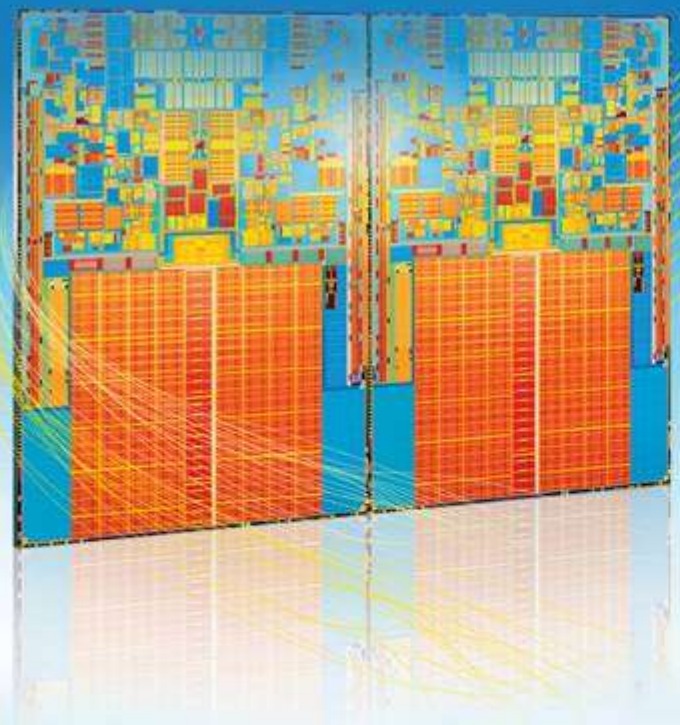
# On Chip Interconnects for TeraScale Computing

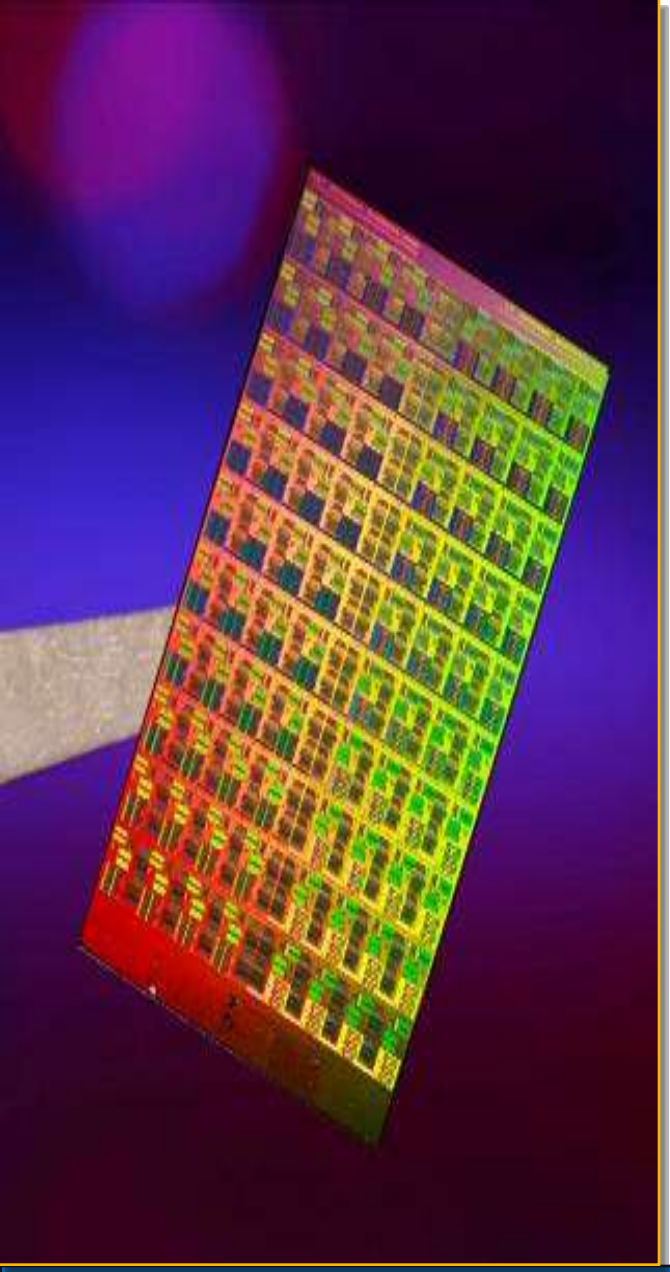


Partha Kundu,
Microprocessor Tech. Labs
Intel Corporation

Computer Arch Day
April 2, 2009
**Princeton University**

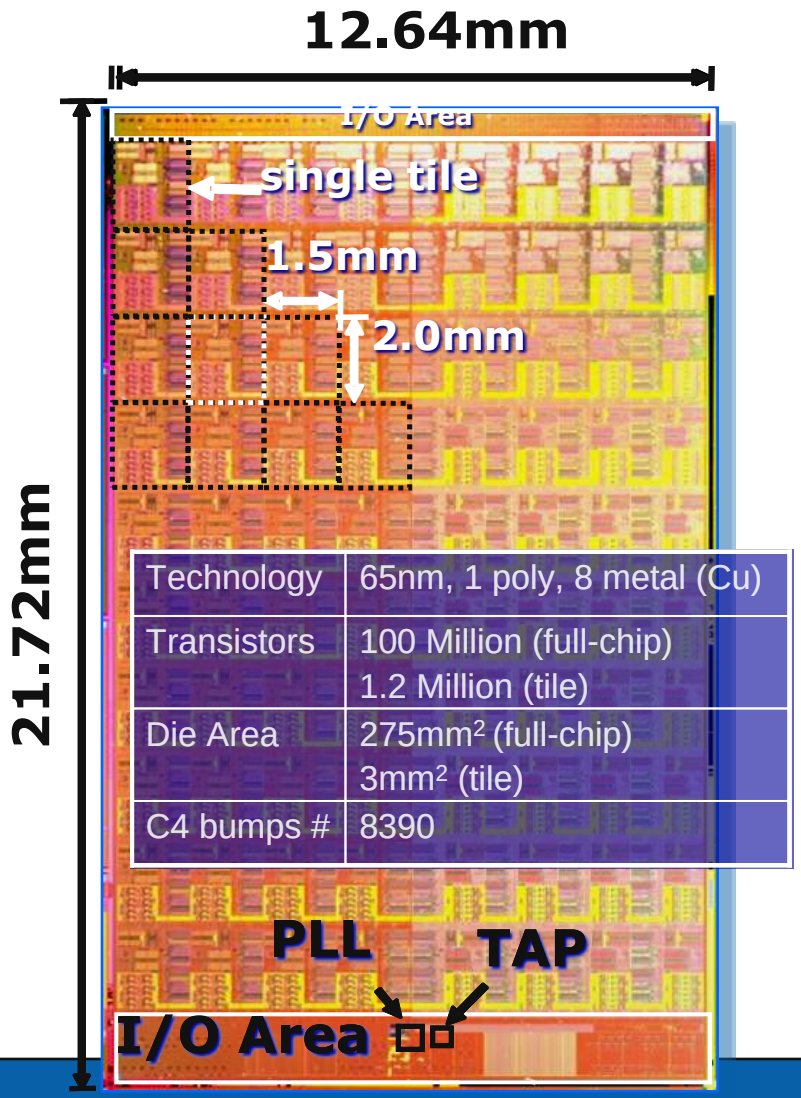
# Overview of Talk

- 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:

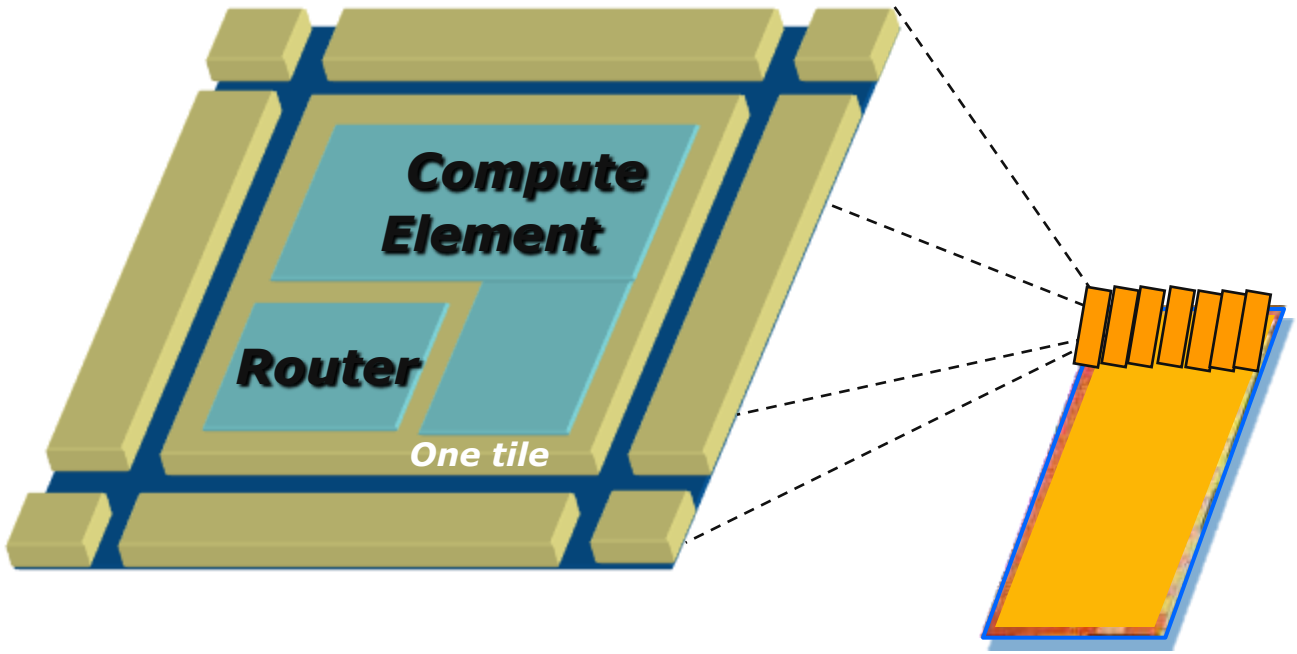
Deliver Tera-scale performance
- Single precision TFLOP at desktop power
- Frequency target 5GHz

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

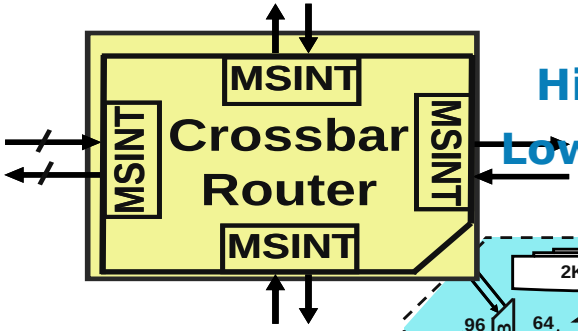
## Assemble & Validate



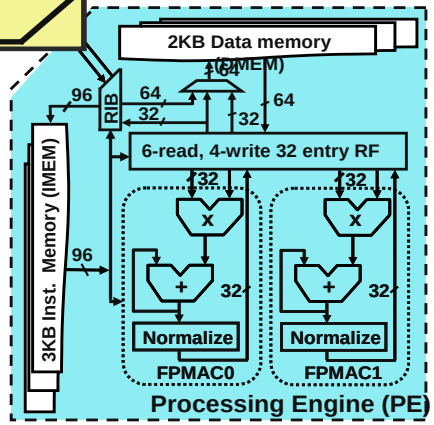

## Step and repeat

# Key Ingredients for Teraflops on a Chip

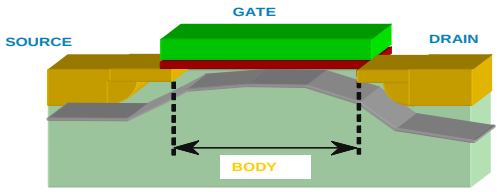
- Core
- Communication
- Technology
- Clocking
- Power management techniques




