On Chip Interconnects for TeraScale Computing

Partha Kundu, Microprocessor Tech. Labs Intel Corporation

Computer Arch Day
April 2, 2009
Princeton University
Overview of Talk

- intel 80-core research prototype
- Support for Fine Grained Parallelism
- Partitioning, Isolation and QoS in Interconnects
80-Core Prototype: Router Design
Teraflops Research Processor

Goals:
Deliver Tera-scale performance
- Single precision TFLOP at desktop power
- Frequency target 5GHz
Prototype two key technologies
- **On-die interconnect fabric**
- 3D stacked memory
Develop a scalable design methodology
- **Tiled design approach**
- **Mesochronous clocking**
- Power-aware capability
Tiled Design & Mesh Network

Assemble & Validate

Step and repeat
Key Ingredients for Teraflops on a Chip

Core
Communication
Technology
Clocking
Power management techniques

High performance
Dual FPMACs

High bandwidth
Low latency router

Crossbar Router

2KB Data memory
(DMEM)
3KB Inst. Memory (IMEM)
6-read, 4-write 32 entry RF

65nm
eight metal CMOS

MSINT MSINT MSINT MSINT MSINT MSINT MSINT MSINT

FPMAC0 x + Normalize 32 32
FPMAC1 x + Normalize 32 32
Industry leading NoC

8x10 mesh
- Bisection bandwidth of 320GB/s
- 40GB/s peak per node
- 4-byte bidirectional links
- 6 port non-blocking crossbar
- Crossbar/Switch double-pumped to reduce area

Router Architecture
- Source routed
- Wormhole switching
- 2 virtual lanes
- On/off flow control

Meso-chronous clocking
- Key enabler for tile based approach
- Tile clock @ 5Ghz
- Phase-tolerant synchronizers at tile interface

High bandwidth, Low-latency fabric
Router Architecture

Five port, 5-stage, two lane, 16-FLIT FIFO, 100GB/s
Shared crossbar architecture, two-stage arbitration
Double-pumped Crossbar Router

**Work in ISSCC 2001**
Scaled to 65nm

<table>
<thead>
<tr>
<th>Router</th>
<th>ISSCC '01</th>
<th>This Work</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>284K</td>
<td>210K</td>
<td>26%</td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>0.52</td>
<td>0.34</td>
<td>34%</td>
</tr>
<tr>
<td>Latency</td>
<td>6</td>
<td>5</td>
<td>16.6%</td>
</tr>
</tbody>
</table>
Mesochronous Interface (MSINT)

Circular FIFO, 4-deep
Programmable strobe delay
Low-latency setting

Tx_data → 38

4-deep FIFO

Tx_clk

Delay Line

write state machine

read state machine

Sync Sync

Synchronizer Circuit

4:1 Stg 1

Rx_data

Rx_clk

Scan Register

Low latency

10
Low Power Clock Distribution

Global mesochronous clocking, extensive clock gating
Fine Grain Power Management

• Modular nature enables fine-grained power management
• New instructions to sleep/wake any core as applications demand
• Chip Voltage & freq. control (0.7-1.3V, 0-5.8GHz)

Dynamic sleep

STANDBY:
• Memory retains data
• 50% less power/tile
FULL SLEEP:
• Memories fully off
• 80% less power/tile

1680 dynamic power gating regions on-chip

21 sleep regions per tile (not all shown)

80% less power/tile

12
Leakage Savings

- NMOS sleep transistors
  Regulated sleep for memory arrays
  Impact - Area : 5.4%, $F_{MAX}: 4\%$

2X-5X leakage power reduction

$V_{REF} = V_{cc} - V_{MIN} + \text{wake}$

Memory Clamping

Processing Engine (PE)

Est. breakdown @ 1.2V 110C

82mW $\rightarrow$ 70mW
42mW $\rightarrow$ 8mW
15mW $\rightarrow$ 7.5mW
63mW $\rightarrow$ 21mW
100mW $\rightarrow$ 20mW
100mW $\rightarrow$ 20mW

