HMC-Sim 2.0: A Simulation Platform for Exploring Custom Memory Cube Operations

John D. Leidel, Yong Chen

May 23, 2016

AsHES 2016

TEXAS TECH
UNIVERSITY

Overview

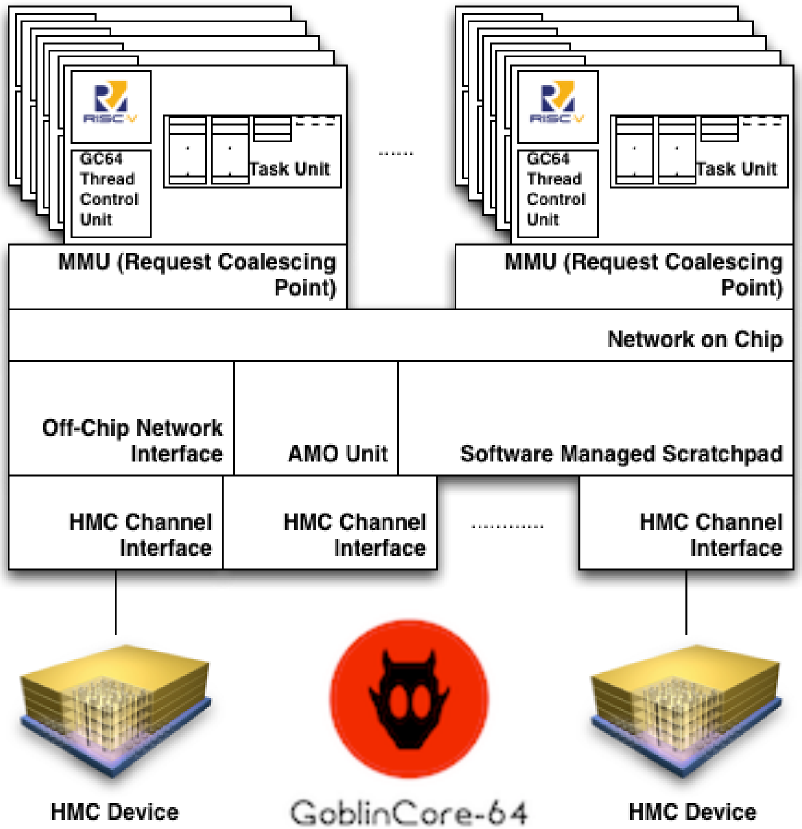
- Introduction & Overview
- CMC Simulation
- Sample CMC Mutexes
- Future Research

Hybrid Memory Cube Device Simulation

INTRODUCTION & OVERVIEW

GC64 Driving Research

- Driving force behind the GC64 architecture research is the ability to find and exploit memory bandwidth
- Exhaustive search on forthcoming memory technologies
  - Traditional DDR/GDDR devices did not provide sufficient accessibility and bandwidth
- Hybrid Memory Cube devices were chosen



http://gc64.org

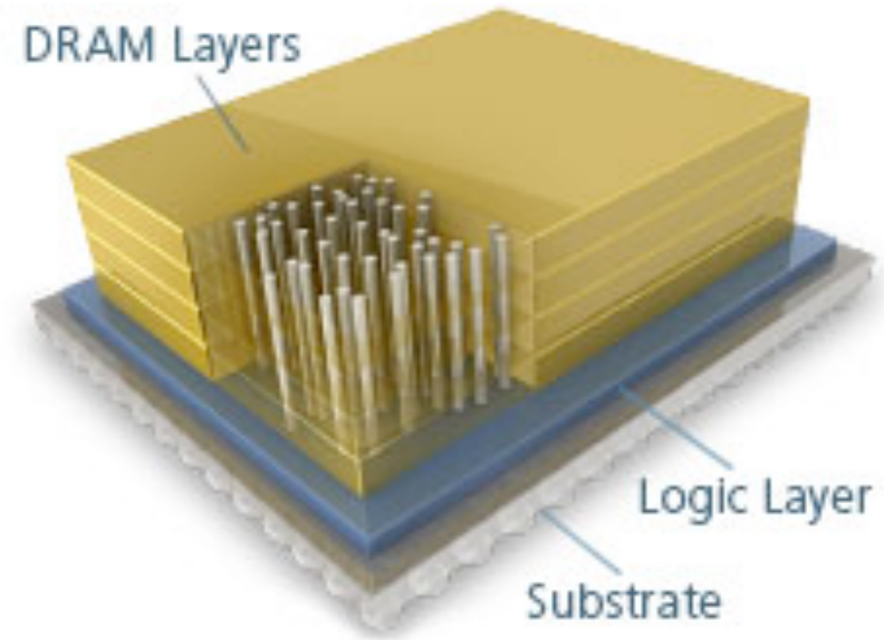
Intro to Hybrid Memory Cube

- Technology
  - Through-silicon-via [TSV] design that combines logic layer and DRAM layers
  - Packetized interface specification the behaves similar to a network device
  - Routing capabilities built into the device logic layer
    - *Device-to-device routing*
- Hybrid Memory Cube Consortium
  - Standards body to drive the public HMC specification.
  - Similar in function to JEDEC for DDR memory
  - http://www.hybridmemorycube.org/