**High bandwidth**
**Low latency router**




**High performance**
**Dual FPMACs**




**65nm**
**eight metal**
**CMOS**

# Industry leading NoC

## 8x10 mesh

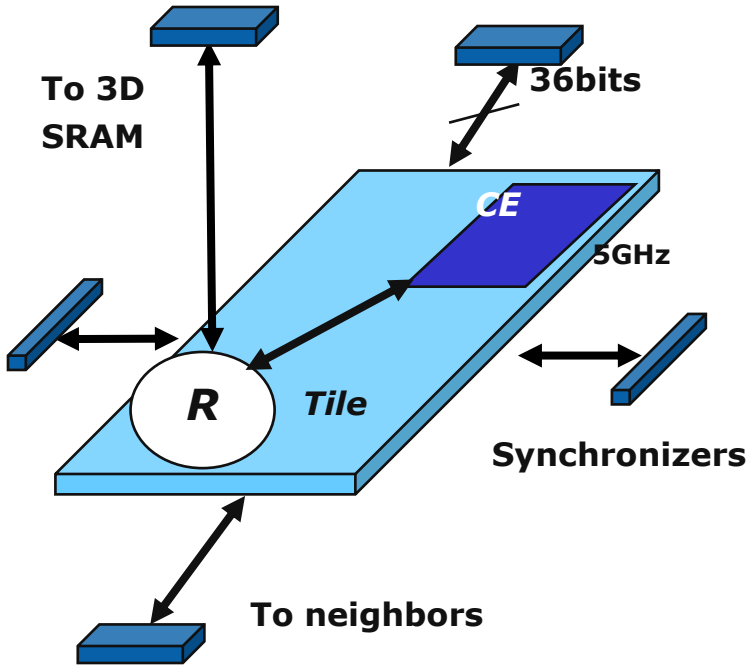
- Bisection bandwidth of 320GB/s
- 40GB/s peak per node
- 4byte bidirectional links
- 6 port non-blocking crossbar
- Crossbar/Switch double-pumped to reduce area

## Router Architecture

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

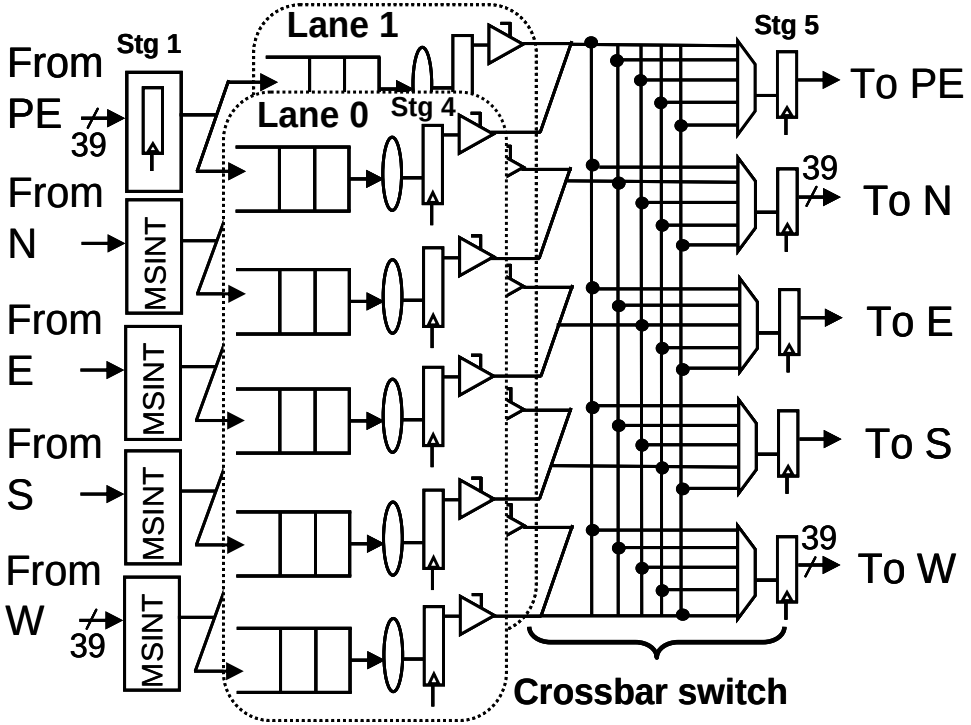
## Meso-chronous clocking

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




**High bandwidth, Low-latency fabric**

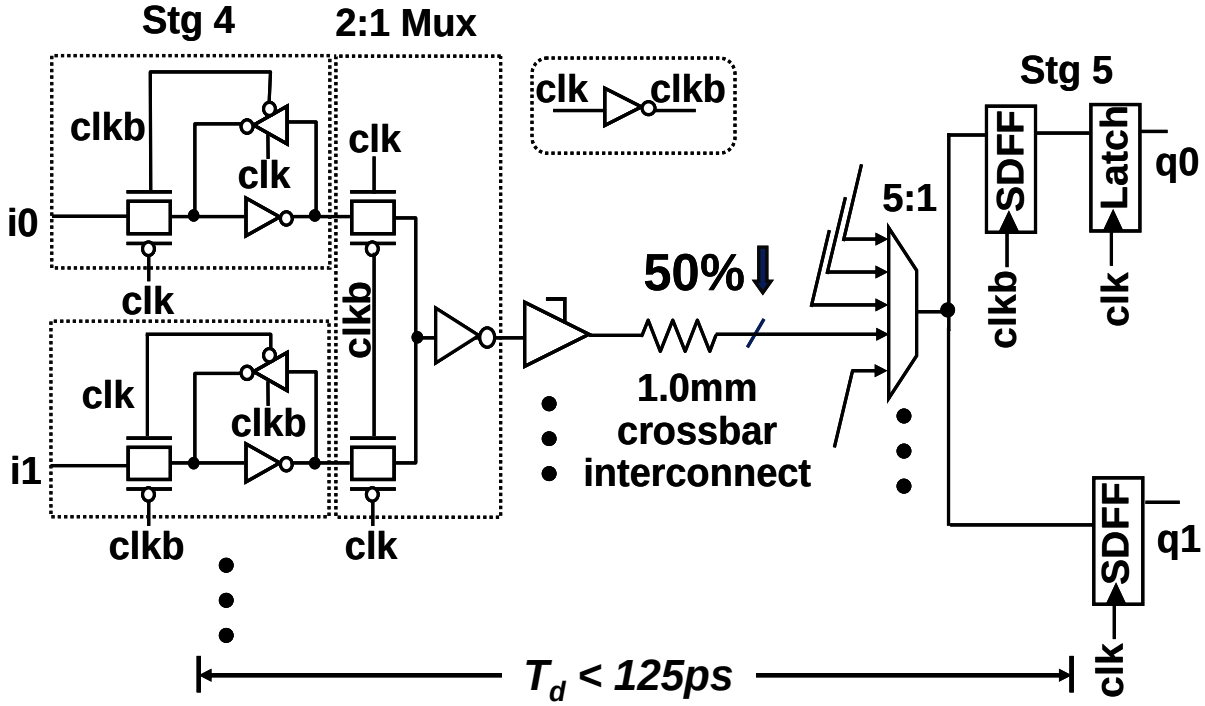
# Router Architecture

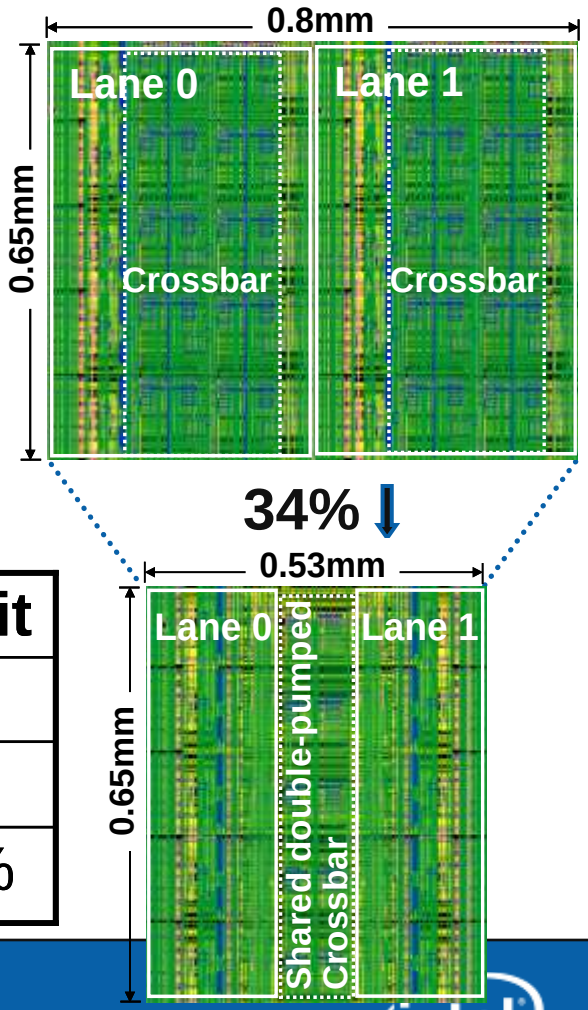



| Buffer Write | Buffer Read | Route Compute | Port/Lane Arb | Switch Traversal | Link Traversal |
|---|---|---|---|---|---|

**Five port, 5-stage, two lane, 16-FLIT FIFO, 100GB/s**
**Shared crossbar architecture, two-stage arbitration**

# Double-pumped Crossbar Router



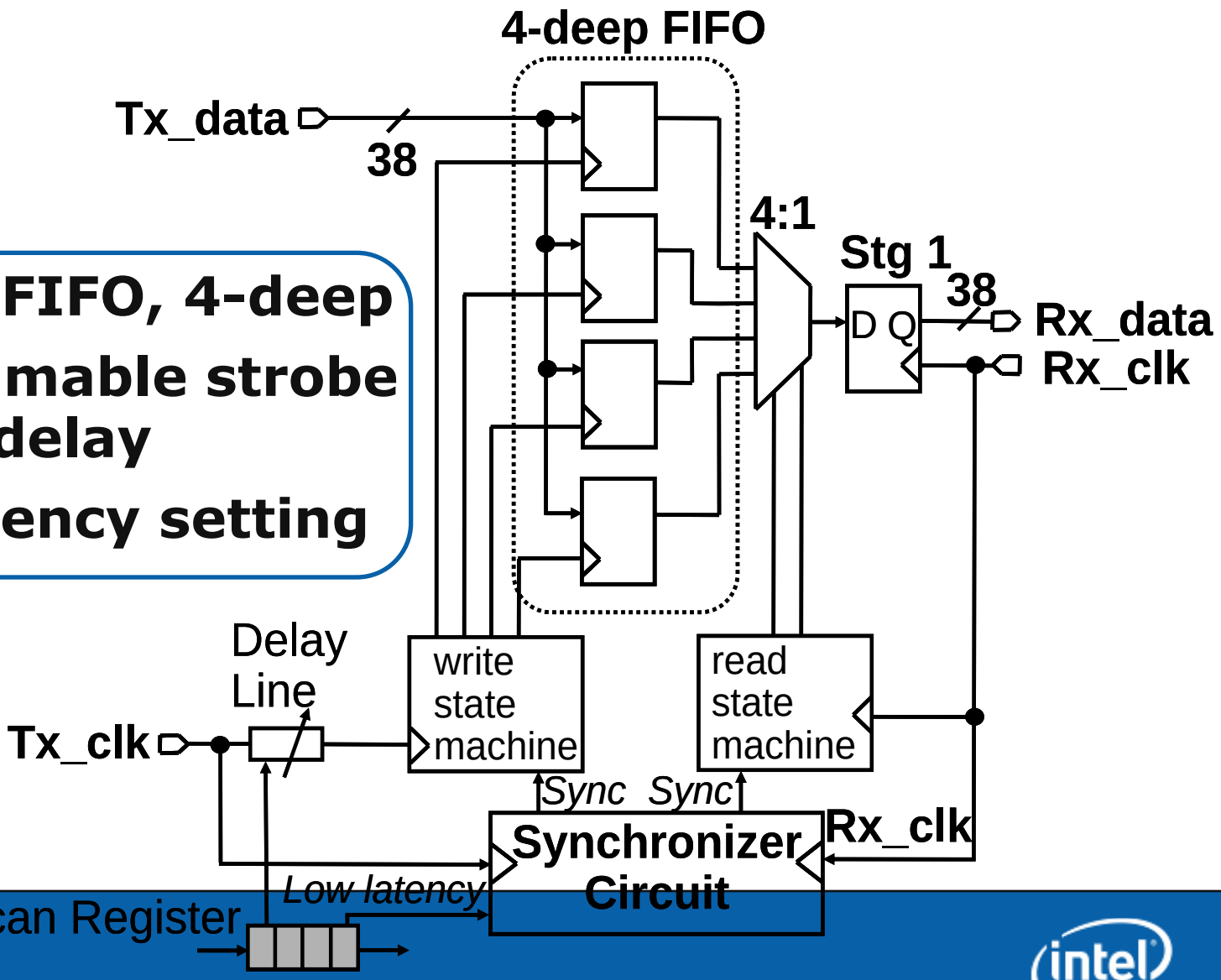

| Router | ISSCC '01 | This Work | Benefit |
|---|---|---|---|
| Transistors | 284K | 210K | 26% |
| Area ($mm^2$) | 0.52 | 0.34 | 34% |
| Latency | 6 | 5 | 16.6% |

