Architectural Models in Graphite

What’s available and how to extend
Overview

- **Architectural Models**
  - List of available models in Graphite
  - Configuration Parameters
  - Base Class / Main Interface Functions
Architectural Models - Outline

• Network Model
• Memory Subsystem
• Core Model
• Contention Model
• Heterogeneity
• Dynamic Frequency Scaling (DFS)
Architectural Models - Outline

- Network Model
- Memory Subsystem
- Core Model
- Contention Model
- Heterogeneity
- Dynamic Frequency Scaling (DFS)
Network Model

- Models the on-chip interconnection network
- Graphite supports 5 networks
  - 2 for user-level messages
  - 2 for shared memory messages
  - 1 for system messages
Network Model
Existing Models in Graphite

• 3 models for electrical mesh networks
  – Hop Counter
  – Analytical
  – Hop-By-Hop (Support broadcast tree)

• Electrical Clos network model

• Magic network model
  – unicast/multicast/broadcast takes 1 cycle
Network Model
Performance and Accuracy Trade-offs

- Electrical mesh networks

<table>
<thead>
<tr>
<th></th>
<th>Hop Counter</th>
<th>Analytical</th>
<th>Hop By Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Routing</td>
<td>Sent directly to destination core</td>
<td>Sent directly to destination core</td>
<td>Stops at intermediate routers (X-Y routing)</td>
</tr>
<tr>
<td>Contention Modeling</td>
<td>No</td>
<td>Models global contention</td>
<td>Models per link contention</td>
</tr>
<tr>
<td>Performance</td>
<td>High (+)</td>
<td>High (+)</td>
<td>Low (-)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Low (-)</td>
<td>Medium</td>
<td>High (+)</td>
</tr>
</tbody>
</table>
Network Model

Limitations

• Only link contention modeled
• Infinite output buffering assumed
Network Model
Configuration + Interface

• Configuration parameters
  – [network]
    user_model_1 = emesh_hop_counter
  – Network model-specific configuration params
    • [network/emesh_hop_counter/router]
      delay = 1

• Base Class / Interface Functions
  – class NetworkModel (common/network/network_model.h)
  – routePacket()
  – processReceivedPacket()
Architectural Models - Outline

• Network Model
• **Memory Subsystem**
  – Caches
  – Directory
  – Memory Controller
• Core Model
• Contention Model
• Heterogeneity
• Dynamic Frequency Scaling (DFS)
Memory Subsystem

• Handles Load/Store requests from cores
• Distributed components communicate using on-chip network
• Both a functional and performance model
  – Verify the correctness of cache coherence protocols
• Currently have memory subsystems with:
  – Private L1/L2 caches
  – Directory-based coherence protocols
  – Directory attached to memory controller
• Two coherence protocols
  – MSI
  – MOSI
Memory Subsystem – Caches

• Building block for private/shared caches
• Set-associative caches with configurable
  – Cache Size, Cache Block Size, Associativity, Replacement Policy, Access Time
• Used to implement
  – Private L1-I, Private L1-D, Private L2 Caches
Memory Subsystem – Caches
Configuration + Interface

• Configuration Parameters
  – [perf_model/l1_icache/T1]
    associativity = 4
  – [perf_model/l1_dcache/T1]
    associativity = 4
  – [perf_model/l2_cache/T1]
    associativity = 8

• Base Class / Interface Functions
  – class Cache (common/tile/memory_subsystem/cache/cache.h)
    – accessSingleLine()
    – insertSingleLine()
    – invalidateSingleLine()
Memory Subsystem – Directory

• Organized as a cache and placed at memory controllers

• Each directory entry can be organized as follows:
  – Full-Map Directory
    • $N$ bits

<table>
<thead>
<tr>
<th>Address</th>
<th>State</th>
<th>Sharer List ($N$ bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>Shared</td>
<td>0 1 1 1 ... 0 1</td>
</tr>
</tbody>
</table>
Memory Subsystem – Directory

• Organized as a cache and placed at memory controllers

• Each directory entry can be organized as follows:
  – Full-Map Directory
    • $N$ bits
  – Limited Directory (Dir$_k$NB, Dir$_k$B, ACKwise)
    • $k\log_2(N)$ bits

<table>
<thead>
<tr>
<th>Address</th>
<th>State</th>
<th>Sharer List (k hardware pointers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>Shared</td>
<td>$Core_{s1}$ $Core_{s2}$ ... $Core_{sk}$</td>
</tr>
</tbody>
</table>
Memory Subsystem – Directory
Configuration + Interface

• Configuration Parameters
  – [perf_model/dram_directory]
    directory_type = full_map

• Base Class / Interface Functions
  – class DirectoryEntry
    (common/tile/memory_subsystem/directory_schemes/directory_entry.h)
    – add(remove)Sharer()
    getSharersList()
    get(set)Owner()
Memory Subsystem – Memory Controller

• Controller for off-chip memory
• Has contention model for accessing DRAM
• Memory Requests served in FIFO order

• Configuration Parameters
  – [perf_model/dram]
    latency = 100 ns
Architectural Models - Outline

- Network Model
- Memory Subsystem
- **Core Model**
- Contention Model
- Heterogeneity
- Dynamic Frequency Scaling (DFS)
Core Model

• Models the instruction fetch/decode and execution units
• Input: Instruction stream from Pin + Dynamic information from the network and memory
• “Special instructions” used to model message passing and synchronization
• Purely a performance model
Core Model
Existing Models + Limitations

• Models assume constant instruction costs except for memory and branch prediction

• Existing Models in Graphite
  – iocoom
    • In-order core model with out-of-order memory completion
  – simple
    • In-order core model that adds all latencies