DISCL

HMC TSV Technology

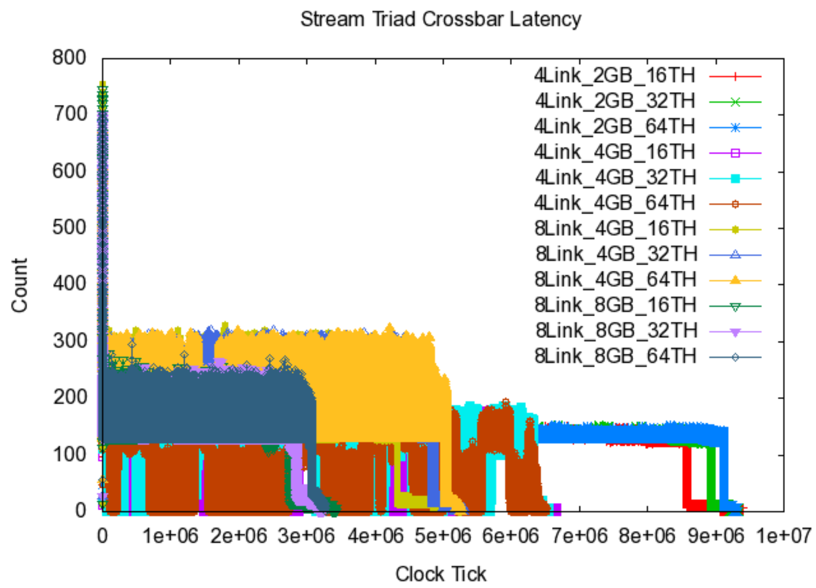
- Substrate
  - Contains the physical pin-out for data, power and ground
  - SERDES
- Logic Layer
  - Contains the logic necessary to perform:
    - *Routing*
    - *Arbitration (weakly ordered)*
    - *Addressing*
    - *AMO*
- DRAM Layers
  - Contains the DRAM arrays



