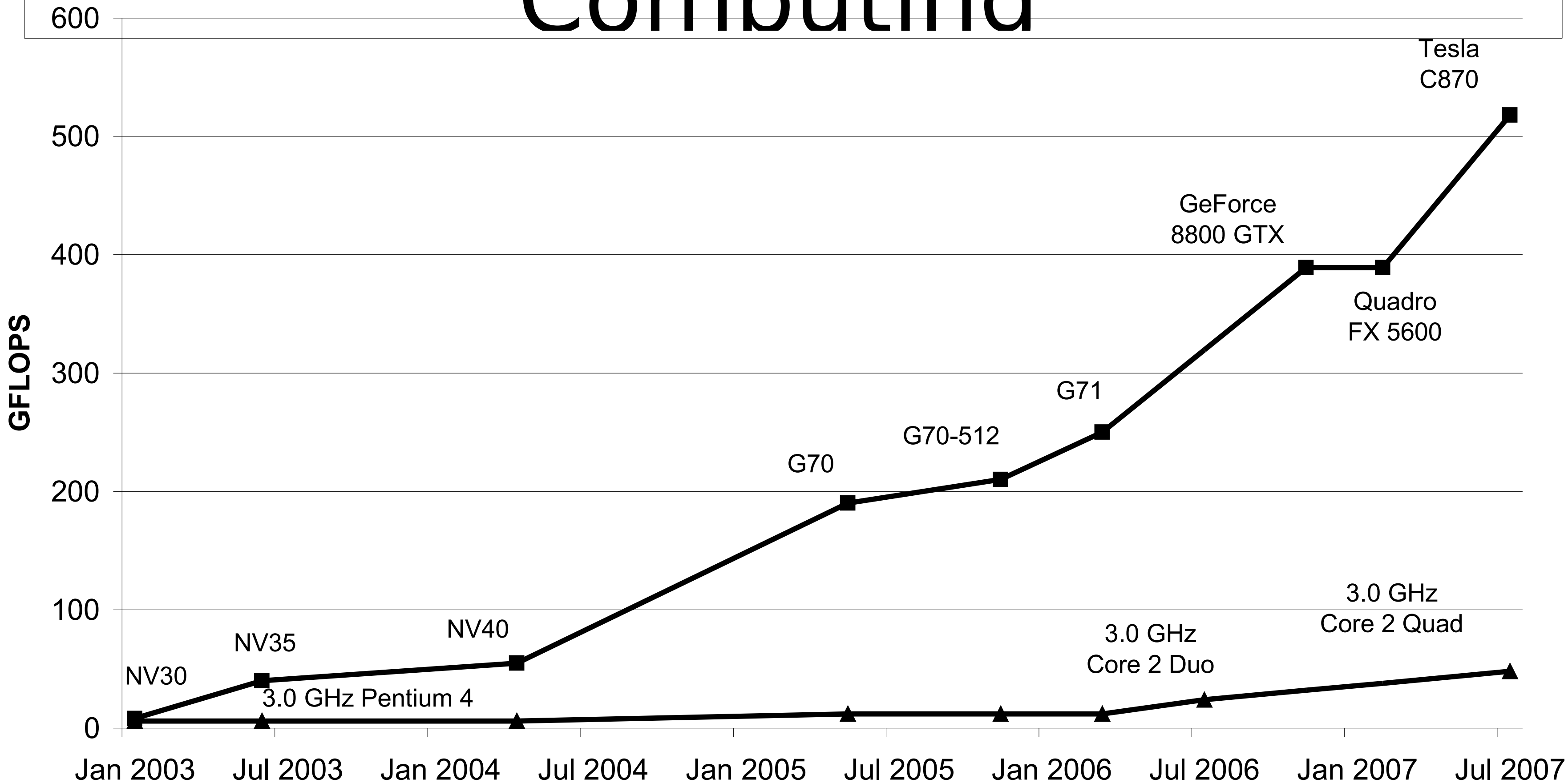
Rigel:
A Scalable Architecture for 1000+

“CoreKong”

Click to edit Master subtitle style

Prof. Sanjay J. Patel
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

Emergent Phenomenon: GPU Computing



S. Green, “GPU Physics,” SIGGRAPH 2007 GPGPU Course.
http://www.gpgpu.org/s2007/slides/15-GPGPU-physics.pdf

Accelerated Computing: Today

Programmable data parallel accelerator: HW entity designed to provide advantages for a class of apps including: **higher performance**, **lower power**, or lower unit cost relative to a general-purpose CPU.

