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

John D. Leidel, Xi Wang, Yong Chen

February 4, 2017

Data-Intensive Scalable Computing Laboratory (DISCL)

TEXAS TECH UNIVERSITY

Workshop on Pioneering Processor Paradigms

Overview

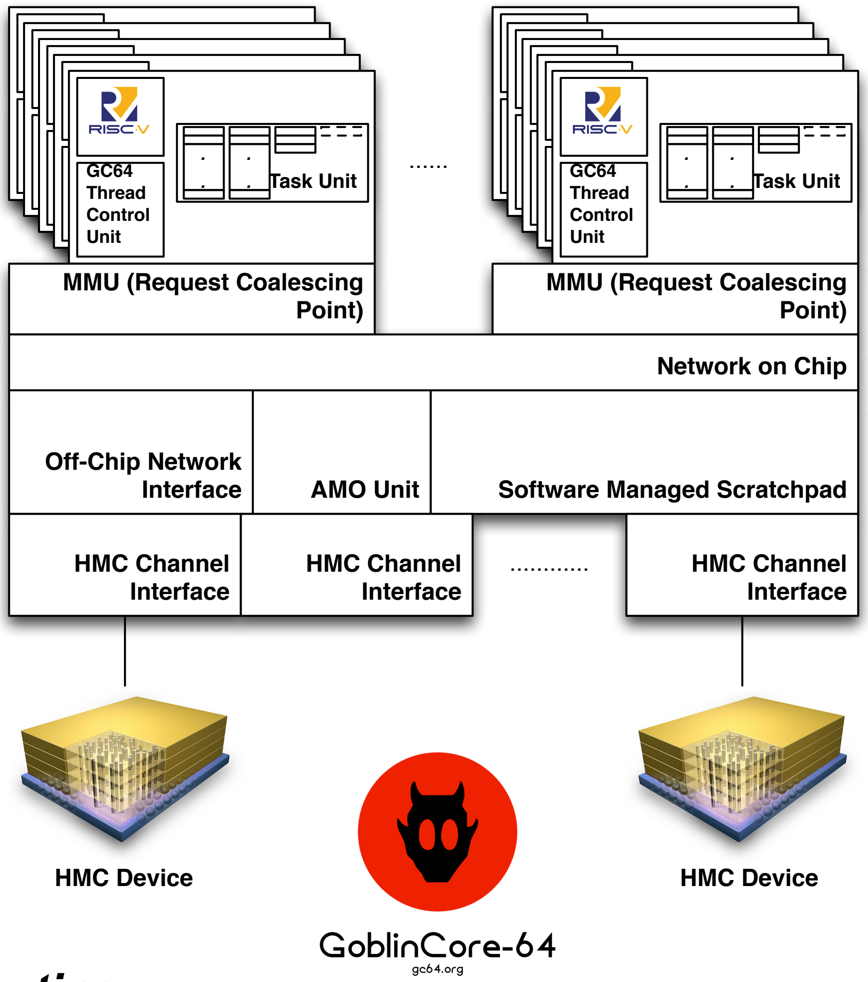
- Background
- Memory-Centric Architecture
- Programming Models
- Risks, Progress and Future Work

Memory-centric Architecture Research

BACKGROUND

History of our HMC Efforts

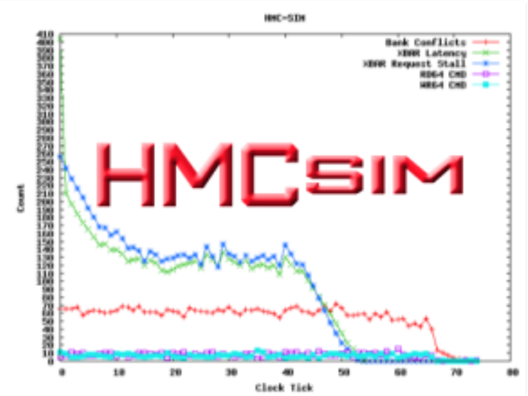
- Our HMC-related research began with the GoblinCore-64 (GC64) project
  - gc64.org
- Purpose-built data intensive high performance computing instrument
- Initial survey of high performance memory technologies pointed toward HMC as an excellent candidate
- GC64 local and global (partitioned) addressing relies upon HMC physical memory specification
- Interconnect is a based upon an extended HMC protocol & phy