H. M. C. Consortium. Hybrid memory cube specification 2.1, 2015.

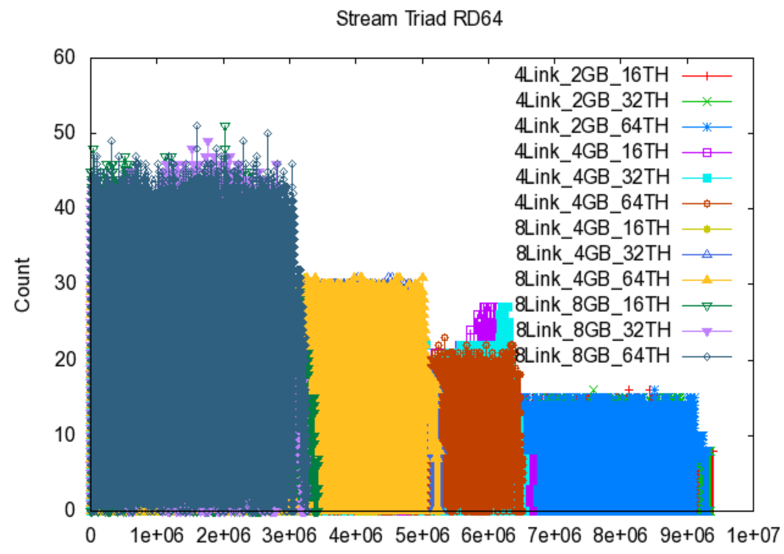
HMC-Sim Overview

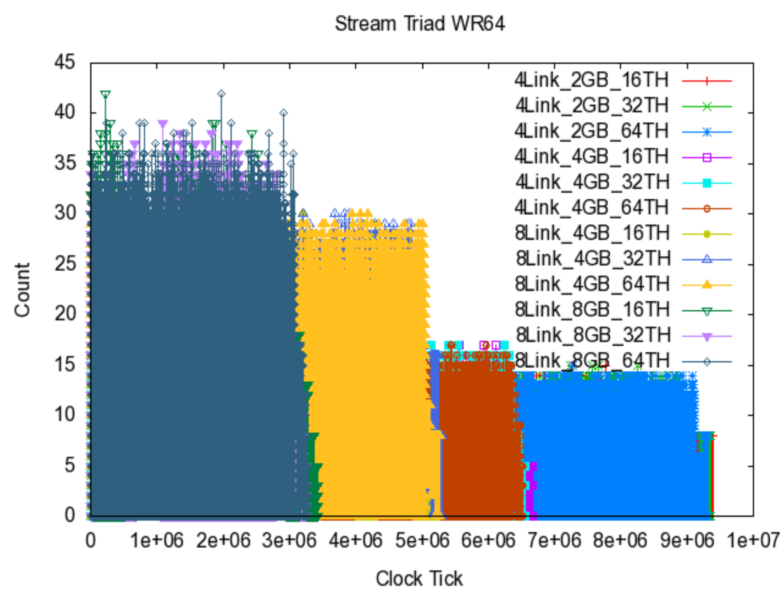
- Our architecture research required access to a configurable HMC simulation platform
  - None existed that were: 1) open source and/or 2) available without an NDA
- We exhaustively studied the HMC specification and developed HMC-Sim based upon the spec
  - …as opposed to a individual device SKU
- HMC-Sim Design Requirements
  - Configurable for different host CPUs (link connectivity, clock frequency, packet configuration, etc)
  - Configuration for different device SKU's
  - Support for device-to-device routing
  - Simulation of all the internal queuing arbitration stages as defined by the spec
  - Cycle-based simulation
  - Discrete logging capabilities
  - Packaged as a library (can be integrated into other high-level simulators)



HMC-Sim 1.0

- Developed the first open source HMC simulation platform
  - Designed to explore how different applications affect memory throughput & latency
  - Becoming the standard for HMC modeling and simulation
- Permits us to model different concurrency mechanisms to determine the best mixture of parallelism and bandwidth across different algorithms and applications





DISCL

HMC-Sim 2.0

- Several users of HMC-Sim requested a number of new features in future revisions:
  - Support for Gen2 HMC specification
  - Gen2 specification's inclusive support for atomic memory operations
  - Gen2 packet specification
  - ***Custom Memory Cube (CMC) exploration***
- CMC Exploration
  - What if we could implement new operations in the HMC logic layer?
  - What if these operations were ***NOT*** just simple memory operations?
  - *Additional Atomic operations, transactional operations, arithmetic reductions, logical reductions, processing near memory, etc*
  - ***If we could have any operation embedded in the HMC logic layer, what would it be?***

Custom Memory Cube Operation Simulation

CMC SIMULATION

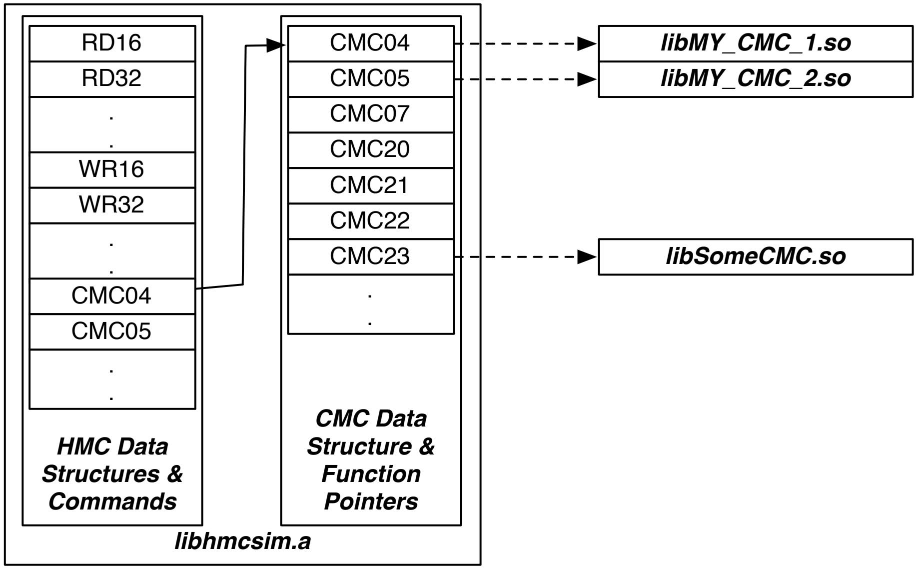
CMC Support Requirements

- API Compatibility:
  - Existing integration with other simulators shouldn't be broken (Sandia SST)
- External Implementation:
  - CMC implementer should focus on CMC, not learning HMC-Sim internals
- Creative Experimentation
  - No limitation to the user's creativity in implementing CMC ops
- Utilize Existing HMC Packet Formatting
  - Existing crack/decode logic should be maintained
- Discrete Tracing
  - HMC-Sim 1.0 had extensive support for logging, CMC ops will need this as well
- Separable Implementation
  - Current HMC-Sim is BSD licensed. We want to make sure users can develop/distribute their CMC ideas separate from the simulator
- No Simulation Perturbation
  - No perturbation to existing simulation results!