# Mesochronous Interface (MSINT)

**Circular FIFO, 4-deep**

**Programmable strobe delay**

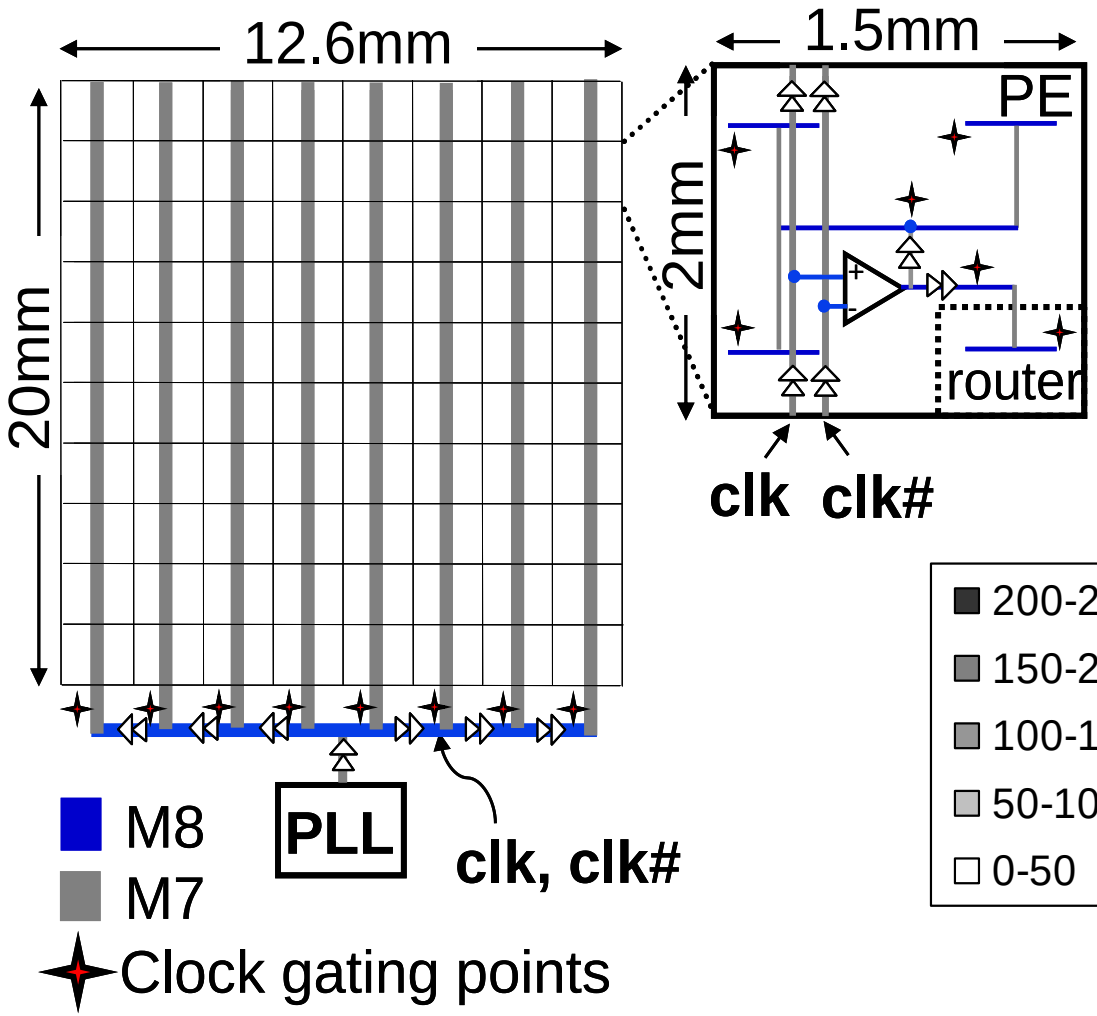
**Low-latency setting**





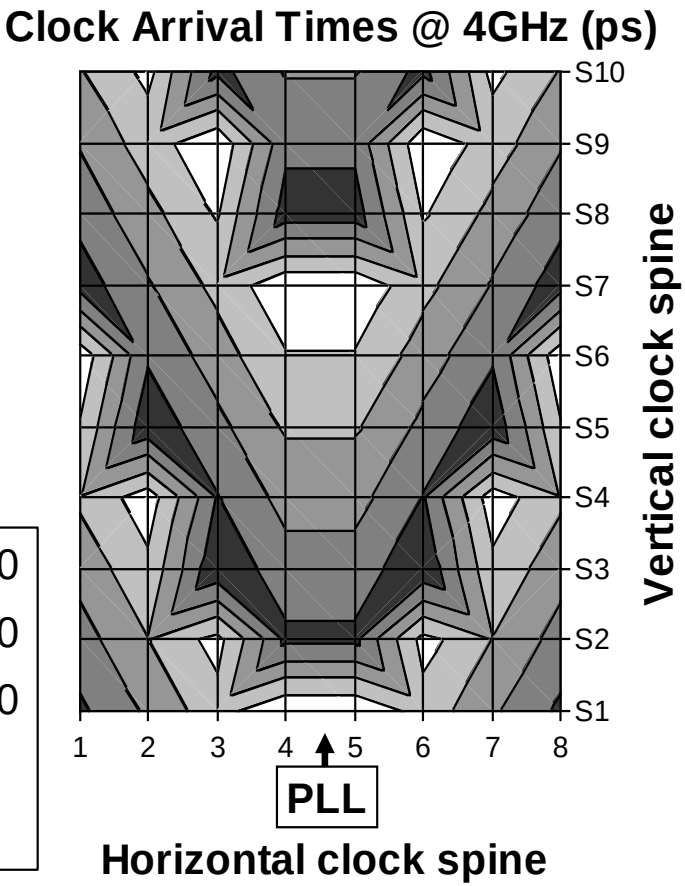

10

# Low Power Clock Distribution










Global mesochronous clocking, extensive clock gating



11

# 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**

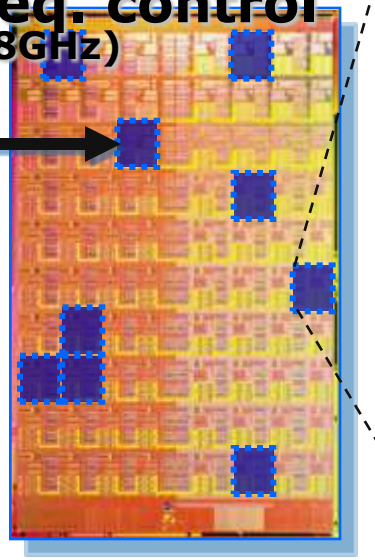
**STANDBY:**

- **Memory retains data**
- **50% less power/tile**

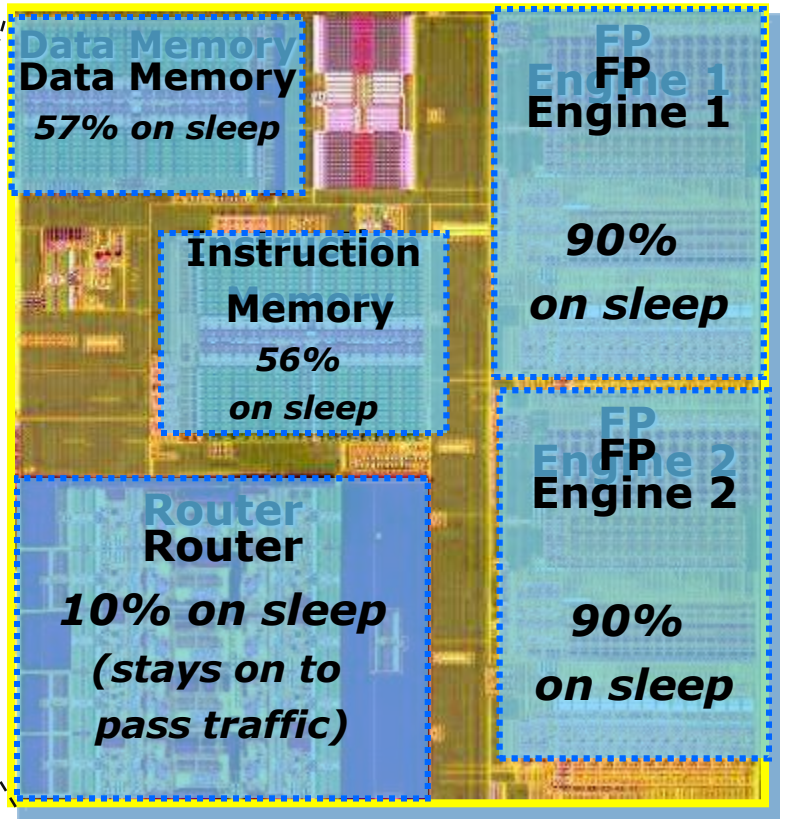
**FULL SLEEP:**

- **Memories fully off**

- **80% less power/tile**



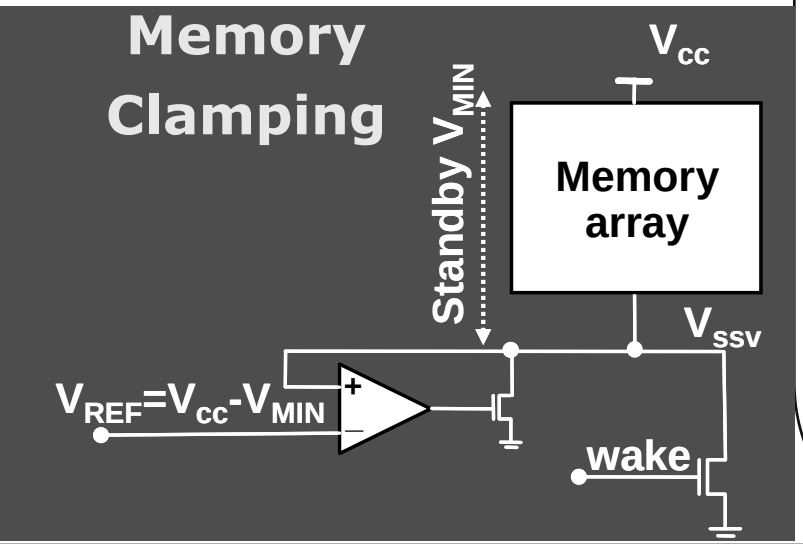
***21 sleep regions per tile (not all shown)***