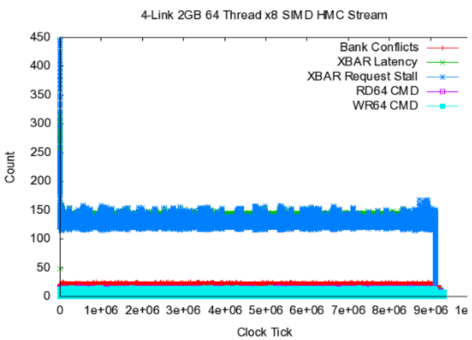


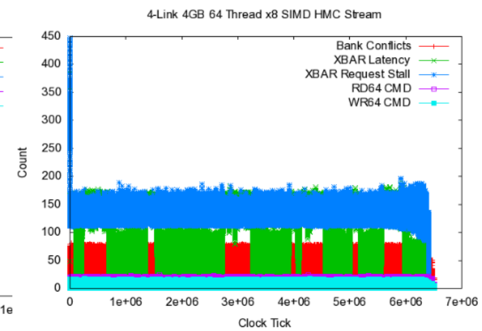
Data Intensive Scalable Computing Laboratory at Texas Tech

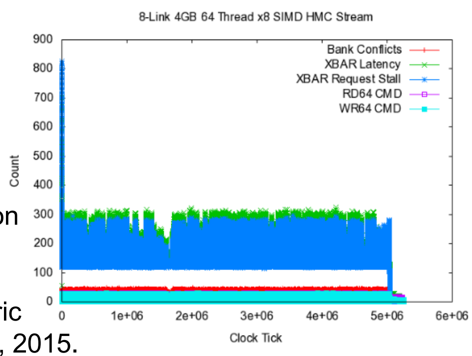
HMC Simulation Effort

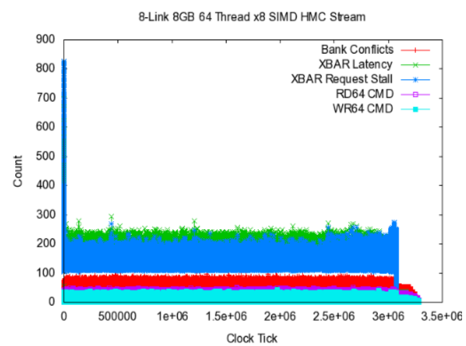
- Significant lack of simulation capabilities for HMC (and other emerging memory specs)
- GC64 is open source and BSD licensed
  - Prevented us from seeking Micron internal simulation capabilities
- HMC-Sim was born!
  - Functional simulation package designed to implement the HMC **<u>specification</u>**
  - Does not consider any one device SKU
  - Packaged as a library written in C
  - Currently integrated into Sandia SST toolkit
    - John D Leidel and Yong Chen. HMC-Sim: A simulation framework for hybrid memory cube devices. *Parallel Processing Letters*, 24(04):1442002, 2014.
    - John D Leidel and Yong Chen. Toward memory centric architecture simulation for data intensive applications, 2015. 2015 Workshop on Modeling & Simulation of Systems and Applications.





