node Architecture

TABLE 1. SYSTEM ARCHITECTURE

Processing Architecture		
Component	Value	
RISC-V Rocket with HWACHA Area	$1.2mm^{2}$	
RISC-V Rocket with HWACHA Freq	951Mhz	
RISC-V Rocket with HWACHA Perf	34 GFlops	
RISC-V Cores per HMC	28	
Peak RISC-V Performance	952 GFlops	
Memory Architecture		
Component	Value	
HMC Configuration	8Link-8GB Device (15Gbps)	
HMC Area (Compute and Router)	$68mm^2$	
System Architecture		
Configuration	Value	
HMC Power	136watt	
5x5x5 Torus Performance	476TFlops	
5x5x5 Torus Memory	1TB	
5x5x5 Torus Power	102KW	
10x10x10 Torus Performance	3.8PFlop	
10x10x10 Torus Memory	8TB	
10x10x10 Torus Power	816KW	
50x50x50 Torus Performance	476PFlop	
50x50x50 Torus Memory	1PB	
50x50x50 Torus Power	103MW	

devices such as infrastructure network (Ethernet) devices, mass storage devices or application-specific sensor devices directly into the memory and compute hierarchy.

Table I summarizes the system architectural details as well as provides some notable scaling parameters for 5^3 , 10^3 and 50^3 cubic, 3D torus topologies. While our largest

example likely exceeds the theoretical power thresholds for an exascale system designed for dense linear algebra, it is an excellent example of how pseudo-commodity components can be adapted for the future of data-intensive scalable computing. Further, the potential to efficient execute scalable data-intensive algorithms on the proposed platform likely warrant the construction of a smaller total system as compared to traditional commodity methods.

2.4. Performance Evaluation

Despite the lackluster peak double precision floating point performance as elicited in Section 2.3 and Table 1, our theoretical system architecture has rather interesting performance characteristics. First, in order to understand these inherent idiosyncracies, we must first analyze the local request structure.

The HMC packet request structure is formulated to provide high bandwidth memory read and write commands with little control overhead for large, coalesced requests [19]. This provides traditional processors the ability to fetch entire cache lines (or multiples thereof) with single HMC packet requests. As of this writing, the smallest read or write request is performed on 16 bytes of data. The largest read or write request is performed on 256 bytes of data. Given the near-memory nature of our proposed architecture, we do not pay the control overhead necessary to operate the SERDES links for local requests. However, this control overhead is still incurred for any requests that go off chip. Further, for scalar RISC-V load or store requests, the largest data element is 8-bytes (RV64IMAFD). As such, our maximum load/store efficiency for local, scalar requests is 50%. Each of the attached HWACHA SIMD units has the ability to manipulate 128-bit (16-byte) registers with single, stride-1 load and store requests. In this case, the local memory requests are well suited to the minimum 16-byte request granularity enforced by the HMC packet specification. However, for remote memory requests across the HMC link network, we are yet again penalized for small packets with a large degree of control overhead. Further investigation is required to fully characterize the local versus remote bandwidth ratio and any potential hardware or software-driven solutions.

In addition to the local performance characteristics, we can also characterize the theoretical peak system architecture characteristics that may be pertinent to certain data intensive applications at scale. First, the SERDES-based HMC interconnect provides an interesting platform for a high bandwidth interconnection network at scale. Assuming the highest performance SERDES links supported by the latest generation of HMC device [3], the 15Gbps would provide up to 40 GB/s per link of bi-directional bandwidth. Also note our node interconnect topology as shown in Figure 3. Each of the four sockets are connected to the HMC routers with a single link, totaling eight links across two routers. In this manner, we provide up to 320 GB/s of bandwidth between the HMC device sockets within a node.