- **Contemporary Accelerators:** GPUs, Cell, Larrabee
- **Some Challenges**:
  1. Inflexible parallel programming models
  2. Lack of conventional memory model
  3. Irregular parallel apps difficult to scale
  4. Significant effort in optimizing code
- **Effect on Development**: Unattractive time to solution

Rigel ILLINOIS UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Accelerated Computing: Tomorrow

- **Challenge**: Performance vs. app development effort
- **Accelerator Trend**: Increasing programmability
  - While still providing performance

Spectrum of Programmable Accelerators

FPGA **GPU, Cell** **Rigel, x86 multicore**

(less) ease of Programmability (more)

Accelerated Computing: Metrics

- **FLOPS/mm2 (throughput)**
- **FLOPS/Watt (power)**
- **FLOPS/Programming Effort**

- SIMD/Vector
- Threading
- Memory management
- Programming model support
- Performance portability

Project Orion

Applications, Programming Environments, and Architecture for 1000-core Parallelism

Architecture

Programming Environments

IMPACT 2

Applications

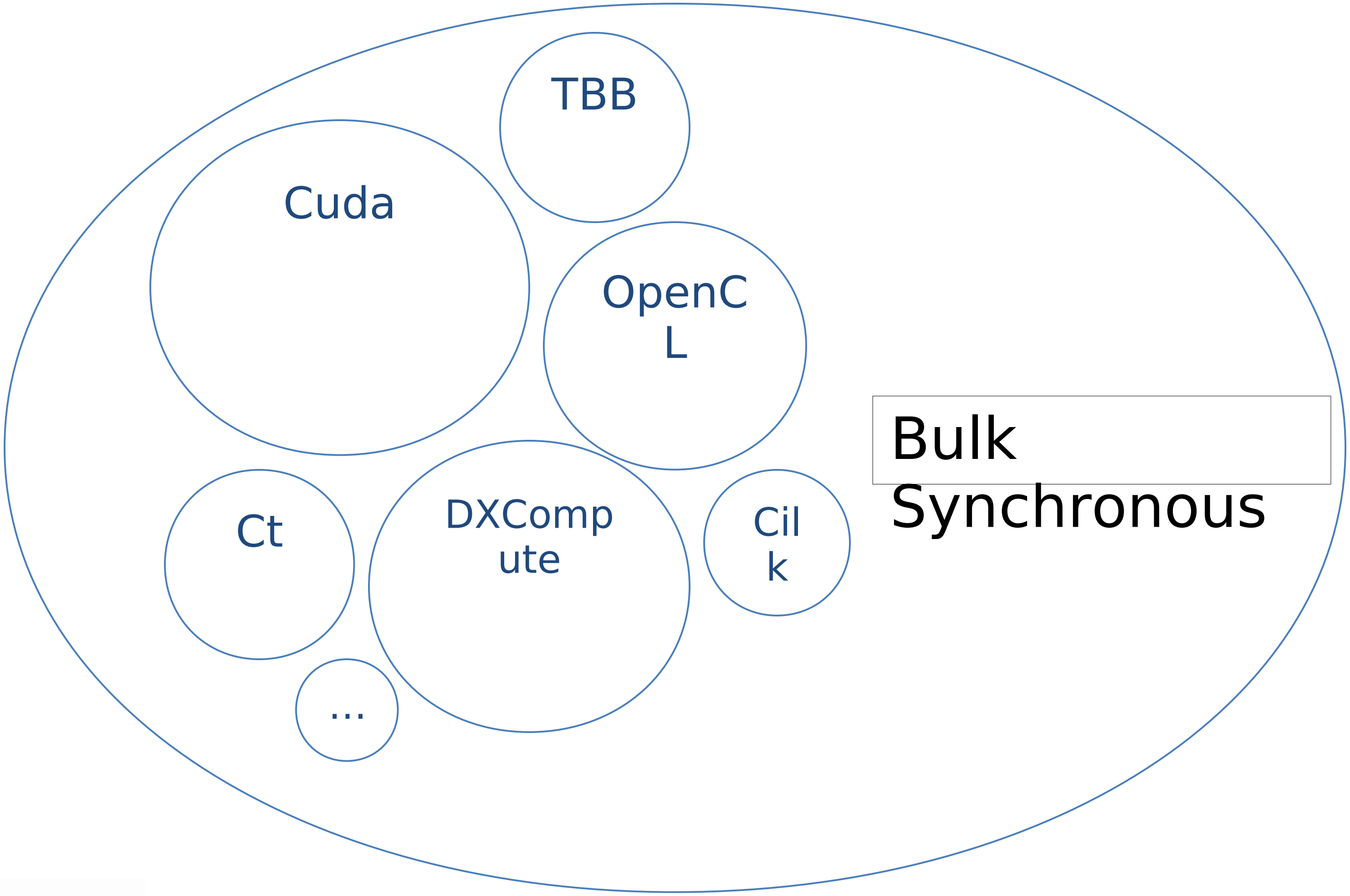
Illinois Image Formation and Processing

Orion

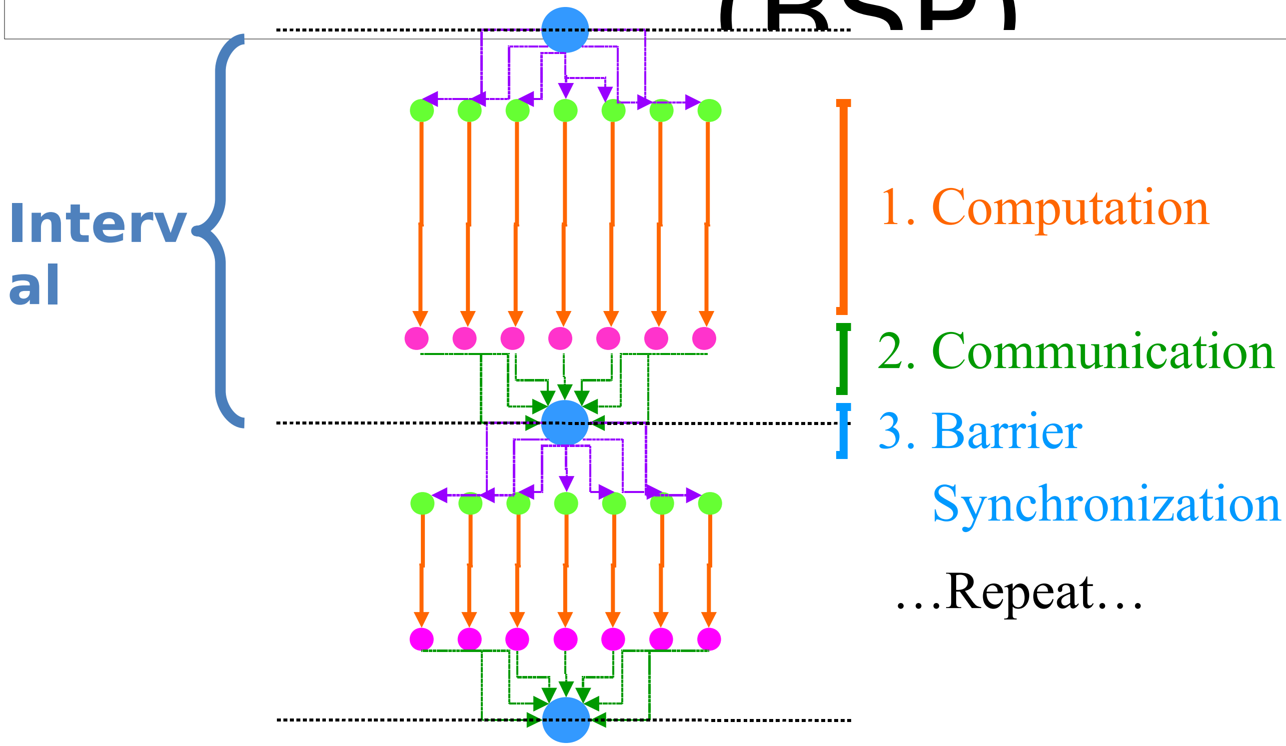
Rigel Design Goals: Clean-slate architecture