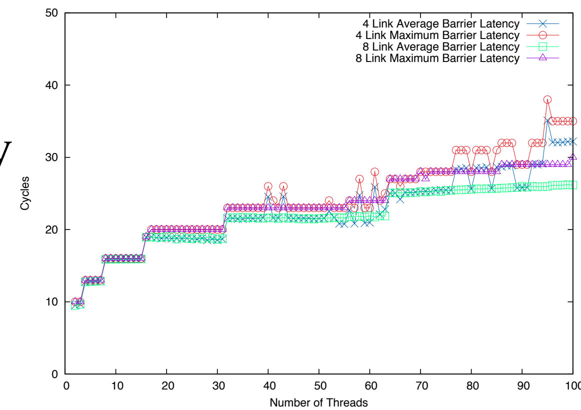


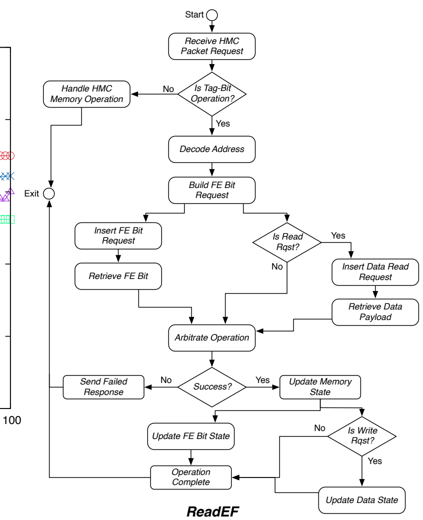


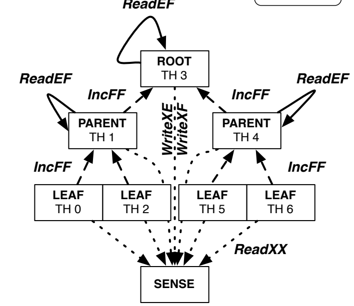


HMC-Sim 2.0: CMC Operations

- Following several requests from users, we integrated the ability to define *Custom Memory Cube* operations in HMC-Sim
- CMC Operations are packaged as shared libraries and loaded by the HMC-Sim infrastructure at runtime
- Operations are handled just as any traditional HMC packetized operation using the defined custom logic
- Permits us to rapidly experiment with future logic operations for HMC (or potentially other memory specs)



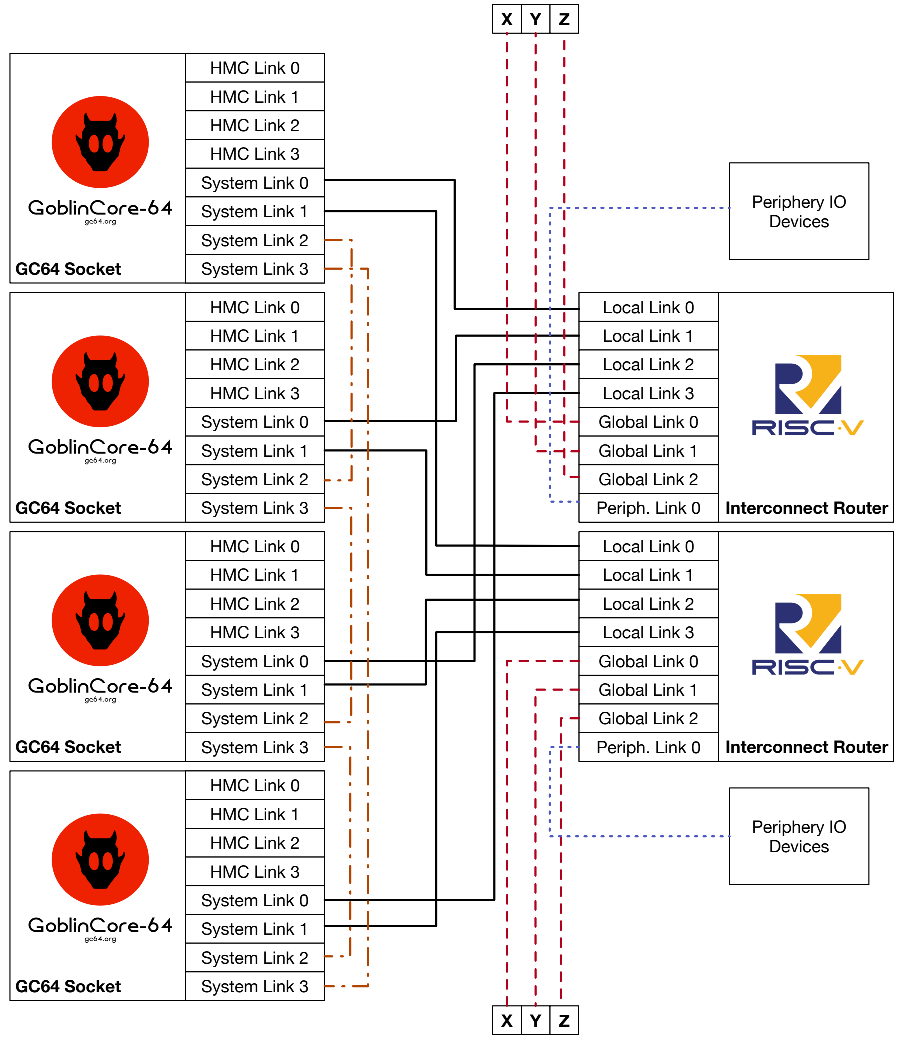




- John D. Leidel and Yong Chen. HMC-Sim-2.0: A simulation platform for exploring custom memory cube operations. In *2016 IEEE International Parallel and Dis- tributed Processing Symposium Workshops (IPDPSW)*, pages 621–630, May 2016.
- JohnLeidelandYongChen.Exploringtag-bitmemoryoperationsinhybridmemory cubes. In *Proceedings of the 2016 International Symposium on Memory Systems*, MEMSYS '16, New York, NY, USA, 2016. ACM.