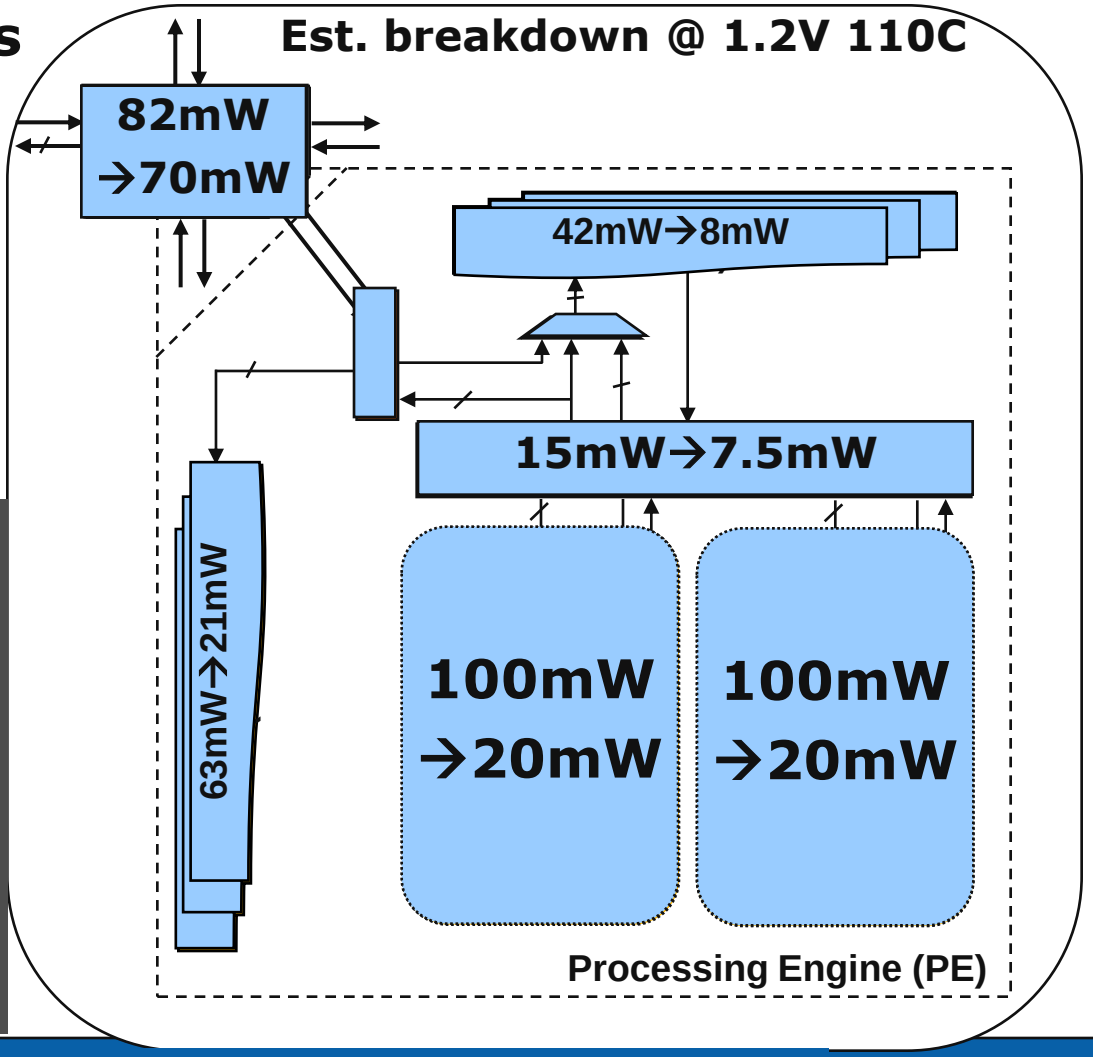



**1680 dynamic power gating regions on-chip**

# Leakage Savings

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




2X-5X leakage power reduction



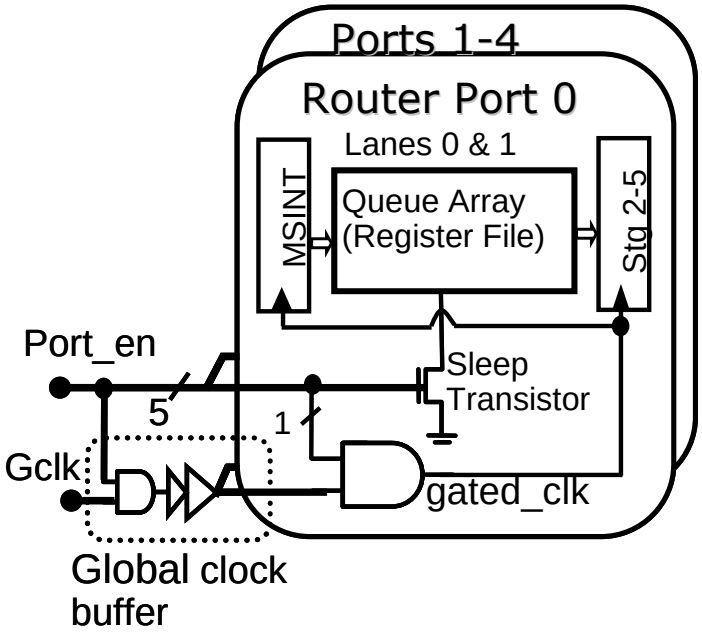
13

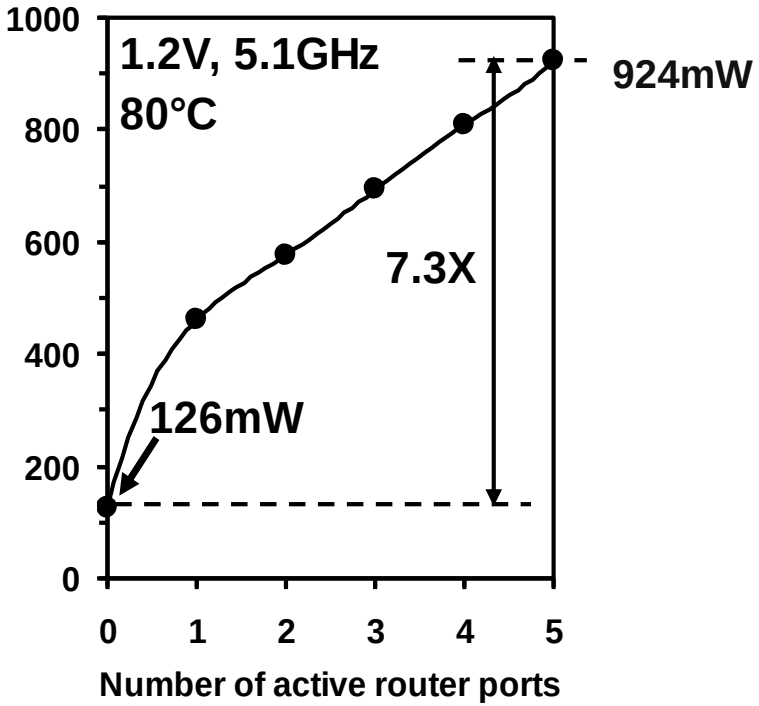
# Router Power

Activity based power management

Individual port enables

- Queues on sleep and clock gated when port idle

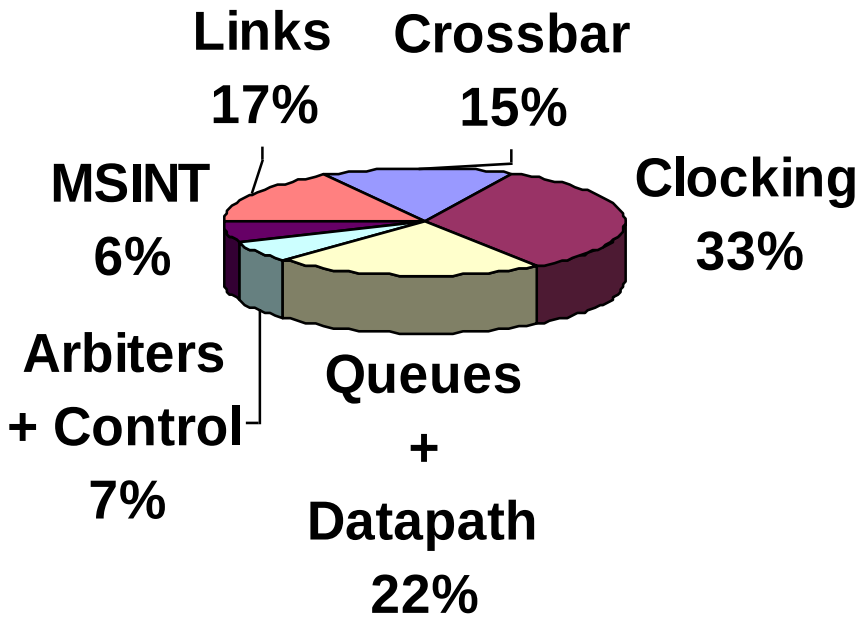



**Measured 7X power reduction for idle routers**

# Estimated Power Breakdown

## Communication Power




**4GHz, 1.2V, 110°C**

- **Significant (>80%) power in router compared to wires**
- **Router power primarily in Crossbar and Buffers**
- **Clocking power w/fwded clock can be expensive**

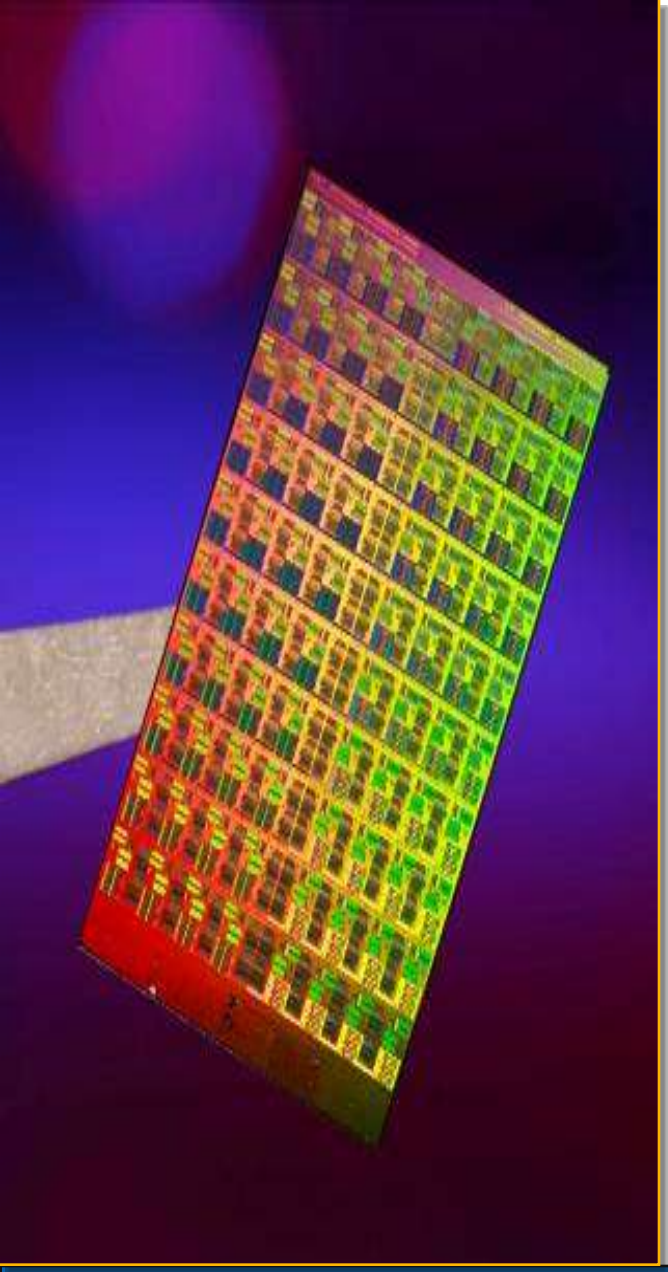


15

# Energy efficiency of Interconnection networks

- Topology work :
  - 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
  - Minimize dynamic buffer usage :
    Express Virtual Channels, Kumar et al, ISCA 2007
  - 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