• Limitations
  – No pipelining/superscalar support
• **Configuration Parameters (General + Model-Specific)**
  - `[perf_model/core/static_instruction_costs]`  
    ```none
    add = 1
    ```
  - `[perf_model/core/iocoom]`
    ```none
    num_outstanding_loads = 32
    ```

• **Base Class / Interface Functions**
  - `class CoreModel`
    ```none
    (common/tile/core/core_model.h)
    ```
  - `queueBasicBlock()`
  - `queueDynamicInstruction()`
  - `push(pop)DynamicInstructionInfo()`

• **We are in need of more sophisticated core models with pipelining/superscalar support**
  - Your contributions are welcome!
Architectural Models - Outline

• Network Model
• Memory Subsystem
• Core Model
• Contention Model
• Heterogeneity
• Dynamic Frequency Scaling (DFS)
Contestion Models

• Compute the queuing delay when accessing a shared object (server)

• Examples of shared objects
  – Network Links
  – Off-chip Memory
  – Shared Caches
Contention Models
A hard problem in Graphite!

- NOT easy as in a cycle-accurate simulator
- Each core has independent clock
- Packets arrive with out-of-order timestamps
Contention Models
A hard problem in Graphite!

- NOT easy as in a cycle-accurate simulator
- Each core has independent clock
- Packets arrive with out-of-order timestamps
Contention Models
A hard problem in Graphite!

- NOT easy as in a cycle-accurate simulator
- Each core has independent clock
- Packets arrive with out-of-order timestamps

![Diagram](image)

Core 1

Memory Controller

Core 2

Time = 20
Length = 5

Queuing Delay?
Contention Models
Existing Models in Graphite

• History Tree
  – Queuing delay calculated from **Server Utilization History**

• Analytical M/G/1 Queuing Model
  – Queuing delay calculated from **Average Server Utilization, Packet Length Statistics**
Contestion Models – History Tree

- Utilization History stored as a set of free intervals
- Each free interval corresponds to a time interval when server is free
- Free intervals organized as a self-balancing binary tree
  - Tree Rotations used for height balancing
- Queuing Delays calculated by searching the history tree
  - Usually $O(\log_2 N)$ complexity for search
- Size of history tree adjusted according to maximum permissible clock skew
Contention Models – History Tree

Example

• Packet with length 5 arrives at time 52
• History Tree Structure shown below
  – (70 – 74), (100 – 110) are free intervals
  – (15 – 35), (110 – 167) are utilized intervals
• Compute Queuing Delay (time = 52, length = 5)
Contestion Models – History Tree

Example (cont’d)

- Compute Queuing Delay (time = 52, length = 5)
• Compute Queuing Delay (time = 52, length = 5)

Queuing Delay = 7
• Compute Queuing Delay (time = 52, length = 5) → 7
• Compute Queuing Delay (time = 52, length = 5) \(\rightarrow\) 7
• History Tree adjusted according to reflect server utilization
Contestion Models – Analytical M/G/1

- Can be used for uniform random traffic
- Stores the following 3 variables across entire history
  - Net Utilization ($\rho$)
  - Average Packet Length ($\mu_{len}$)
  - Standard Deviation of Packet Length ($\sigma_{len}^2$)

\[ \text{QueueingDelay} = \frac{\rho}{2(1-\rho)} \left( \mu_{len} + \frac{\sigma_{len}^2}{\mu_{len}} \right) \]
## Contention Models

### Performance Accuracy Trade-off

<table>
<thead>
<tr>
<th></th>
<th>History Tree</th>
<th>Analytical M/G/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Adaption to</td>
<td>Yes</td>
<td>No (Averages across entire history)</td>
</tr>
<tr>
<td>Network Traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>Low (-)</td>
<td>High (+)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>High (+)</td>
<td>Low (-)</td>
</tr>
</tbody>
</table>

- We are working on comparing these contention models to a cycle accurate one.
• Configuration Parameters
  – [network/emesh_hop_by_hop_basic/queue_model]
    type = history_tree
  – Model-specific configuration params
    • [queue_model/history_tree]
      max_list_size = 100

• Base Class / Interface Functions
  – class QueueModel
    (common/shared_models/queue_model.h)
  – computeQueueDelay(uint64_t time, uint64_t length)
Architectural Models - Outline

- Network Model
- Memory Subsystem
- Core Model
- Contention Model
- Heterogeneity
- Dynamic Frequency Scaling (DFS)
Heterogeneity

• Cores can be configured with the following performance asymmetries:
  – Frequency
  – Different Core Models (magic/iocoom)
  – Private L1/L2 Cache Sizes and Organizations

• All cores follow the same ISA
Heterogeneity
Uses and Limitations

• Uses
  – Modeling Helper Core
    • Core – very low frequency and simple core model
  – Modeling 1 Big Core + Many small cores
    • Big core – high frequency and sophisticated core model

• Limitations
  – No multithreading (support in progress)
Heterogeneity
Configuration

• Configuration Parameters
  – Format of each tuple
    < number of cores, frequency, core model, L1-I Cache Config, L1-D Cache Config, L2 Cache Config >
  – [perf_model/core]
    model_list = “<30,1.0,simple,T1,T1,T1>,<2,2.5,iocoom,T2,T2,T2>”
Architectural Models - Outline

• Network Model
• Memory Subsystem
• Core Model
• Contention Model
• Heterogeneity

• **Dynamic Frequency Scaling (DFS)**
Dynamic Frequency Scaling (DFS)

• Core Dynamic Frequency Scaling
  – Core Frequency can be increased/decreased at runtime by inserting a call in the application
  – See `common/user/dvfs.h` for APIs

• Network/Memory operate at constant frequency