HMC to PIM (P*n*M)

- Following several efforts to experiment with CMC operations, we began to theorize what else was possible
  - Communication primitives?
  - Additional AMO's?
  - Concurrency features?
- Our ongoing work in the RISC-V community alongside our CMC simulation efforts triggered an interesting thought
  - *Can we utilize our CMC simulation capabilities and our RISC-V experience to build a PIM (PnM)?*
  - *What are the performance, power and programmability ramifications of doing so!?*



John D. Leidel, GoblinCore-64: A Scalable, Open Architecture for Data Intensive High Performance Computing. Doctoral Dissertation. May 2017.

Merging Stacked Memories and Lightweight RISC Cores

MEMORY-CENTRIC ARCHITECTURE

Memory-Centric Architecture Thought Experiment

- Construct a memory-centric, processor near memory architecture using HMC and open source RISC-V cores
  - RISC-V devices have excellent power/performance efficiency and we have existing hardware implementations that are BSD licensed and proven in silicon
  - The HMC device specification isn't perfect, but it has been proven in silicon using Micron's production-scale fab
- Construct the device using reasonable extensions to the existing Gen2 HMC specification
  - The Gen2 spec has a reasonable number of unused opcodes that can be used to extend the existing spec without drastically expanding the scope of the logic layer
  - Avoid performing protocol translations when communicating off a local cube (eg, AXI)
  - Minimize the disparate hardware modules
- Demonstrate the approach using traditional and novel programming models using known simulation methods
  - Using the existing HMC-Sim and RISC-V Spike simulators, demonstrate a reasonable simulation of our environment
  - Utilize existing PGAS and shared memory approaches (UPC, Chapel, OpenMP)
  - Utilize novel dataflow programming models

Memory Architecture

- Consists of 8-Link, 8GB HMC devices constructed in two configurations
- Processing Nodes (1a)
  - Implements processing capabilities in the logic layer of an HMC device
  - HMC DRAM storage utilized for application workload memory/working set
- Router Nodes (1b)
  - Implements a modified HMC packet specification to include our extended memory request operations
  - HMC DRAM storage utilized for routing tables, configuration information, etc
- Logic area is doubled beyond the base HMC device implementation
  - 34 mm^2 to 68 mm^2
  - Additional die area utilized to implement multicore RISC-V SoC on processing nodes
  - Additional die area utilized to implement routing logic on router nodes

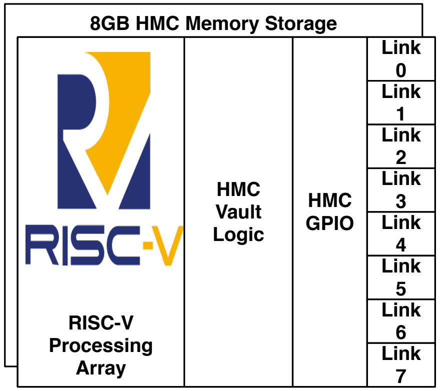


Figure 1a

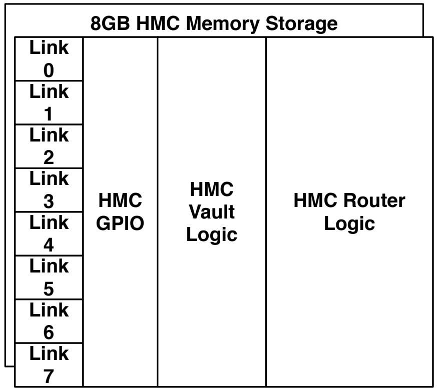
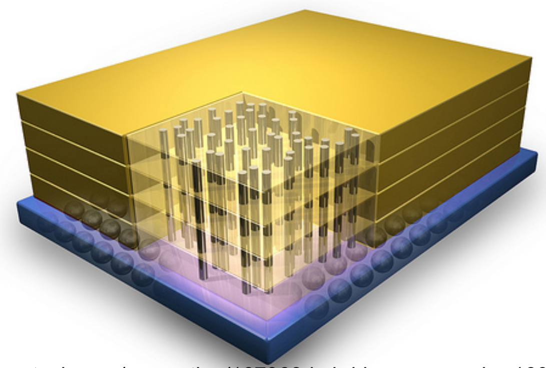