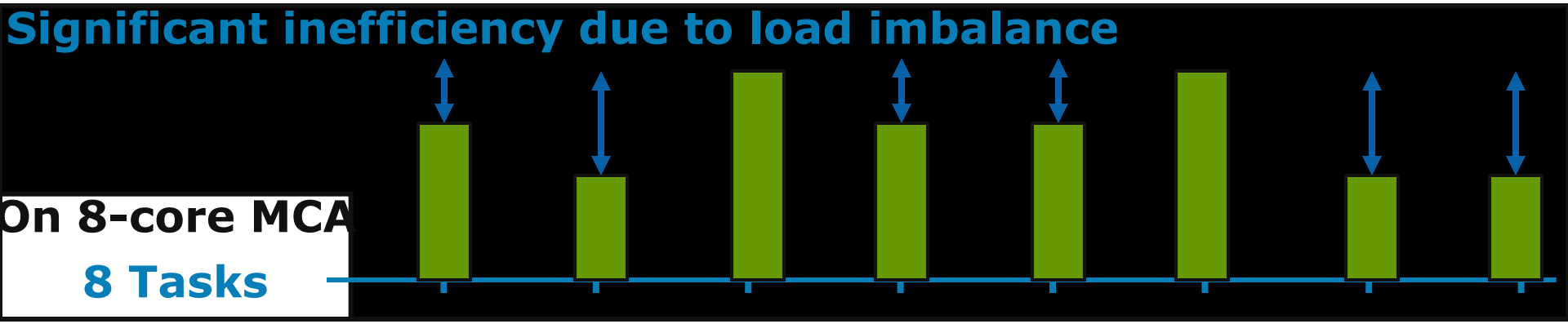
➢ Sometimes even nested

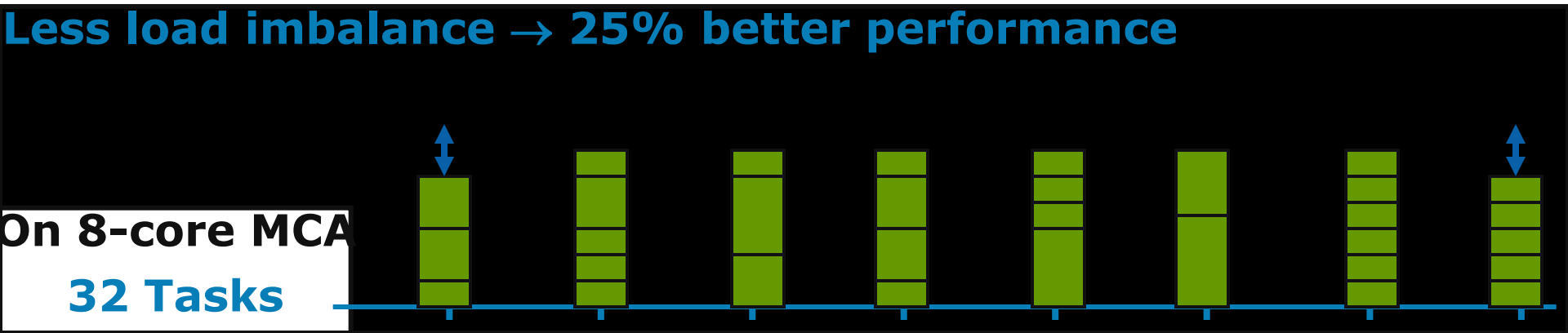
Need to support them at the same time

# Asymmetry

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

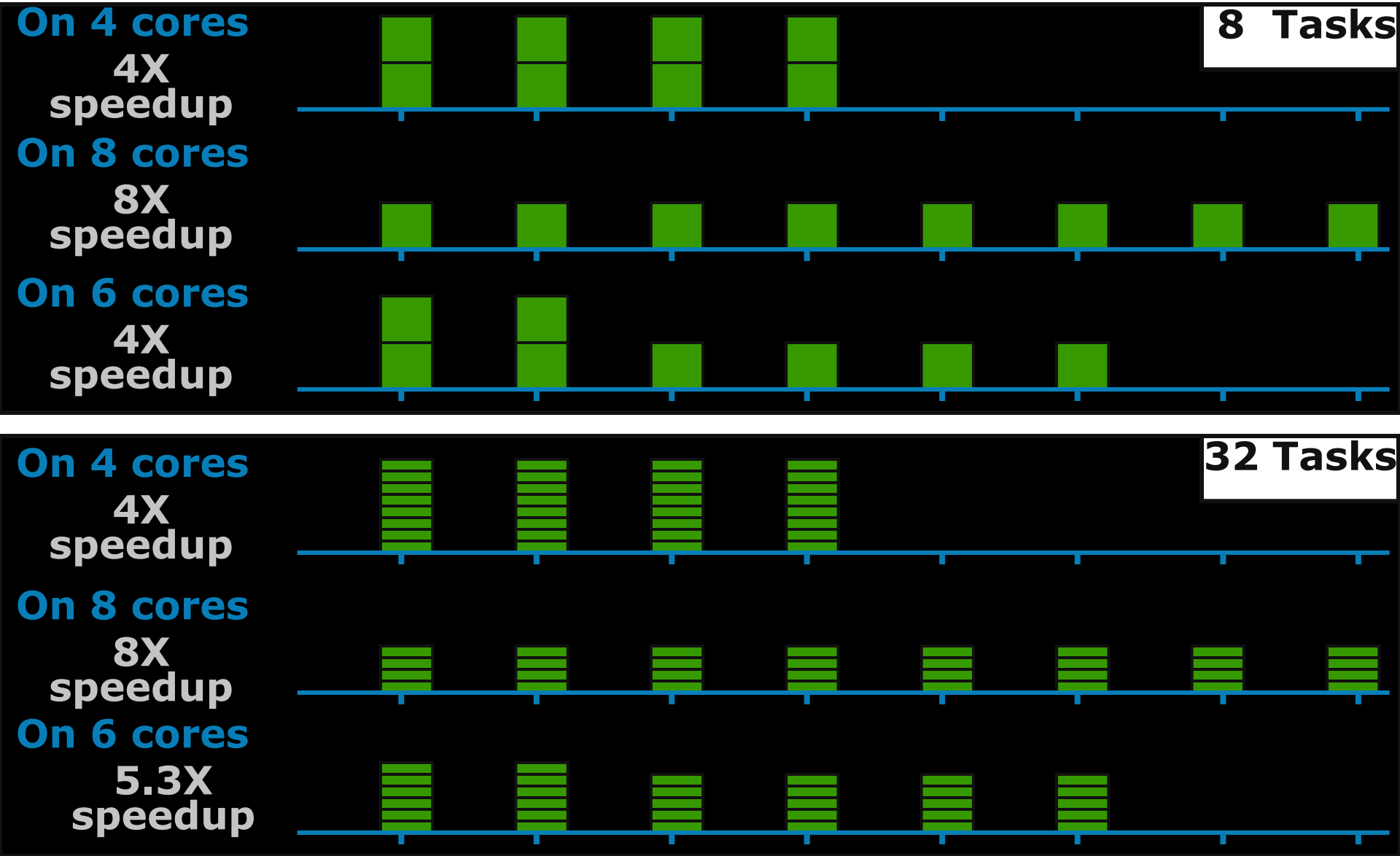
fine-grained parallelism can mitigate performance asymmetry



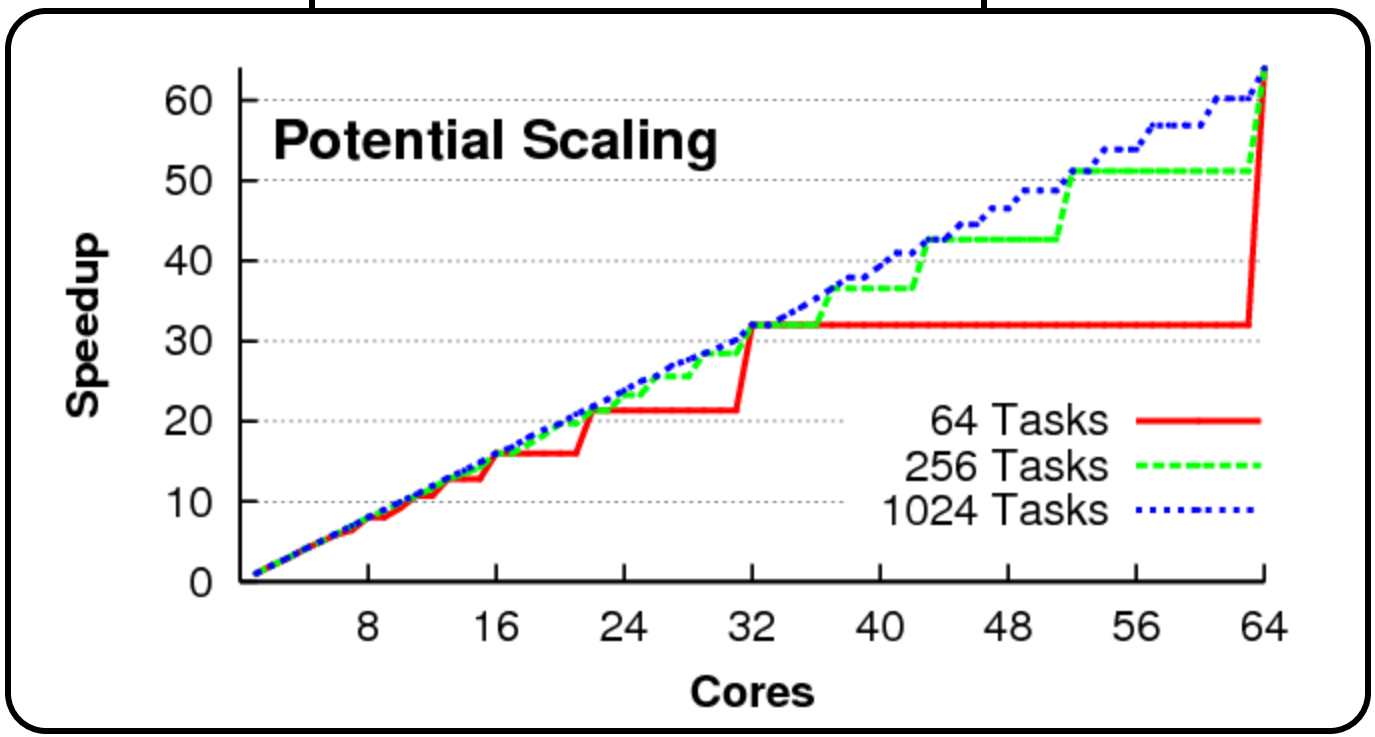
# Platform Portability

## Consider a 8-core MCA



# Platform Portability Cont'd

**MCA with 64 cores**




Requires *finer-granularity* parallelism

Even an order of magnitude



21

# 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

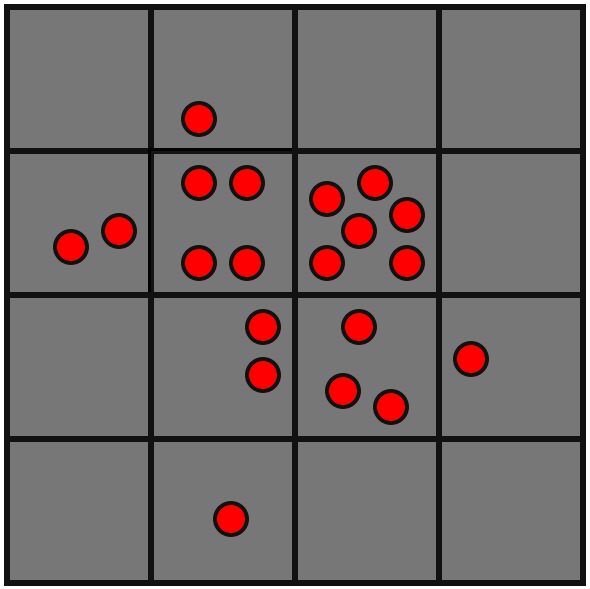
**Carbon : Architectural Support for Fine-Grained Parallelism, Kumar et al**
**ISCA 2007**