DISCL

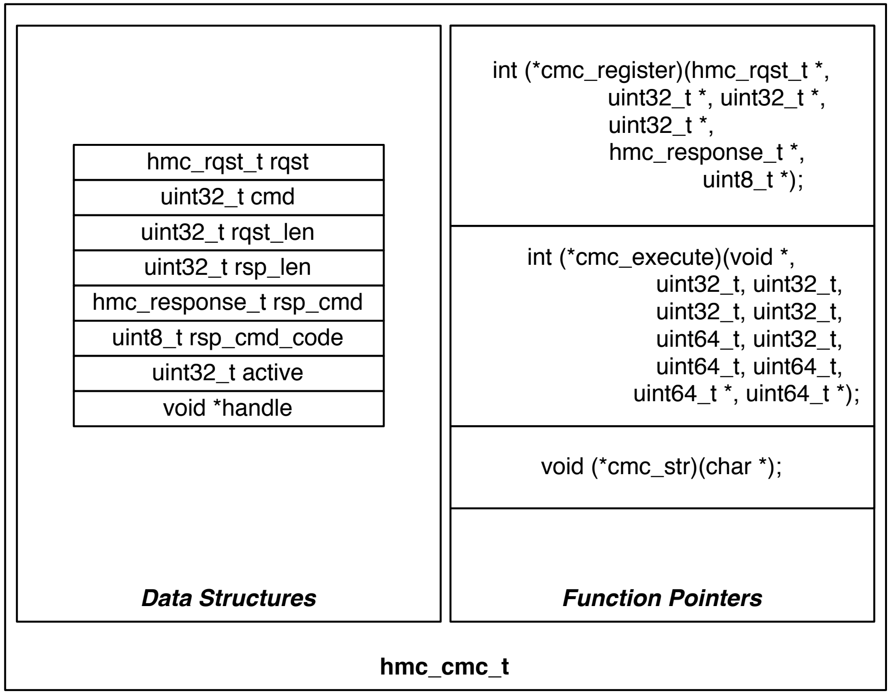
CMC Support Architecture

- We explicitly map all the unused HMC opcodes to CMC* ops
  - 70 potential CMC opcodes
- We provide a template infrastructure to construct a single CMC operation mapped to a single opcode in a shared library
- We provide one additional API interface to load the CMC shared library at runtime
- **Runtime processing is otherwise the same for CMC operations!**