Figure 1b

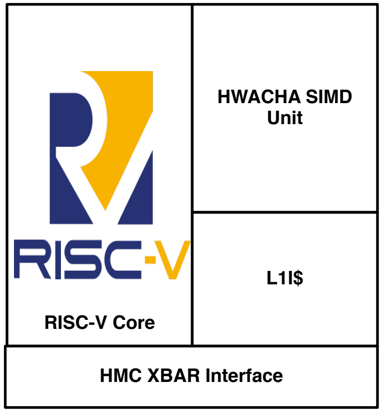


https://www.extremetech.com/computing/167368-hybrid-memory-cube-160gbsec-ram-starts-shipping-is-this-the-technology-that-finally-kills-ddr-ram

DISCL

Processing Architecture

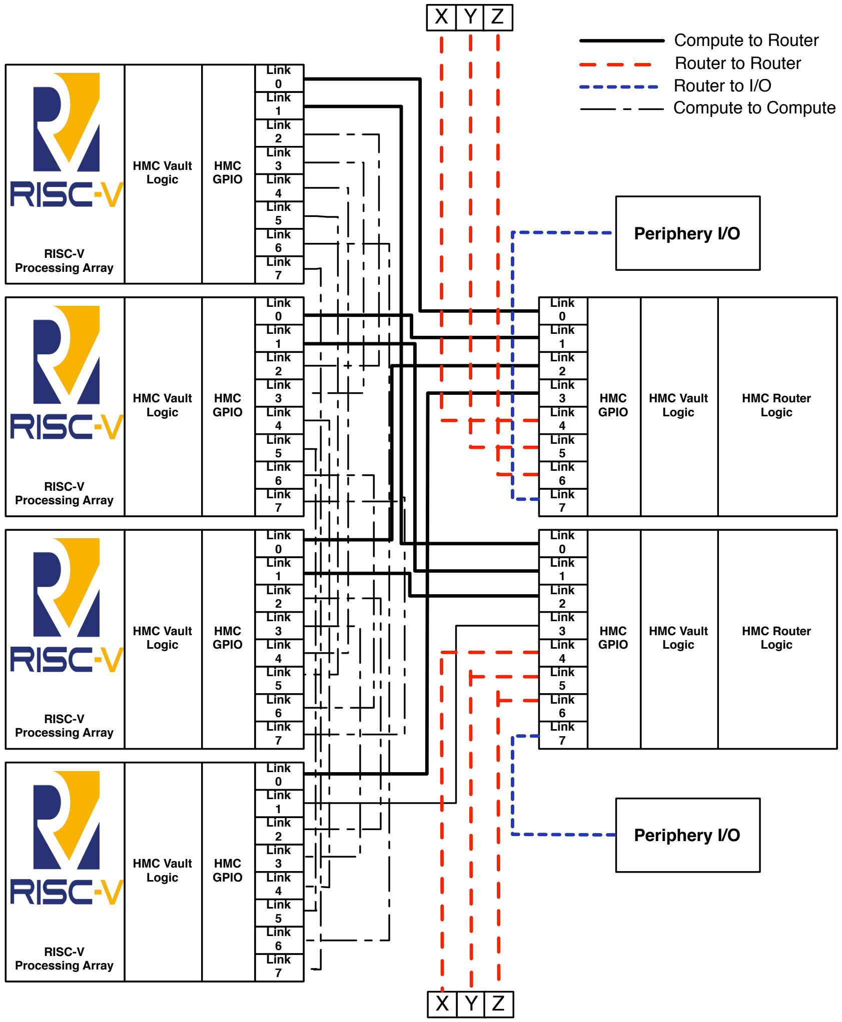
- Processing nodes feature [28] RISC-V cores in the logic layer
  - Cores are configured using the RISC-V IMAFD (G) spec
  - Currently utilize the Rocket 5-stage, in-order pipeline
    - *Simple, but effective!*
  - 34GFlops each at ~1Ghz and 1.2mm^2 (28nm)
- Configured with SIMD extensions to increase processing throughput
  - HWACHA SIMD unit
  - Could be substituted for the forthcoming RISC-V SIMD/vector spec
- Peak performance per core: 34GFlops
- Peak performance per 3D stack: 952GFlops