22

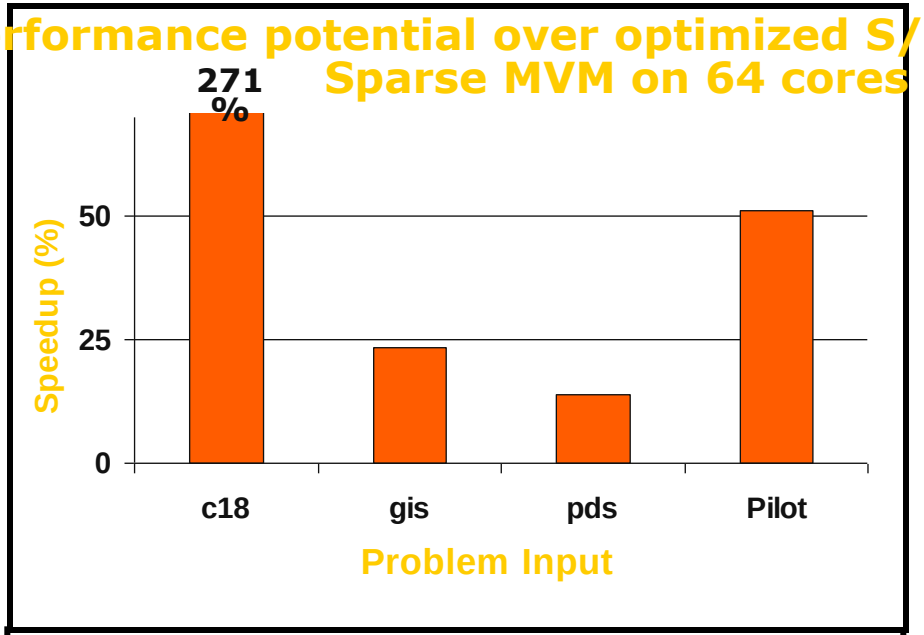
# Loop-Level parallelism

Most common form of parallelism supported by OpenMP, NESL

Requires dynamic load balancing



**Computation per iteration (tile) can vary dramatically**




**Significant performance potential**

- **13-271%**

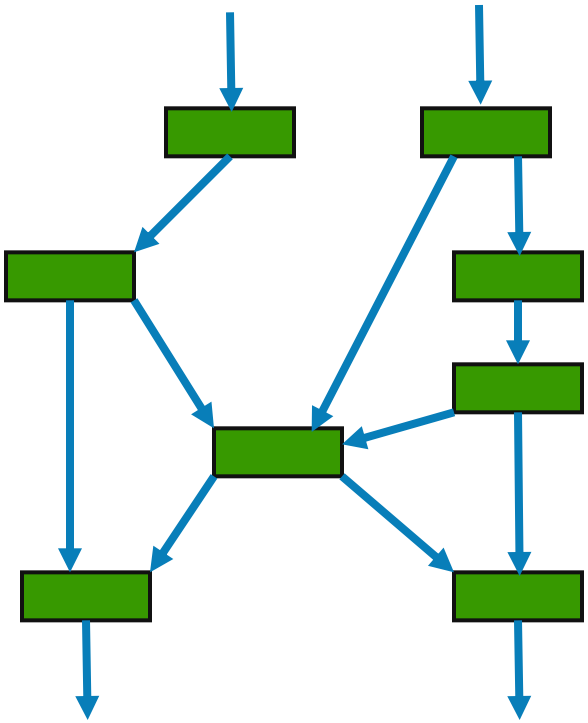


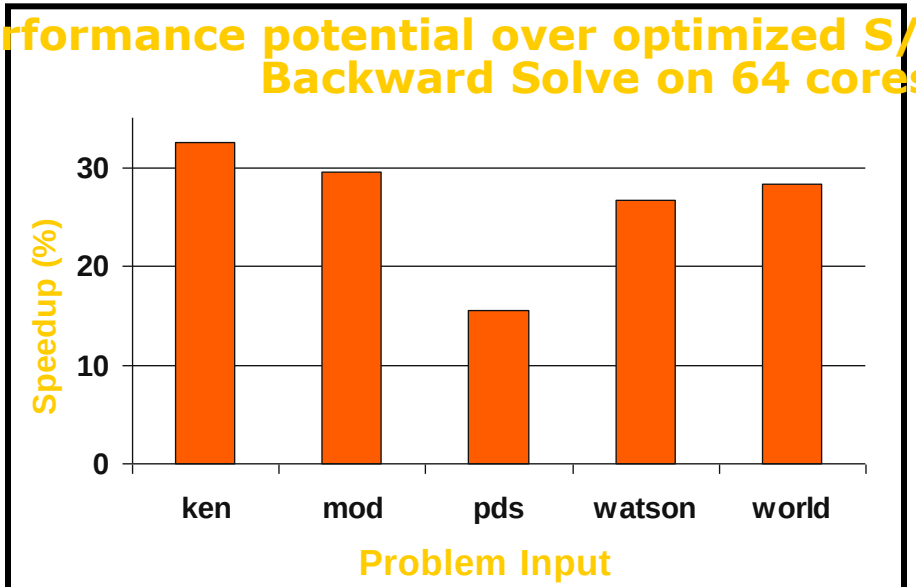
23

# Task-Level Parallelism

Irregular structured parallelism

- Trees to complex dependency graphs

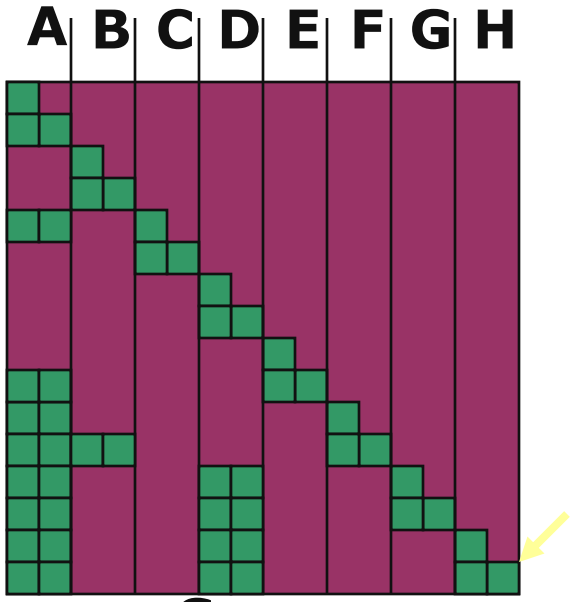





**Significant performance potential**

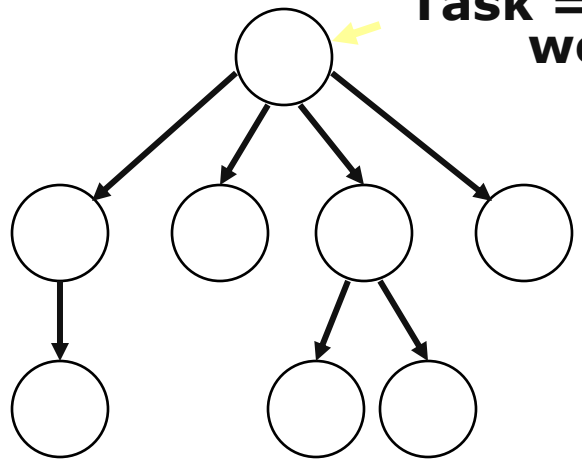
- **15-32% for Backward solve**
- **Up to 241% for Canny**

# Typical Parallel Program




Sparse matrix

Non-zero



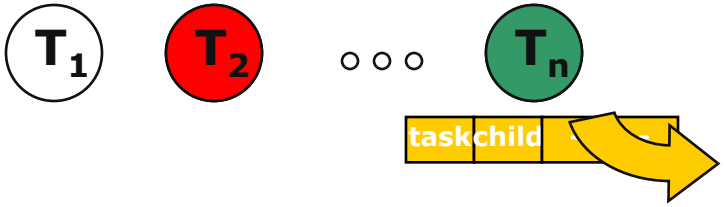

Task dependence graph

**Example module: forward solve**

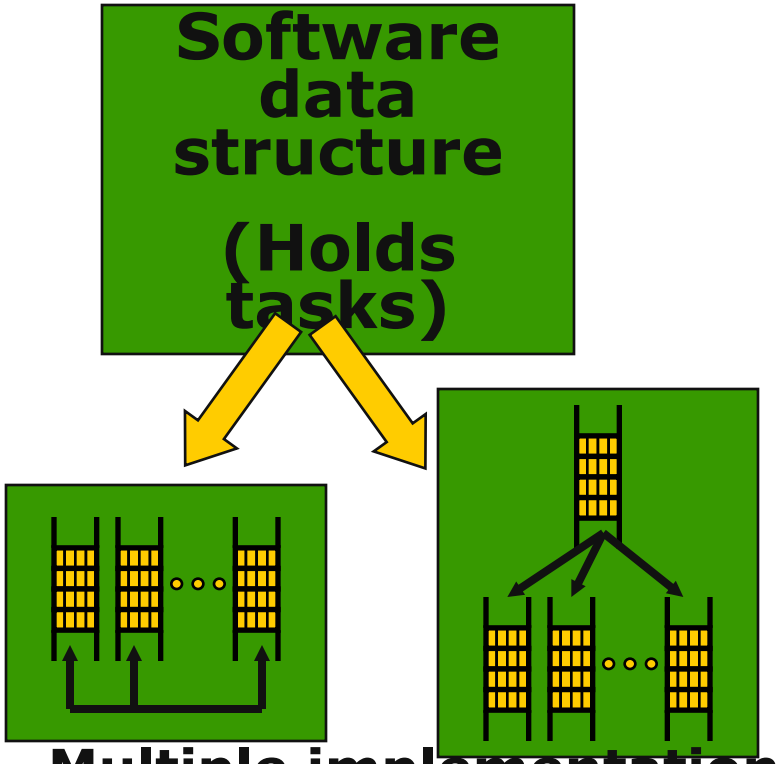
**Use task parallelism**

```
void task(node)
{
    Do(node);

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

# Typical Parallel Run-Time System




```
void task(node)
{
    Do(node);

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


Run-time system creates tuple to represent task

- GPUs, conventional S/W (e.g., TBB) do this today


26

# Need for Hardware Acceleration

Software "Enqueue" & "Dequeue" is slow
- Serial access to head/tail

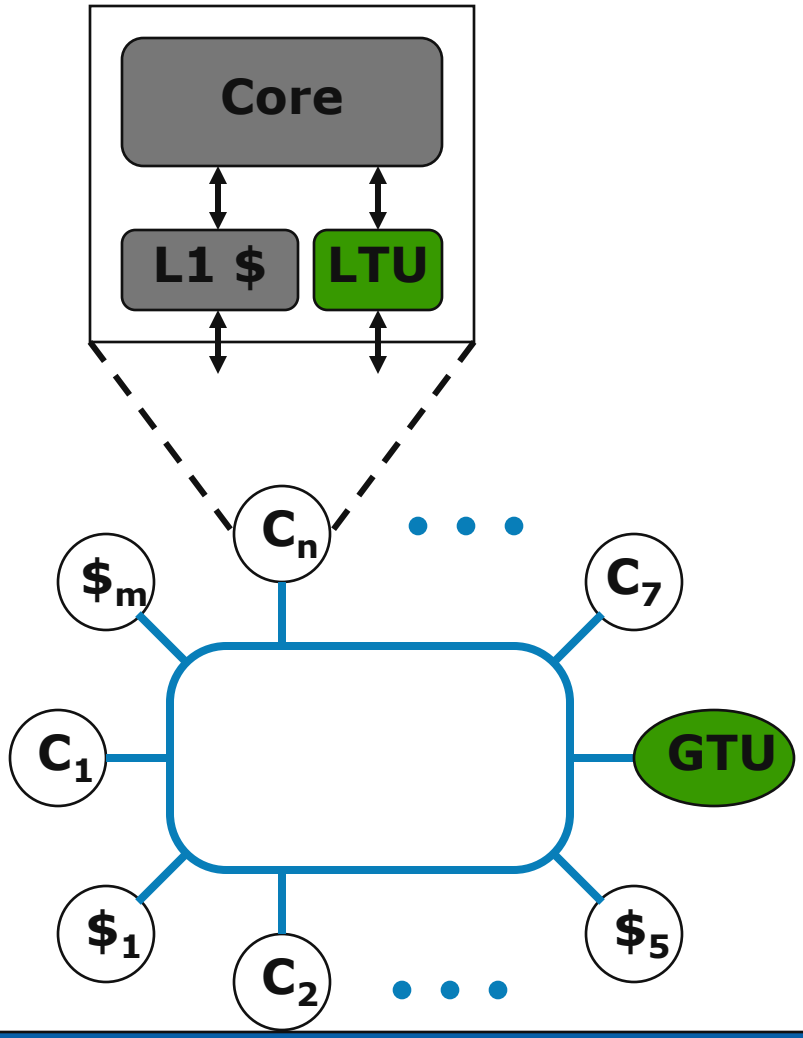
Additional overhead for smart ordering of tasks
- Placement, Cache/data locality, process prioritization, etc.