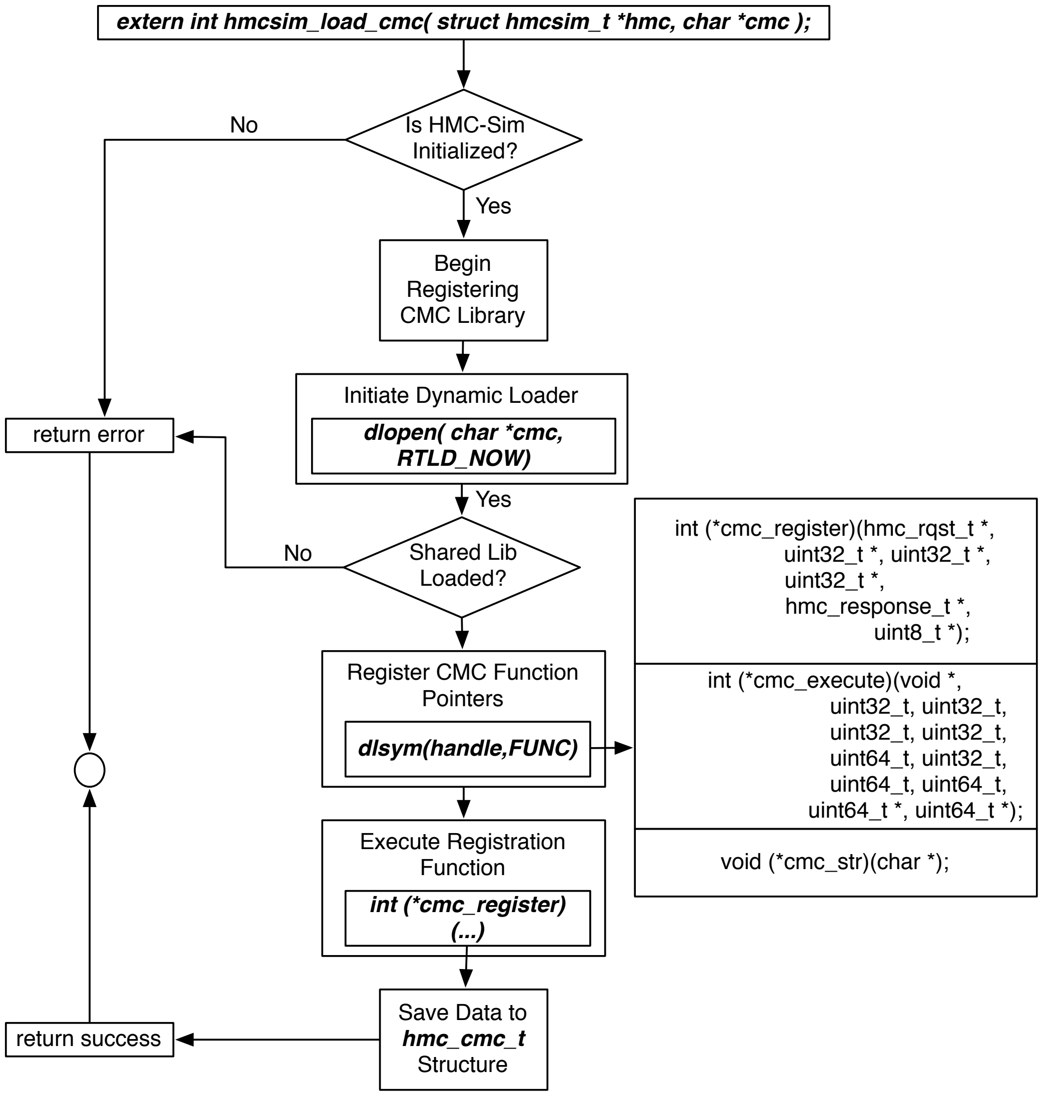
CMC Library Architecture

- The CMC library requires the user to define structure of the CMC operation:
  - CMC Name (string): used for logging
  - Request command enum (from the list of 70)
  - Request & Response packet lengths
  - Response command enum (can be custom response)
- One function must be implemented by the user:
  - ***hmcsim_execute_cmc()***
- ***Everything else is provided in our example CMC implementation***