$V_{cc}, V_{ssv}, V_{MIN}$
Router Power

Activity based power management
Individual port enables
- Queues on sleep and clock gated when port idle

Measured 7X power reduction for idle routers
Estimated Power Breakdown

Communication Power

- **Clocking**: 33%
- **Queues** + **Datapath**: 17% + 22%
- **Arbiters** + **Control**: 7%
- **Links**: 6%
- **Crossbar**: 15%
- **MSINT**: 6%
- **Clocking w/ forwarded clock can be expensive**
- **Significant (>80%) power in router compared to wires**
- **Router power primarily in Crossbar and Buffers**

**4GHz, 1.2V, 110°C**
Energy efficiency of Interconnection networks

• Topology work:
  o low diameter:
    Concentrated Mesh, Balfour et al, ICS 2006
    Flattened Butterfly, Kim et al, Micro 2008
    Multi-drop Express Channels, Grot et al, HPCA 2009

• Router micro-architecture:
  o Minimize dynamic buffer usage:
    Express Virtual Channels, Kumar et al, ISCA 2007
  o Reduce Buffers:
    Rotary Router: Abad et al, ISCA 2007
    ViChaR, Nicopoulos et al, Micro 2006

• Clocking schemes:
  Synchronous vs. GALS (Async, Meso-chronous)
Support for Fine-Grained Parallelism
Flavors of Parallelism

Support multiple types of parallelism
- Vector parallelism
- Loop (non-vector) parallelism
- Task parallelism (irregular)

➢ A single RMS application might use multiple types of parallelism

➢ Sometimes even nested

Need to support them at the same time
Asymmetry

Sources of Asymmetry
- Applications
- MCA: Heterogeneous cores, SMT, NUCA Cache Hierarchies

fine-grained parallelism can mitigate performance asymmetry

Significant inefficiency due to load imbalance

On 8-core MCA:
8 Tasks

Less load imbalance → 25% better performance

On 8-core MCA:
32 Tasks
# Platform Portability

Consider a 8-core MCA

<table>
<thead>
<tr>
<th>Tasks</th>
<th>On 4 cores</th>
<th>On 8 cores</th>
<th>On 6 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Tasks</td>
<td>4X speedup</td>
<td>8X speedup</td>
<td>4X speedup</td>
</tr>
<tr>
<td></td>
<td>On 4 cores</td>
<td>On 8 cores</td>
<td>On 6 cores</td>
</tr>
<tr>
<td></td>
<td>4X speedup</td>
<td>8X speedup</td>
<td>4X speedup</td>
</tr>
<tr>
<td>32 Tasks</td>
<td>4X speedup</td>
<td>8X speedup</td>
<td>5.3X speedup</td>
</tr>
</tbody>
</table>

- On 4 cores: 4X speedup
- On 8 cores: 8X speedup
- On 6 cores: 4X speedup
- On 4 cores: 4X speedup
- On 8 cores: 8X speedup
- On 6 cores: 4X speedup
- On 4 cores: 4X speedup
- On 8 cores: 8X speedup
- On 6 cores: 5.3X speedup
Platform Portability Cont’d

Requires *finer-granularity* parallelism

Even an order of magnitude
Problem Statement

Fine-grained parallelism needs to be efficiently supported in MCA

- Several key RMS modules exhibit very fine-grained parallelism
- Platform portability requires application to expose parallelism at a finer granularity
- Account for asymmetries in architecture

Carbon: Architectural Support for Fine-Grained Parallelism, Kumar et al
ISCA 2007
Loop-Level parallelism

Most common form of parallelism supported by OpenMP, NESL

Requires dynamic load balancing

Performance potential over optimized S/W
Sparse MVM on 64 cores

Significant performance potential

Computation per iteration (tile) can vary dramatically

Speedup (%)