- Each core contains a local L1 instruction cache, but no data cache
- Why?
  - Logic area is incredibly precious in our device; we don't want to expend area on local cache coherency
  - We don't want to expend communication bandwidth to maintain cache coherency

Interconnect Architecture

- Interconnect is based upon HMC methodology adapted for long-haul connectivity
  - Constructed using [4] processing nodes and [2] routing nodes in a 3D torus
- Connectivity is split into four sets of links
  - 2 links per device connected to routers
  - 6 links per device connected to adjacent 3 nodes
    - *Local connectivity promotes tiered NUMA*
  - Each router has 3 links for [X,Y,Z] torus connectivity to adjacent routers
  - One link per router dedicated to peripheral I/O connectivity



Interconnect Packet Structure

Extended Header (8-bytes)		HMC Packet		Extended Tail (8-bytes)

Full complement of partitioned global address packets using the GC64 extended physical addressing spec

Command	Opcode	Request FLITS	Response FLITS
Dual 8-byte Signed Add Imm	0x70	3	2
Single 16-byte Signed Add Imm	0x71	3	2
Posted dual 8-byte Signed Add Imm	0x72	3	0
Posted single 16-byte Signed Add Imm	0x73	3	0
Dual 8-byte Signed Add Imm And Return	0x74	3	3
Single 16-byte Signed Add Imm and Return	0x75	3	3
8-byte Increment	0x76	2	2
Posted 8-byte Increment	0x77	2	0
16-byte XOR	0x55	3	3
16-byte OR	0x56	3	3
16-byte NOR	0x57	3	3
16-byte AND	0x58	3	3
16-byte NAND	0x59	3	3
8-byte Compare & Swap (GT)	0x5A	3	3
16-byte Compare & Swap (GT)	0x5B	3	3
8-byte Compare & Swap (LT)	0x5C	3	3
16-byte Compare & Swap (LT)	0x5D	3	3
8-byte Compare & Swap (EQ)	0x5E	3	3
16-byte Compare & Swap (ZERO)	0x66	3	3
8-byte Equal	0x67	3	2
16-byte Equal	0x6B	3	2
8-byte bit write	0x6C	3	2
Posted 8-byte Bit Write	0x6D	3	0
8-byte Bit Write with Return	0x6E	3	3
16-byte Swap	0x6F	3	3

Command	Opcode	Request FLITS	Response FLITS
16-byte Write	0x4	3	2
32-byte Write	0x5	4	2
48-byte Write	0x6	5	2
64-byte Write	0x7	6	2
80-byte Write	0x14	7	2
96-byte Write	0x15	8	2
112-byte Write	0x16	9	2
128-byte Write	0x17	10	2
256-byte Write	0x20	18	2
16-byte Posted Write	0x24	3	0
32-byte Posted Write	0x25	4	0
48-byte Posted Write	0x26	5	0
64-byte Posted Write	0x27	6	0
80-byte Posted Write	0x29	7	0
96-byte Posted Write	0x2A	8	0
112-byte Posted Write	0x2B	9	0
128-byte Posted Write	0x2C	10	0
256-byte Posted Write	0x2D	18	0
16-byte Read	0x2E	2	3
32-byte Read	0x2F	2	4
48-byte Read	0x38	2	5
64-byte Read	0x39	2	6
80-byte Read	0x3A	2	7
96-byte Read	0x3B	2	8
112-byte Read	0x3C	2	9
128-byte Read	0x3D	2	10
256-byte Read	0x3E	2	18

- John D. Leidel, Xi Wang, and Yong Chen. GoblinCore-64: Architectural Specification. Technical report, Texas Tech University, September 2015.

Performance Potential

- Example peak performance caveats:
  - Maximum local scalar bandwidth is limited to 50% of peak DRAM bandwidth (16-byte HMC fetch)
  - Maximum local bandwidth can be achieved using HWACHA SIMD ops
  - 15Gbps SERDES links de-rated to account for DRAM/logic latency (40GB/s per link)
- Device power is based upon measured HMC values and measured RISC-V values
  - Does not account for process differences or process improvements
  - Does not account for long-haul SERDES connectivity, eg, translation to optical