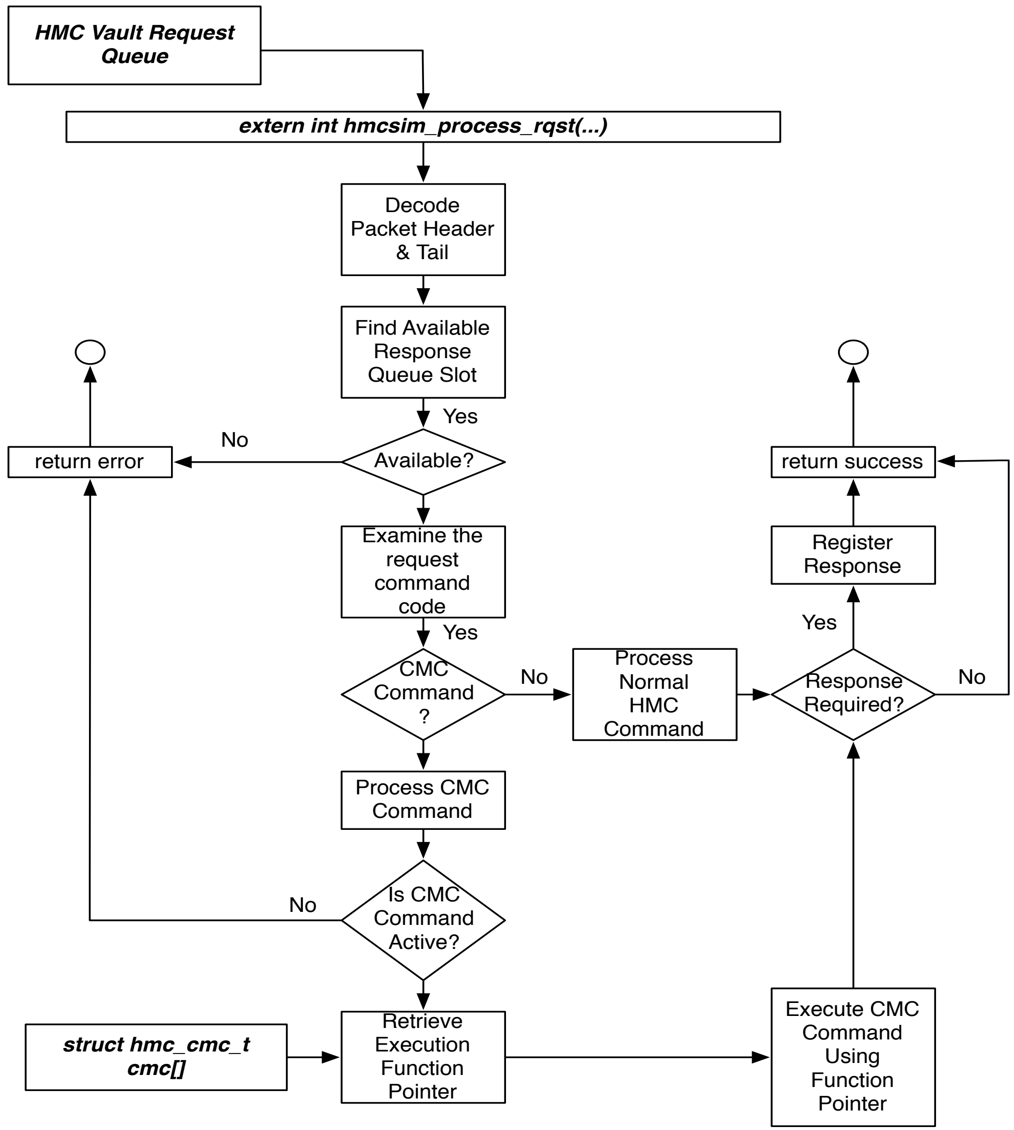


CMC Tutorial:
http://gc64.org/?page_id=140

CMC Registration



CMC Processing



Locking Primitives as CMC Operations

CMC MUTEXES

CMC Mutexes

- We implemented several CMC commands as initial tests
- What if we could accelerate traditional mutex operations?
  - *HMC_LOCK*
  - *HMC_TRYLOCK*
  - *HMC_UNLOCK*
- Designed to perform pthread-style mutex operations
  - **does not block on HMC_LOCK

Thread/Task ID	*Lock*
127 … 64	63 … 0

- Each HMC mutex payload is a 16-byte memory location
- Lower 8 bytes: LOCK region
- Upper 8 bytes: Thread/Task ID
  - “Owner” of the LOCK region
  - Relative to the user’s process space
- 16-bytes is wasteful… but
  - 16-bytes in the minimum request size for normal HMC RD/WR requests
  - Minimal logic overhead required to implement our mutexes

CMC Mutex Implementation

Operation	*Pseudocode*	*Command Enum*	*Request Command*	*Request Length*	*Response Command*	*Response Length*
hmc_lock	IF (ADDR[63:0] == 0){ ADDR[127:64 = TID; ADDR[63:0]=1; RET 1}ELSE{ RET 0 }	CMC125	125	2 FLITS	WR_RS	2
hmc_trylock	IF (ADDR[63:0] == 0){ADDR[127:64 = TID; ADDR[63:0]=1; RET ADDR[127:64]}ELSE{ RET ADDR[127:64] }	CMC126	126	2 FLITS	RD_RS	2
hmc_unlock	IF (ADDR[127:64] == TID && ADDR[63:0] == 1){ ADDR[63:0] = 0; RET 1}ELSE{ RET 0 }	CMC127	127	2 FLITS	WR_RS	2

HMC_LOCK

```
if( LOCK == 0 ){
  TID = MY_TID;
  LOCK = 1;
  return 1;
}else{
  return 0;
}
```

HMC_TRYLOCK

```
if( LOCK == 0 ){
  TID = MY_TID;
  LOCK = 1;
  return TID;
}else{
  return TID;
}
```

HMC_UNLOCK

```
if( TID == MY_TID
  && LOCK == 1){
  LOCK = 0;
  return 1;
}else{
  return 0;
}
```