Also note that the bandwidth between nodes, via the routed three dimensional torus, is 25% less than the band-

TABLE 2. SYSTEM ARCHITECTURE PERFORMANCE

System Architecture	
Configuration	Total Network Bandwidth
5x5x5 Torus	30 TB/s
10x10x10 Torus	240 TB/s
50x50x50 Torus	30 PB/s
Configuration	Bisection Bandwidth
5x5x5 Torus	4 TB/s
10x10x10 Torus	16 TB/s
50x50x50 Torus	4 PB/s
Configuration	Peak Bytes/Flop
5x5x5 Torus	0.063
10x10x10 Torus	0.063
50x50x50 Torus	0.063

width within node. We dedicate a single link per router device to periphery devices. However, this ratio of inner node to intra node bandwidth is still significantly better than current large scale systems. We utilize our routed network bandwidth as the basis for a total network injection bandwidth calculations for various system configurations in Table 2. The 5x5x5 or small configuration scales to 30TB/s of network injection bandwidth, followed by the 10x10x10 with 240GB/s and the 50x50x50 configuration with 30PB/s. Given the reliance of certain applications such as N-body physics, graph analytics and unstructured solvers on scalable network bandwidth, we also report the equivalent bisection bandwidth of our theoretical system architecture. The largest reported configuration would report a peak bisection bandwidth of over 2PB/s while the smallest 5x5x5 configuration, would still maintain 200GB/s of bisection bandwidth. This. will minimal protocol translation and unnecessary buffering.

Finally, examine the total system byte per flop ratio. We make a conservative estimation of this metric by comparing our total network injection bandwidth to our peak double precision floating point performance metric for each respective configuration. We utilize the global network injection rate as a deliberately pessimistic view of the architecture in order to make no admission on inherent or algorithmic data locality. This permits us to account for any global, or *all-to-all*, communication scenarios within a potential algorithm or application. Each of the configurations would report a theoretical byte per flop ratio of 0.063. This ratio would will improve for real-world applications by a rate analogous to the local versus global memory request ratio. EG, the byte per flop ratio is expected to improve based upon the existence of any local memory requests.

2.5. Risks

The risks associated with platform can be categorized into two general areas: manufacturing risks and system software. The most obvious risk associated with this theoretical system architecture is the inherent use of through-siliconvia, or *TSV*, stacked devices. At the time this work was written, few manufacturing organizations had the ability to produce a large number of stacked devices with fabrication yields that would lend sufficient cost benefit to manufacture.

This inability to easily contract these devices for large-scale production may initially prohibit small-scale prototyping efforts barring sufficient funding.

In addition to the inherent manufacturing risks, our proposed system architecture also holds risks in terms of the overall system software. In a near-memory architecture such as this, the traditional role of the operating system to provide memory protection, virtualization and allocation is quite orthogonal given the locality of the memory with respect to the core computing units. In order to construct a full scale prototype or full-scale system, we would be required to integrate a certain degree of management processing capabilities in order to service tasks such as the operating system, memory management and sideband debugging facilities. As such, we must perform further research in order to develop a full understanding of system software in an architecture such as this.

3. Algorithm Exploitation

3.1. Compute-Communication Execution

The first programming model that we will explore on the aforementioned platform is based upon a traditional compute-communication model. Each of the RISC-V cores embedded in the logic layer of the HMC device will execute a stream of instructions that built as a part of a parallel algorithm or application. Cores may communicate with one another via message queues reserved from within the in-situ memory storage.

Given the traditional compute-communication model of this programming model, the first language construct that we will explore is the use of OpenMP [12]. OpenMP provides a well supported, portable approach to shared memory parallelism. Typical constructs such as thread atomicity, synchronization and physical memory locality are all constructs either currently addressed in the OpenMP specification or will soon be addressed in forthcoming versions.

The second traditional programming model that we will explore is the Unified Parallel C, or UPC, programming model [13]. The underlying addressing model found in the architecture is well suited to the natural remote and global memory partitioning found in UPC. As such, mapping a UPC runtime implementation to this platform will require minimal machine-level support.

In addition to the aforementioned compute-communication programming models, we may also attempt to adapt programming models such as the MPI standard or a bulk synchronous parallelism model such as MapReduce. Both may provide interesting platforms by which to implement data-intensive algorithms or applications.