<table>
<thead>
<tr>
<th>Problem Input</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c18</td>
<td>271%</td>
</tr>
<tr>
<td>gis</td>
<td></td>
</tr>
<tr>
<td>pds</td>
<td></td>
</tr>
<tr>
<td>Pilot</td>
<td></td>
</tr>
</tbody>
</table>

13–271%
Task-Level Parallelism

Irregular structured parallelism
- Trees to complex dependency graphs

Significant performance potential
- 15-32% for Backward solve
- Up to 241% for Canny
Typical Parallel Program

```
void task(node)
{
    Do(node);
    foreach child of node
        Enqueue(task, child);
}
```

Example module: forward solve
Use task parallelism

Task = unit of work

Task dependence graph

Sparse matrix

Non-zero
Typical Parallel Run-Time System

```
void task(node) {
    Do(node);
    foreach child of node
        Enqueue(task, child);
}
```

Software data structure
(Holds tasks)

Run-time system creates tuple to represent task
- GPUs, conventional S/W (e.g., TBB) do this today

Multiple implementation options
Need for Hardware Acceleration

Software “Enqueue” & “Dequeue” is slow
- Serial access to head/tail
Additional overhead for smart ordering of tasks
- Placement, Cache/data locality, process prioritization, etc.
Overheads increase with more threads

For “frequent” enqueue/dequeue
- resource checking overhead is wasteful
- hardware does a fine job w/scheduling
- allow fall-back to software on hardware queue limit (overflow/underflow)

⇒ Accelerate data structure accesses & task ordering with H/W
uArch Support for Carbon

Local Task Unit (LTU)
Prefetches and buffers tasks

Global Task Unit (GTU)
Task pool storage
Performance: Loop/Vector Parallelism

- **Significant performance benefit over optimized S/W**
  - 88% better on average
- **Similar performance to “Ideal”**
  - 3% worse on average
Performance: Task-Level Parallelism

- Significant performance benefit over optimized S/W
  - 98% better on average
- Similar performance to “Ideal”
  - 2.7% worse on average
Task Queuing in the Interconnect

• Arrangement of GTU and LTU suffices for apps studied

• But, long vectors running on MCA can be tricky
  ○ requires dynamic resource discovery
  ○ Vector length breaks

  *Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow, Fung et al (Micro ‘07)*
Partitioning, Isolation and QoS in Interconnects
Virtualization and Partitioning of on-chip Resources

Virtualize: interconnect, cache, memory, I/O, etc
Partition: Cores, private caches, dedicated interconnect

Focus:
Domain isolation for performance and security

Allow “arbitrary” shaped domains

Shared Channel Reservation

Application Isolation enforced by Interconnect
Isolation with Fault Tolerance

Reconfigure Partitions

Good cores become un-usuable

Enable Fault Discovery & Repartitioning
Route Flexibility

Motivation
- Performance isolation
- Fault-tolerance
- Topology independence
- Load-balancing and
- Improved network efficiency

2 big cores, 12 small cores, 4 MCs

Challenge:
low area/timing overhead
to achieve routing flexibility

Logic Based Distributed Routing in NoC, Flich & Duato, Computer Arch Letters, Jan 2008
Fair BW Allocation

Adversarial traffic to Shared (critical) resource

=> network hot spots can lead to unfair resource access

Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks, Lee, Ng, Asanovic, ISCA 2008
Technical Contributors

Mani Azimi, Akhilesh Kumar, Roy Saharoy, Aniruddha Vaidya, Sriram Vangal, Yatin Hoskote, Sanjeev Kumar, Anthony Nguyen, Chris Hughes, Niket Agarwal, and the prototype design team

Ack support of: Mani Azimi, Nitin Borkar
Summary

• Energy efficient interconnects that scale are important for future multi-cores

• Interconnect can play a part in thread scheduling

• Application consolidation in many-core requires close cooperation with run-time.

• Efficient support required for route flexibility
Thanks!

Questions?