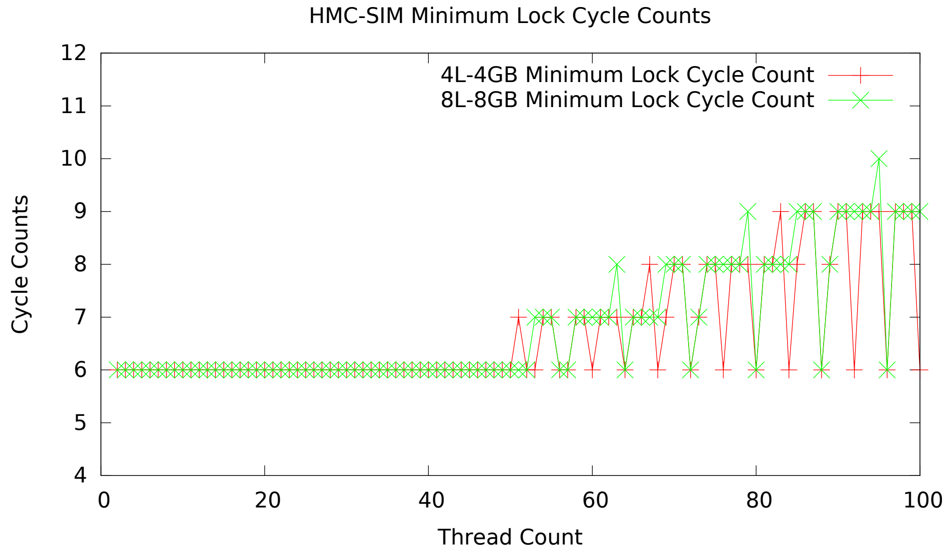
CMC Mutex Experimentation

- Attempt to perform naïve spin-wait locks on a single mutex location
- Deliberate hot-spotting
- Scale the number of parallel threads/ tasks from 2-100
- Execute the tests for different HMC configurations
  - 4LINK-4GB
  - 8LINK-8GB
- Record:
  - ***Min_Cycle***: Minimum number of cycles for any thread to obtain the lock
  - ***Max_Cycle***: Maximum number of cycles for any thread to obtain the lock
  - ***Avg_Cycle***: Average number of cycles for all threads to obtain the lock

Algorithm 1 CMC Mutex Algorithm

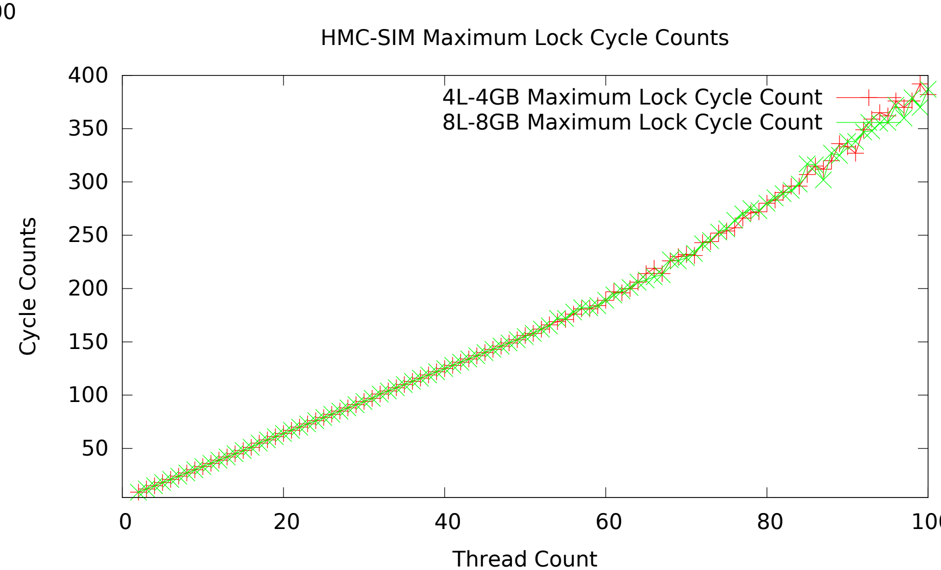
```
for Nthreads do
  HMC_LOCK(ADDR)
  if LOCK_SUCCESS then
    HMC_UNLOCK(ADDR)
  else
    HMC_TRYLOCK(ADDR)
    while LOCK_FAILED do
      HMC_TRYLOCK(ADDR)
    end while
    HMC_UNLOCK(ADDR)
  end if
end for
```