3.2. Data Flow Execution

The second style of algorithmic exploitation that we will explore on our platform is akin to the traditional data flow

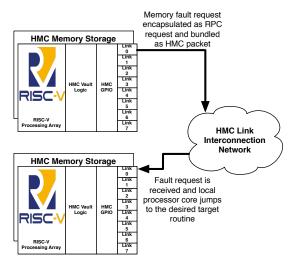


Figure 4. Data Flow Execution

programming model [14]. Data flow programming models have been found to be efficient for executing algorithms in graph/network theory often utilized in data-intensive computing. The inherent data centric nature of this programming model provides an advantageous route by which to exploit the partitioned global addressing methods described above.

However, we utilize a mechanism similar to the interprocess communication style of remote procedure calls (RPC) to drive our data flow model. Any one of a given RISC-V processor cores is permitted to execute as long as the target memory blocks utilized for a given permutation are stored in the physically attached memory vaults. In this manner, the core is permitted to execute using a traditional load-store model. However, whenever a permutation is requested for a remote memory block, rather than initiating a memory fetch operation from a potentially distant memory storage device, the local core initiates an RPC call to a remote core that is physically co-located with the requested memory block. We elicit a simple example in Figure 4.

This concept is often referred to as *moving the compute* to the memory in processing near memory theory. The elegance behind this model is the ability to easily perform complex graph traversals and connectivity measurements by simply iterating across a small number of core, recursive routines. However, the difficulty in providing such a model is promoting the use of simple, recursive entry points within the application such that the cores may efficiently distribute the compute tasks effectively. Further, there is an inherent downside to this style of computing as the graph or graphs may be heavily weighted to one or more memory nodes or have recursive loops in the graphs that induce deadlocks in the program execution. These items will be addressed in future research activities.

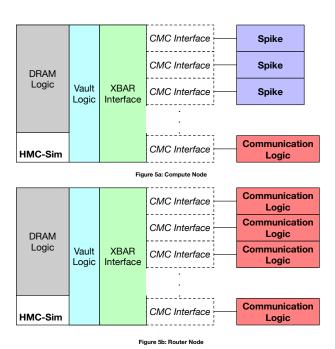


Figure 5. Simulation Architecture

4. Conclusion and Ongoing Work

In this position paper, we propose a system corresponding of a heterogeneous set of stacked memory devices based upon the existing second generation of the Hybrid Memory Cube specification. The goal of this work is to provide a theoretical system architecture loosely based upon existing technologies in order to provide a potentially significant increase in computational capability for data-intensive applications executing at scale. In addition to the core architectural methodologies, we provide two potential paths by which to exploit the system architecture for efficient algorithmic exploration and exploitation at scale.

Despite the inherently theoretical nature of this work, we have performed sufficient investigation into the feasibility of the aforementioned platform. As a result, the next step in this research is to prepare a simulation infrastructure that permits us to begin experimenting with sample algorithms and addressing the system software risks described in Section 2.5. This simulation infrastructure will make use of one or more instances of the RISC-V Spike simulator [10] coupled to the HMC-Sim [15] [16] [17] infrastructure. The HMC-Sim simulation infrastructure includes a unique capability to prototype future HMC commands and operations. This interface, herein referred to as the custom memory cube or CMC logic, provides a shared library interface into the HMC simulator by which users can build custom logic operations that are processed as normal HMC operations [15]. In this manner, the CMC operations may inject requests into the crossbar and/or vault queues in order to provide user-defined functionality.

We leverage this interface in order to construct the afore-

mentioned simulation infrastructure. First, we will construct a small shim interface such that we may couple the standard Spike simulation infrastructure to the HMC-Sim layer. This will provide the ability to couple a single instance of Spike representing a single core to the HMC-Sim logic. Figure 5a depicts the basic compute device interface. In addition to the compute logic simulation, we will also construct one or more CMC instances that implement any necessary communication commands required to send messages across the coupled HMC link infrastructure. This logic is depicted in Figure 5b. These commands will serve as the basis for any higher-level programming model messaging. The end result of our simulation infrastructure prototyping will be an integrated platform that supports algorithmic experimentation, compiler/tool chain development and system software development.