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)

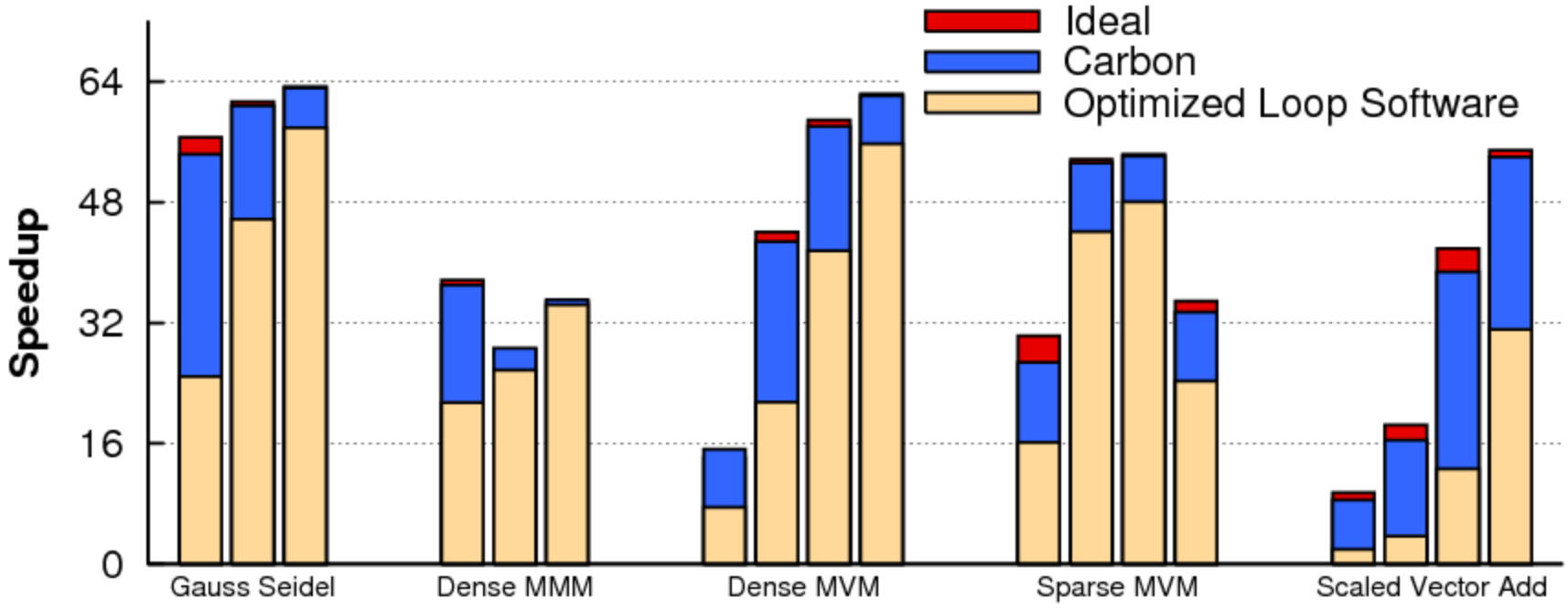
**Prefetches and buffers tasks**

## Global Task Unit (GTU)

**Task pool storage**


28

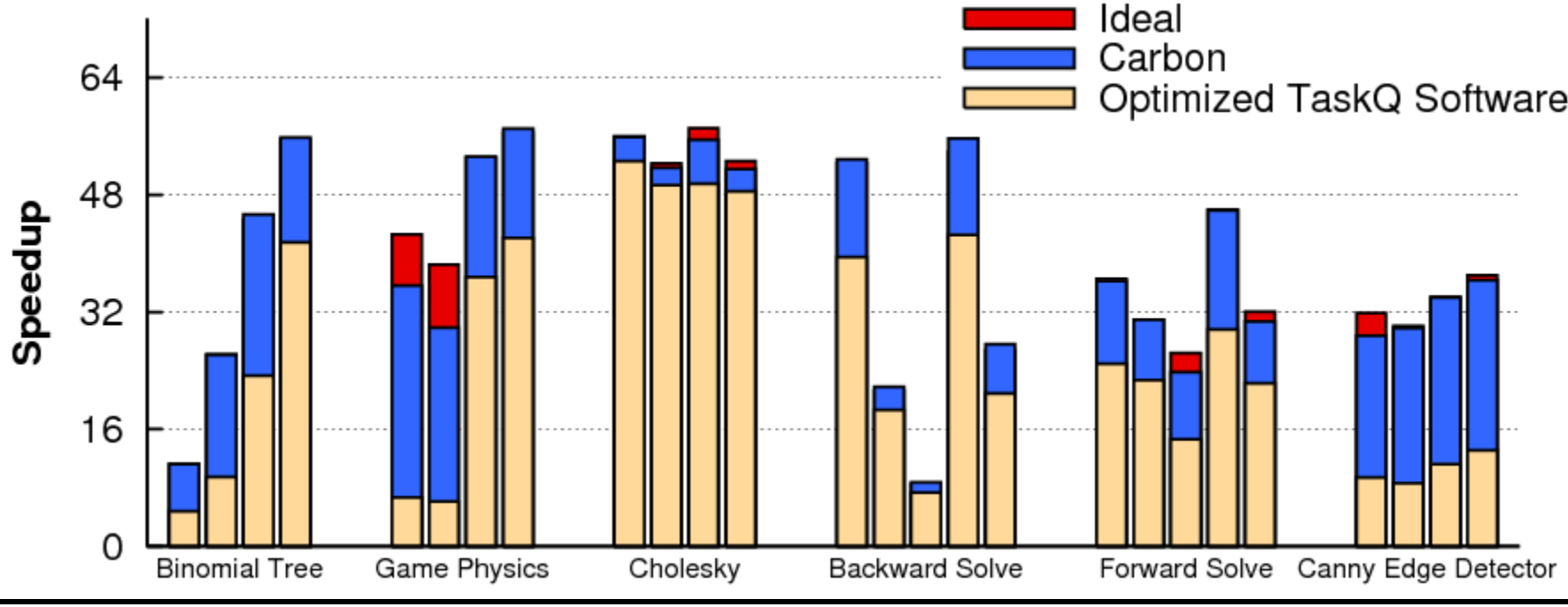
# Performance: Loop/Vector Parallelism




- **Significant performance benefit over optimized S/W**
  - **88% better on average**
- **Similar performance to "Ideal"**
  - **3% worse on average**


29

# Performance: Task-Level Parallelism



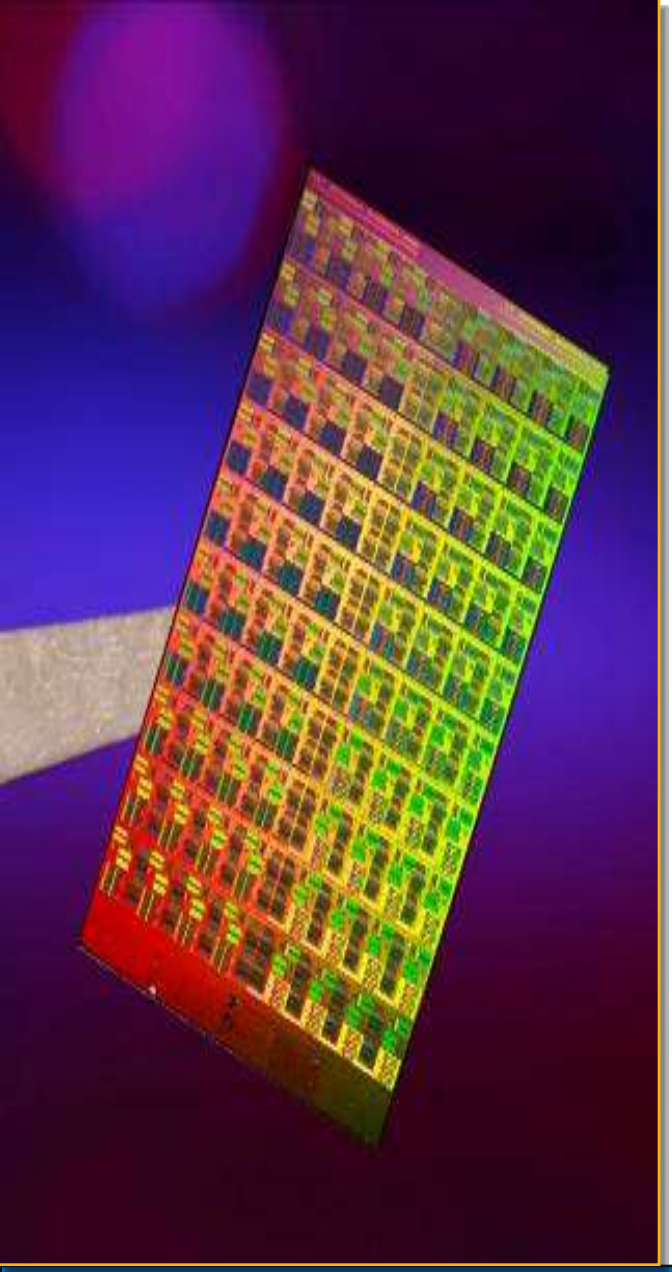

- **Significant performance benefit over optimized S/W**
  - **98% better on average**
- **Similar performance to "Ideal"**
  - **2.7% worse on average**


30

# Task Queuing in the Interconnect

- Arrangement of GTU and LTU suffices for apps studied

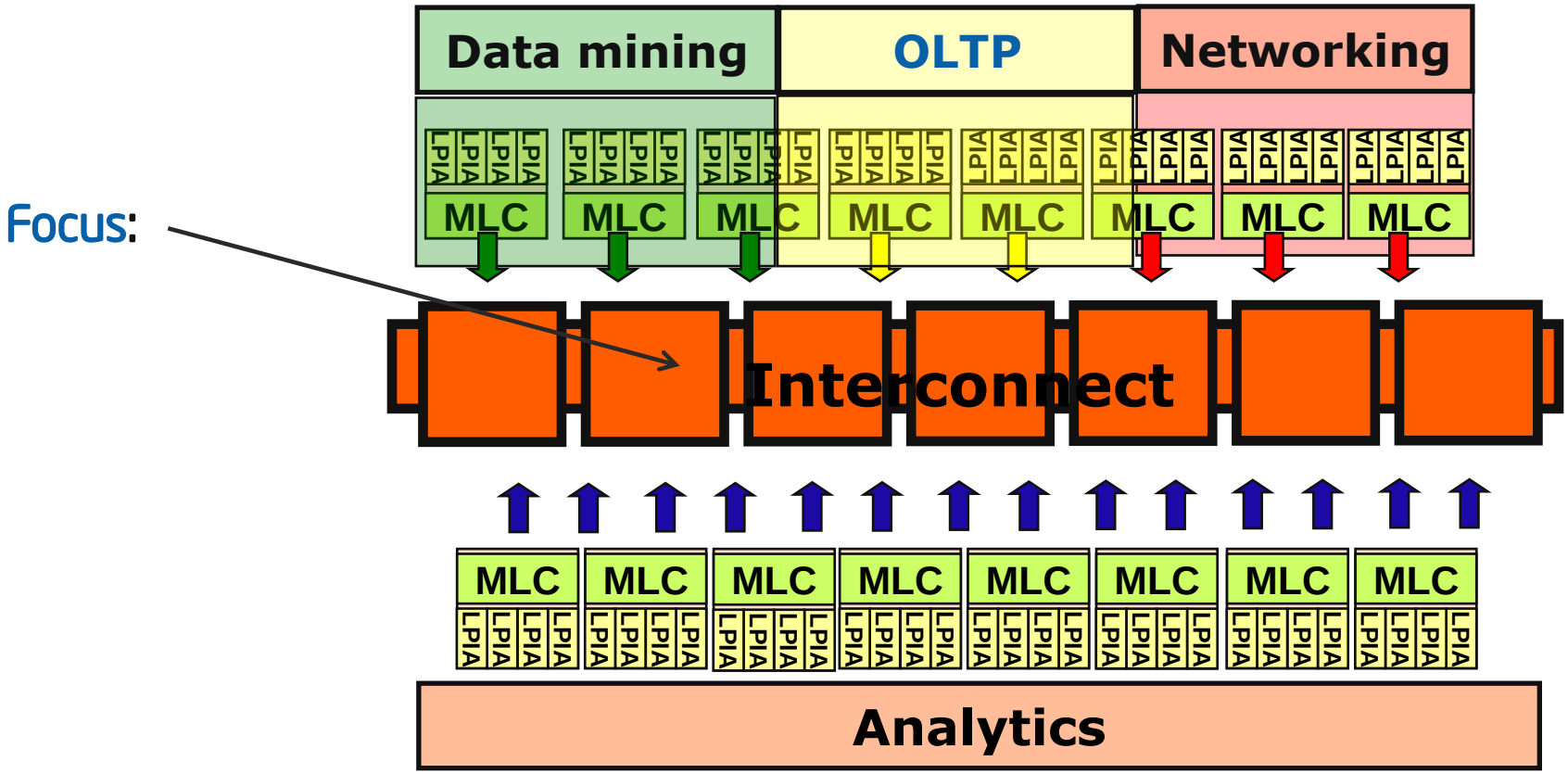
- But, long vectors running on MCA can be tricky
  - requires dynamic resource discovery
  - Vector length breaks
    *Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow*, Fung et al (Micro '07)