CMC Mutex Min and Max Cycle Results

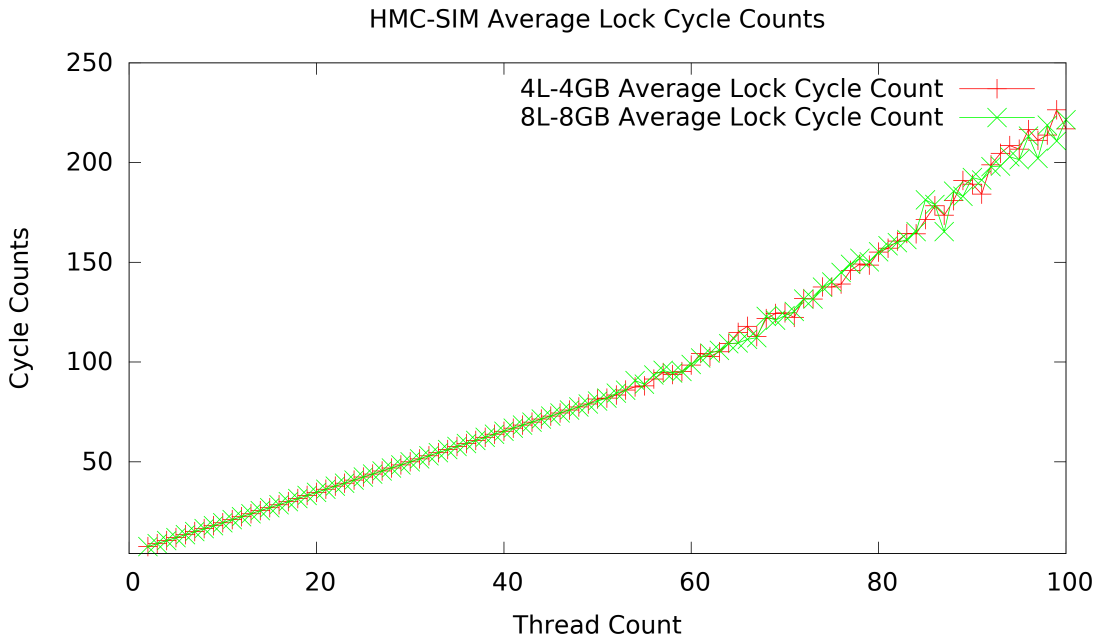


- Cycle counts are in HMC logic cycles (not host cycles)
- 4LINK-4GB device has slightly higher maximum latency
- Identical minimum latencies

Device	*Min Cycle Count*	*Max Cycle Count*	*Avg Cycle Count*
4Link-4GB	6	392	226.48
8Link-8GB	6	387	221.48



CMC Mutex Average Cycle Results



- 8LINK-8GB device has slightly lower average and maximum latencies
- For latency-sensitive applications dependent upon primitive locking operations (embedded applications), the additional queuing capacity with more links is helpful
- The weak ordering of the HMC device promotes ***sub-linear*** scaling for both device configurations!

Additional Possibilities in CMC Exploration

FUTURE RESEARCH

Future CMC Simulation Research

What other common operations would be interesting to simulate as CMC operations?

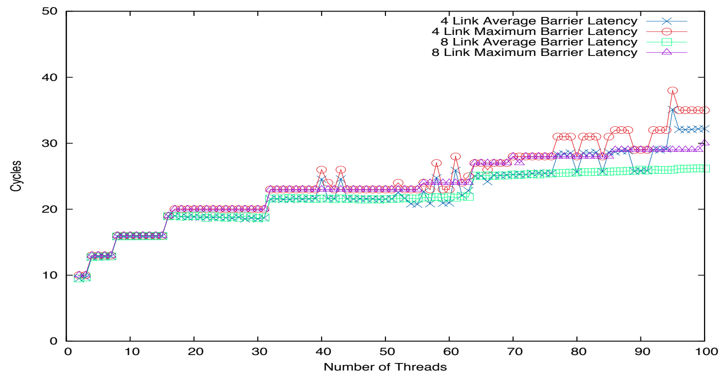
Currently packaged with HMC-Sim:

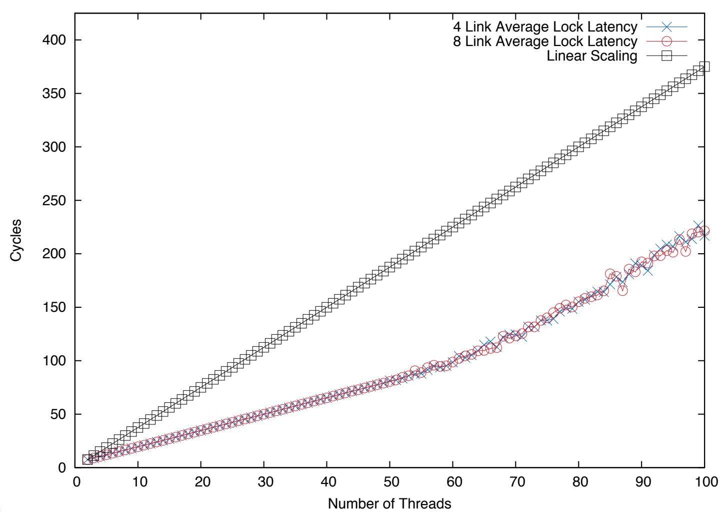
- Atomic Popcount
- HMC Lock
- HMC Trylock
- HMC Unlock
- HMC Full Empty Bit Ops**

Other Interesting Operations:

- Reductions
- Sorting
- Bitwise Atomics
- Processing Near Memory

Full Empty Bit CMC Operations





Simulating Fine-Grained Locking Primitives:

- Similar to MTA/XMT style full-empty (tag) bit operations
- Performs *read-modify-write* on lock bits and data payloads with a single command
- Splits the storage in the HMC array into tag bit vectors and data payloads for better concurrency
- Supports full complement of tag-bit operations
- ***Publication accepted for MemSys 2016***

Questions

John Leidel

john.leidel@ttu.edu

Yong Chen

yong.chen@ttu.edu

HMC-Sim Development and Tutorials:

http://gc64.org

TEXAS TECH UNIVERSITY™