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)
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 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 specification
  - Does not consider any one device SKU
  - Packaged as a library written in C
  - Currently integrated into Sandia SST toolkit
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).


HMC to PIM (PnM)

- 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!?
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

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 1.2mm² (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
- 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

Table 3.3: GoblinCore-64 Atomic System Inteconnect Packets

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

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

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

### Table 1. System Architectural Details

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC-V Rocket with HWACHA Area</td>
<td>1.2mm²</td>
</tr>
<tr>
<td>RISC-V Rocket with HWACHA Freq</td>
<td>951Mhz</td>
</tr>
<tr>
<td>RISC-V Rocket with HWACHA Perf</td>
<td>34 GFlops</td>
</tr>
<tr>
<td>RISC-V Cores per HMC</td>
<td>28</td>
</tr>
<tr>
<td>Peak RISC-V Performance</td>
<td>952 GFlops</td>
</tr>
</tbody>
</table>

### Table 2. System Architecture

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC Power</td>
<td>136watt</td>
</tr>
<tr>
<td>5x5x5 Torus Performance</td>
<td>476TFlops</td>
</tr>
<tr>
<td>5x5x5 Torus Memory</td>
<td>1TB</td>
</tr>
<tr>
<td>5x5x5 Torus Power</td>
<td>102KW</td>
</tr>
<tr>
<td>10x10x10 Torus Performance</td>
<td>3.8PFlop</td>
</tr>
<tr>
<td>10x10x10 Torus Memory</td>
<td>8TB</td>
</tr>
<tr>
<td>10x10x10 Torus Power</td>
<td>816KW</td>
</tr>
<tr>
<td>50x50x50 Torus Performance</td>
<td>476PFlop</td>
</tr>
<tr>
<td>50x50x50 Torus Memory</td>
<td>1PB</td>
</tr>
<tr>
<td>50x50x50 Torus Power</td>
<td>103MW</td>
</tr>
</tbody>
</table>

### Processing Architecture

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC Configuration</td>
<td>8Link-8GB Device (15Gbps)</td>
</tr>
<tr>
<td>HMC Area (Compute and Router)</td>
<td>68mm²</td>
</tr>
</tbody>
</table>

### Memory Architecture

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC Power</td>
<td>136watt</td>
</tr>
<tr>
<td>5x5x5 Torus Performance</td>
<td>476TFlops</td>
</tr>
<tr>
<td>5x5x5 Torus Memory</td>
<td>1TB</td>
</tr>
<tr>
<td>5x5x5 Torus Power</td>
<td>102KW</td>
</tr>
<tr>
<td>10x10x10 Torus Performance</td>
<td>3.8PFlop</td>
</tr>
<tr>
<td>10x10x10 Torus Memory</td>
<td>8TB</td>
</tr>
<tr>
<td>10x10x10 Torus Power</td>
<td>816KW</td>
</tr>
<tr>
<td>50x50x50 Torus Performance</td>
<td>476PFlop</td>
</tr>
<tr>
<td>50x50x50 Torus Memory</td>
<td>1PB</td>
</tr>
<tr>
<td>50x50x50 Torus Power</td>
<td>103MW</td>
</tr>
</tbody>
</table>

### System Architecture

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Total Network Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>5x5x5 Torus</td>
<td>30 TB/s</td>
</tr>
<tr>
<td>10x10x10 Torus</td>
<td>240 TB/s</td>
</tr>
<tr>
<td>50x50x50 Torus</td>
<td>30 PB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Bisection Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>5x5x5 Torus</td>
<td>4 TB/s</td>
</tr>
<tr>
<td>10x10x10 Torus</td>
<td>16 TB/s</td>
</tr>
<tr>
<td>50x50x50 Torus</td>
<td>4 PB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Peak Bytes/Flop</th>
</tr>
</thead>
<tbody>
<tr>
<td>5x5x5 Torus</td>
<td>0.063</td>
</tr>
<tr>
<td>10x10x10 Torus</td>
<td>0.063</td>
</tr>
<tr>
<td>50x50x50 Torus</td>
<td>0.063</td>
</tr>
</tbody>
</table>
Mapping Communication and Dataflow Paradigms

PROGRAMMING MODELS
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

```c
#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/
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
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)
Questions/Comments?

John Leidel
john.leidel<at>ttu.edu

http://gc64.org

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