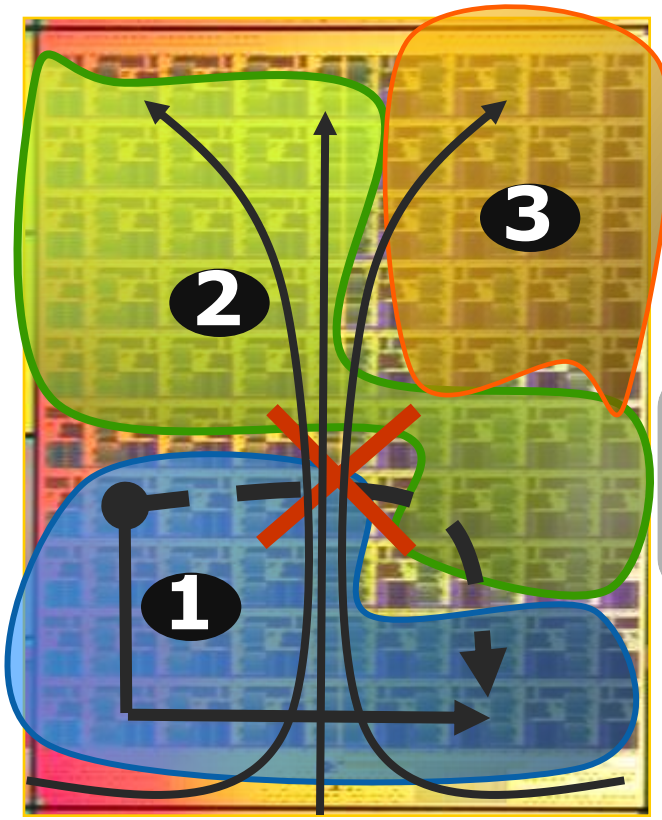


# Partitioning, Isolation and QoS in Interconnects

# Virtualization and Partitioning of on-chip Resources

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





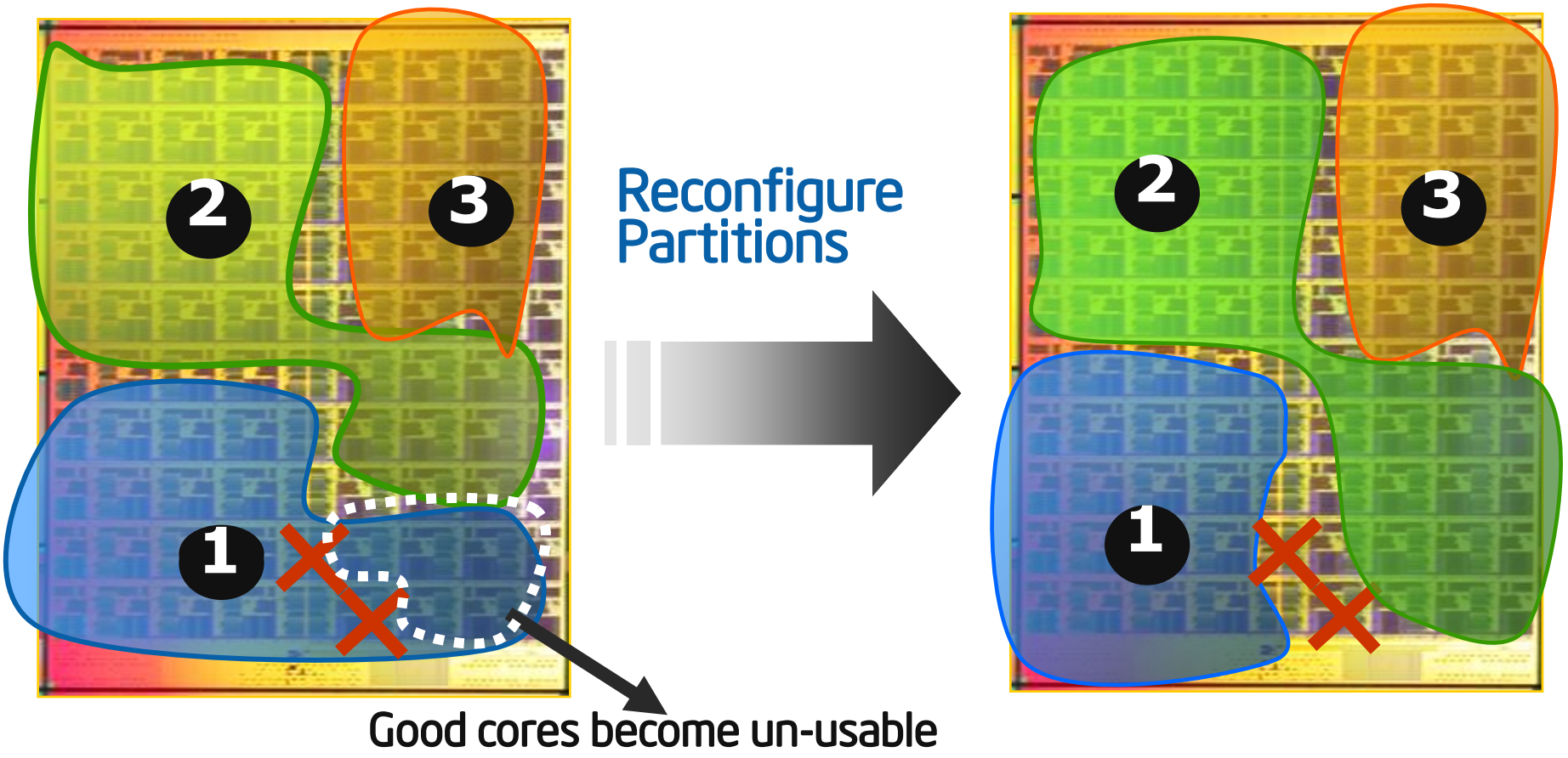

Domain isolation for performance and security

Allow “arbitrary” shaped domains

Shared Channel Reservation

**Application Isolation enforced by Interconnect**

# Isolation with Fault Tolerance

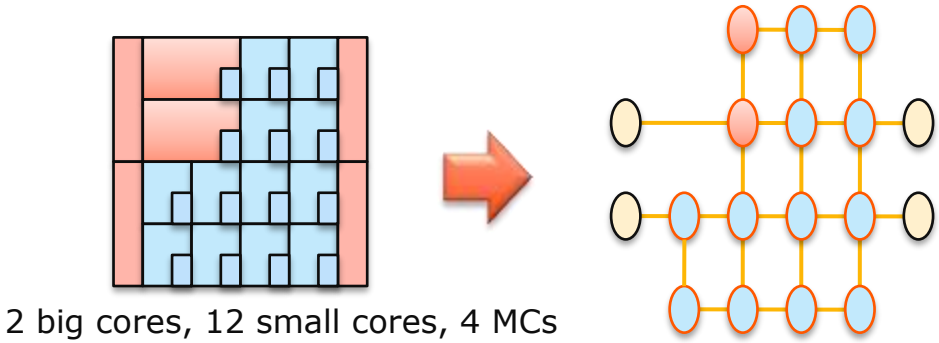



Enable Fault Discovery & Repartitioning

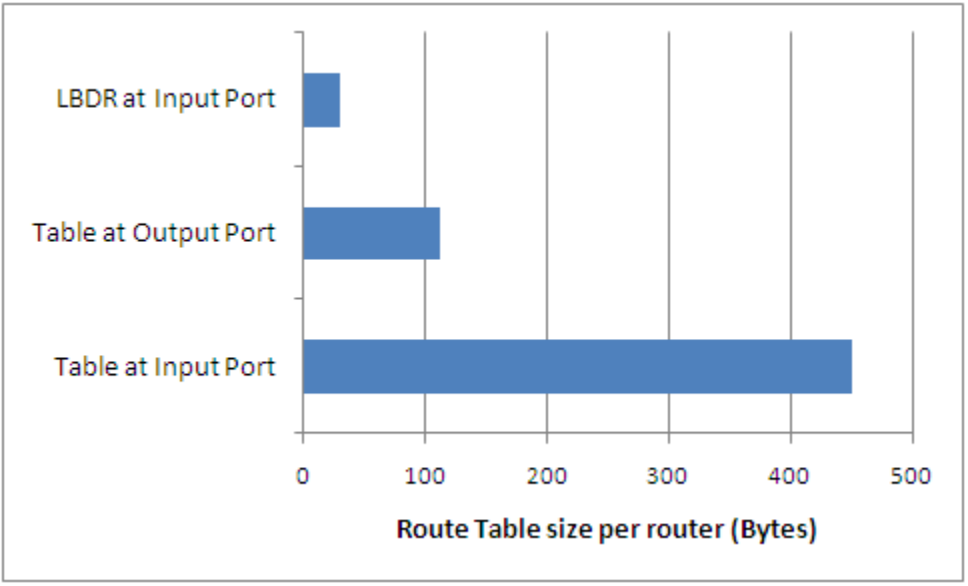
# Route Flexibility

**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

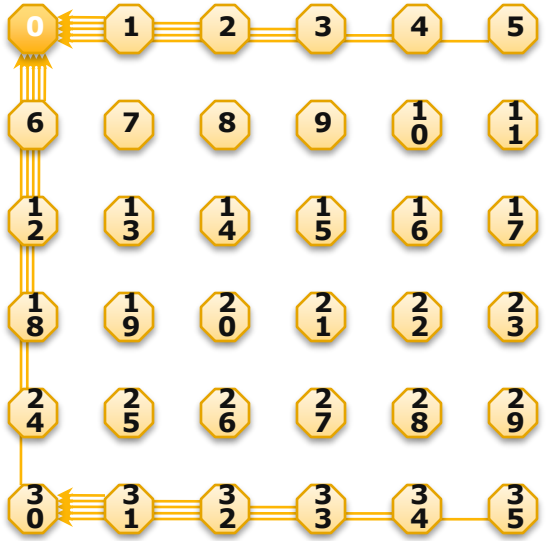
Logic Based Distributed Routing in NoC, Flich & Duato, Computer Arch Letters, Jan 2008

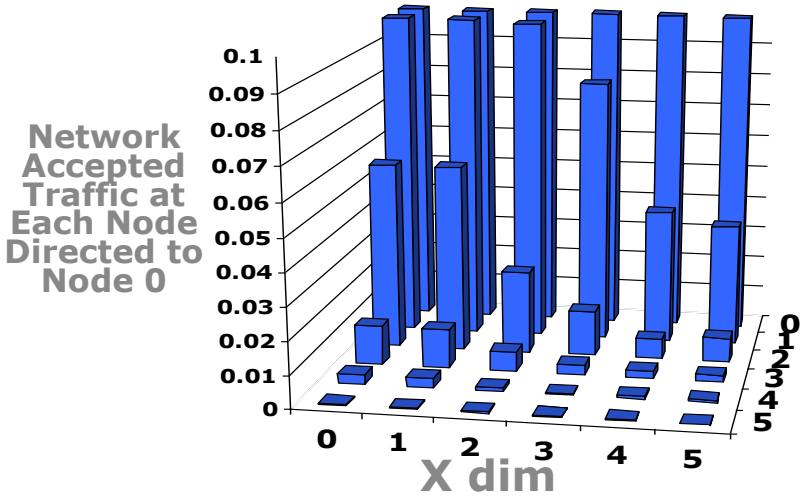
# Fair BW Allocation

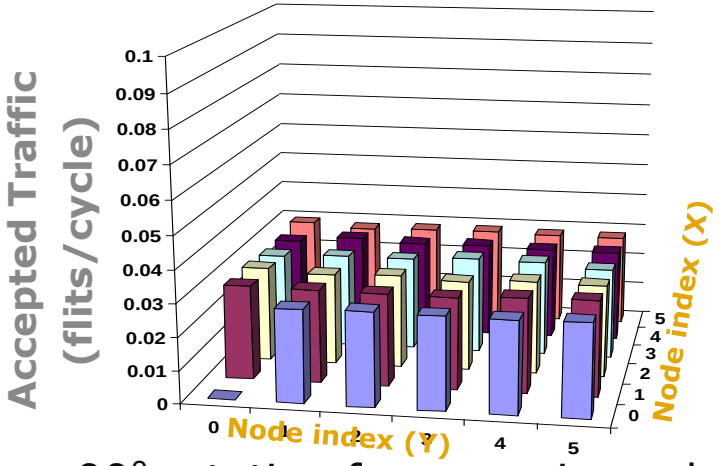
Adversarial traffic to Shared (critical) resource

=> 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




$90^0$ rotation from previous plot

# 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?