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

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

DISCL

Mapping Communication and Dataflow Paradigms

PROGRAMMING MODELS

Programming Models for PIM

- Many different programming models utilized across historical PIM (PnM) architectures
  - Not all were created equal
  - Not all were successful
- Bit serial:
  - Thinking Machines: Lisp
  - ICL/CPP DAP: Vectorized Fortran; templated C++
  - Terasys: dbC (C-extensions)
- MIMD:
  - EMU Gossamer: Cilk-style threadlets
  - ISI DIVA: distributed shared memory model
- SIMD:
  - Berkeley CRAM/IRAM
- What worked?
  - We have become more adept at expressing parallelism in programming models
  - Data flow, SIMD compilation, SIMD+MIMD (OpenMP 4.X+), Cilk, MapReduce, etc.
- What didn't?
  - We are not yet good at expressing locality
    - *PGAS languages are a great step forward, but they are not widely accepted*
  - Virtual memory is a MAJOR issue
    - *Is just dealing with physical addressing the solution?*
  - Communication (NUMA) latency is hard to express in the instruction set

Compute-Communication/SPMD Programming Model

- We want to experiment with porting very traditional languages to the architecture
- Shared Memory
  - OpenMP 5.x proposed spec has additional clauses to specify certain degrees of physical memory locality
    - *Specified on a per-variable basis using clause modifiers*
  - We would like to experiment with adapting these for PIM architectures
  - OpenMP has been utilized in graph algorithm research (see GAP benchmark suite)
- Partitioned Global Address Space
  - Given our adoption of partitioned physical addresses (from GC64), PGAS languages are a natural fit
  - UPC is the first PGAS language target
    - *Berkeley UPC translator and runtime*
  - Chapel may follow
  - Our biggest concern: minimizing the overhead from the PGAS runtime
  - UPC is not traditionally known for graph algorithm/data intensive computing research

```
#pragma omp memkind(fastmem : val: a , b, persistent: ref: c,d)
```

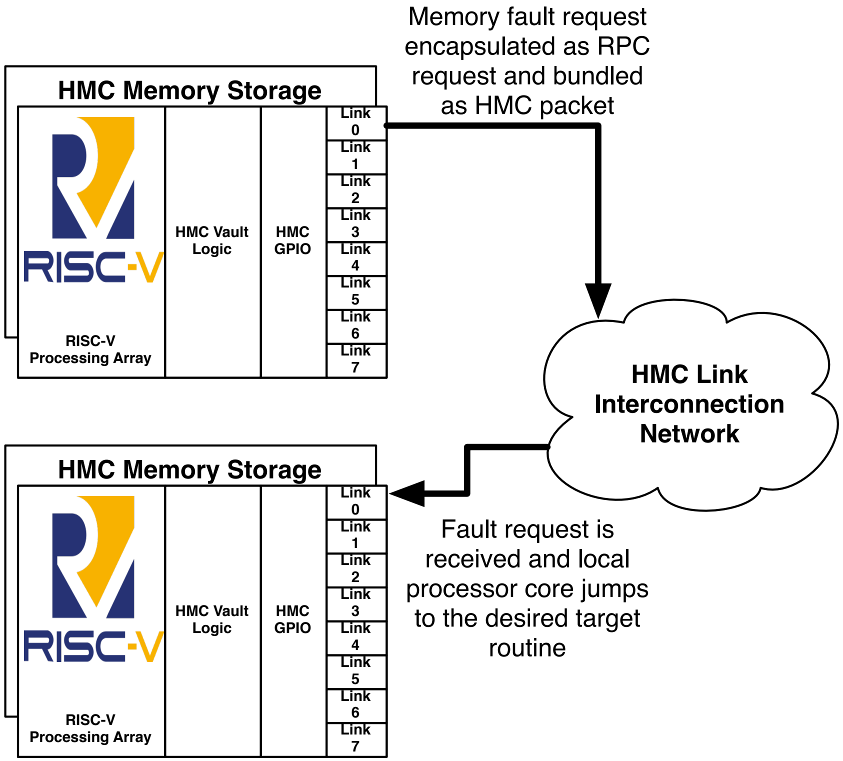
```
#define N <something large>
shared int v1[N], v2[N], v3[N]

upcforall( i=0; i<N; i++ ){
  v3[i] = v1[i] + v2[i];
}
```

http://upc.lbl.gov/

http://www.openmp.org

Data Flow Programming Model

- Data flow model similar to the Emu execution model
  - *Move the compute to the memory*
- Cores are permitted to execute providing they access memory in local cube storage
- Remote memory accesses trigger a fault
  - Faults trigger the equivalent to an RPC request to a remote core co-located with the remote memory
  - Local core builds a message block including the PC and the remote memory location
  - Message is sent to the remote device and queued
  - Remote core picks up the request and begin execution
- Implications?
  - Application is copied *everywhere* in order to avoid moving the entire binary.
  - Applications must efficiently utilize locality when possible
- Target apps:
  - Graph traversals/pointer chasing