Acknowledgments

This work is supported in part by the National Science Foundation under grant CNS-1338078.

References

- S. Graham, "HMC Overview: A Revolutionary Approach to System Memory", Memcon, 2013.
- [2] Y. Lee, B. Zimmer, A. Waterman, et.al, "Raven: A 28nm RISC-V Vector Processor with Integrated Switched-Capacitor DC-DC Converters and Adaptive Clocking", *HotChips* 27, 2015.
- [3] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 2.1". Technical Report, http://www.hybridmemorycube.org/files/SiteDownloads/. December, 2015
- [4] G. Kim, J. Kim, J. H. Ahn, and J. Kim. "Memory-centric system interconnect design with hybrid memory cubes". In Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, pages 145155, Sept 2013.
- [5] E. Azarkhish, D. Rossi, I. Loi, and L. Benini. "High performance axi-4.0 based interconnect for extensible smart memory cubes". In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 15, pages 13171322, San Jose, CA, USA, 2015. EDA Consortium.
- [6] M. Gokhale. "Near Data Processing are we there yet," Keynote address. WoNDP: 2nd Workshop on Near-Data Processing, 47th IEEE/ACM International Symposium on Microarchitecture (MICRO-47), December 2014.
- [7] R. Nair. "Active Memory Cube: A Processing-in-Memory Approach to Power Efficiency in Exascale Systems". Keynote address. WoNDP: 2nd Workshop on Near-Data Processing, 47th IEEE/ACM International Symposium on Microarchitecture (MICRO-47), December 2014.
- [8] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T-M. Low, L. Pileggi, J. Hoe, F. Franchetti. "3D-Stacked Memory-Side Acceleration: Accelerator and System Design". 2nd Workshop on Near Data Processing (WONDP) in conjunction with the 47th International Symposium on Microarchitecture (MICRO-47), 2014.
- [9] F. Fatollahi-Fard, D. Donofrio, G. Michelogiannakis, J. Shalf. "OpenSoC Fabric: On-chip network generator". 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2016.
- [10] A. Waterman. "Design of the RISC-V Instruction Set Architecture". Technical Report No.UCB/EECS-2016-1, EECS Department, University of California, Berkeley, January 2016.

- [11] E. Azarkhish, D. Rossi, I. Loi, L. Benini. "A Logic-base Interconnect for Supporting Near Memory Computation in the Hybrid Memory Cube". 2nd Workshop on Near Data Processing (WONDP) in conjunction with the 47th International Symposium on Microarchitecture (MICRO-47), 2014.
- [12] "OpenMP 4.5 Specification". OpenMP Architecture Review Board. November 2015.
- [13] "UPC Language and Library Specifications, v1.3". Lawrence Berkeley National Lab Tech Report LBNL-6623E, Nov 2013.
- [14] W.M. Johnston, J.R.P. Hanna, and R.J. Millar. "Advances in dataflow programming languages." ACM Comput. Surv. 36, March 2004, 1-34.
- [15] J. Leidel, Y. Chen. "HMC-Sim 2.0: A Simulation Platform for Exploring Custom Memory Cube Operations". 6th International Workshop on Accelerators and Hybrid Exascale Systems (AsHES) in conjunction with the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 23 May 2016, Chicago, IL.
- [16] J. Leidel, Y. Chen, "HMC-SIM: A Simulation Framework for Hybrid Memory Cube Devices", Journal of Parallel Processing Letters, December 2014.
- [17] J. Leidel, Y. Chen, "HMC-SIM: A Simulation Framework for Hybrid Memory Cube Devices", *Large-Scale Parallel Processing Workshop* in conjunction with the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 19 May, 2014, Phoenix, AZ.
- [18] J. Leidel, Y. Chen, "Exploring Tag-Bit Memory Operations in Hybrid Memory Cubes", Proceedings of the Second International Symposium on Memory Systems, 2016, 153–163.
- [19] X. Wang, J. Leidel, Y. Chen, "Concurrent Dynamic Memory Coalescing on GoblinCore-64 Architecture", Proceedings of the Second International Symposium on Memory Systems, 2016, 177–187.