Future Research Directions

RISKS, PROGRESS AND FUTURE WORK

Risks

Software

- Near-memory architectures are notoriously difficult to program
  - What additional programming models or feature support do we need to promote locality?
- System software?
  - Memory protection, virtualization, allocation are all difficult tasks when operating close to memory
  - Supervisor versus user-mode execution?
- Tools?
  - Debugging PIM/PnM architectures can be especially tricky
  - How do we exploit temporal and spatial memory locality in our compiler optimizations?

Hardware

- Hardware IP licensing
  - TSV technology is highly process-specific and expensive to license
  - RISC-V, GC64 and our other designs are BSD licensed
  - TSV providers may not be amenable to an open source hardware design
- HMC DRAM Layer Licensing
  - Analog DRAM logic and manufacturing IP is highly proprietary
  - Difficult to persuade few remaining DRAM manufacturers to license only the DRAM portion of the device

DISCL

Current Progress/Future Work

- Initial prototyping efforts have begun to marry the HMC-Sim infrastructure and the RISC-V Spike simulator
  - Fairly significant effort to make HMC-Sim centric in the simulation infrastructure (as opposed to Spike
- Additional prototyping is under way to produce a set communication packet specification for HMC
  - Forthcoming publications (late Spring/early Fall)

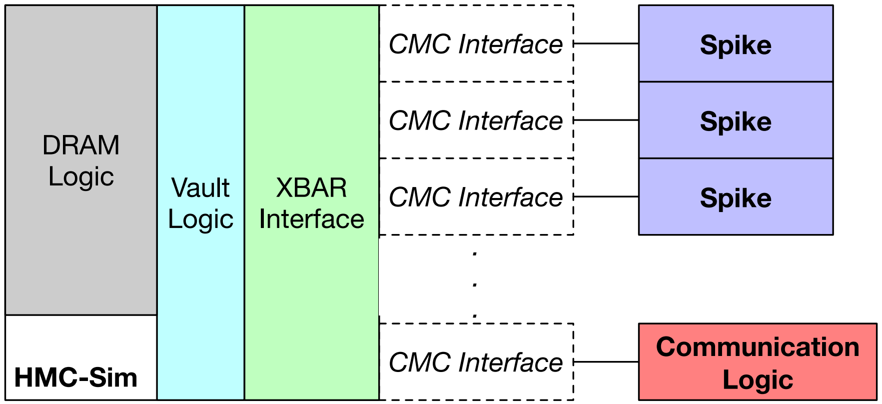


Figure 5a: Compute Node

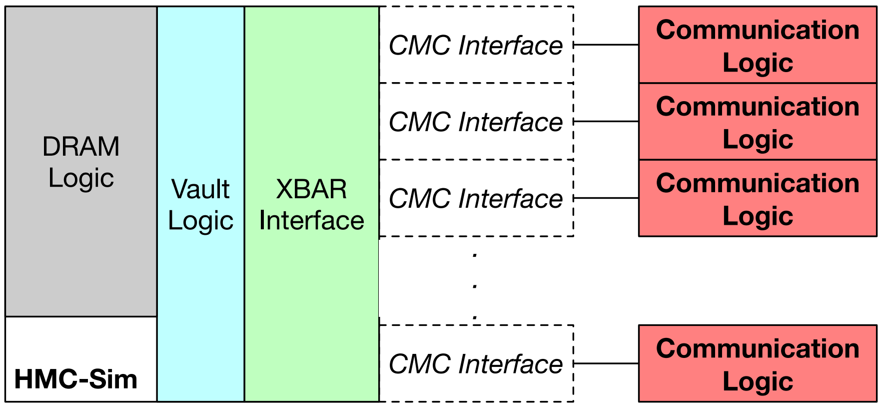


Figure 5b: Router Node

Questions/Comments?

John Leidel

john.leidel<at>ttu.edu

http://gc64.org

http://discl.cs.ttu.edu/gitlab/groups/gc64

TEXAS TECH UNIVERSITY™