- Generalized computation accelerator oriented towards data parallel applications
  - Focus on emerging visual computing and interactive HPC applications
- Maximize objectives
  - Perf/area, Perf/watt (ops/joule), Perf/effort
- Support for work queue-based data parallel programming models
  - E.g., CUDA, OpenCL, OpenMP, Ct, Cilk, etc
  - Task-based models starting to get commercial traction

Work Queue-based Data Parallel Programming



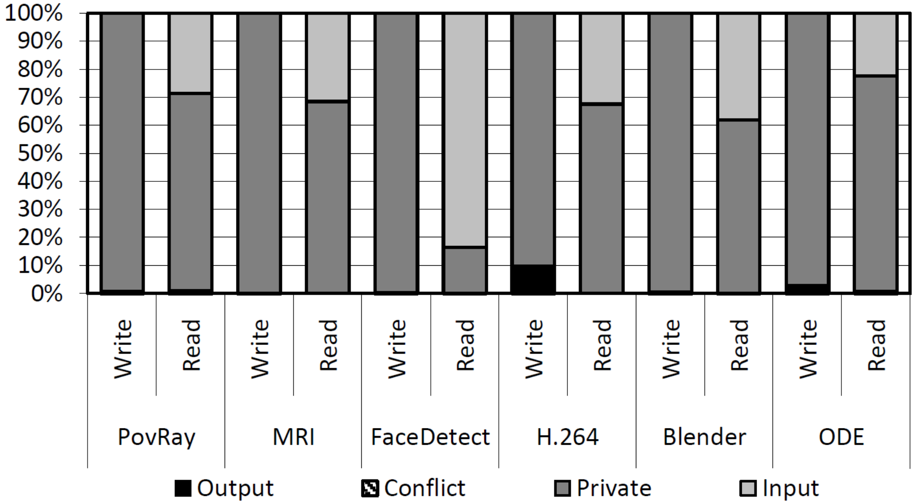
Pattern: Bulk-Synchronous (BSP)



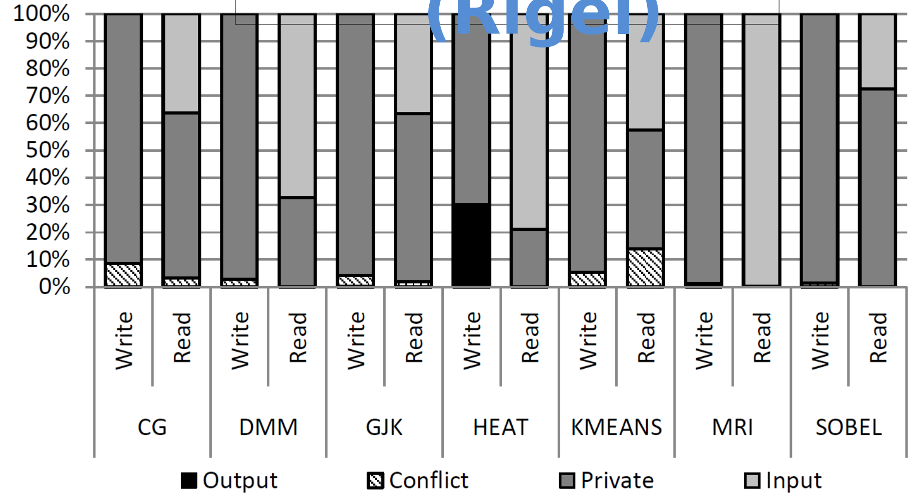
- Compute and communication phases separated by barriers
- High degree of read-sharing within interval
- Private working set, minimal write sharing within intervals

Motivation: Sharing Patterns

VISBench



RTM Kernels (Rigel)



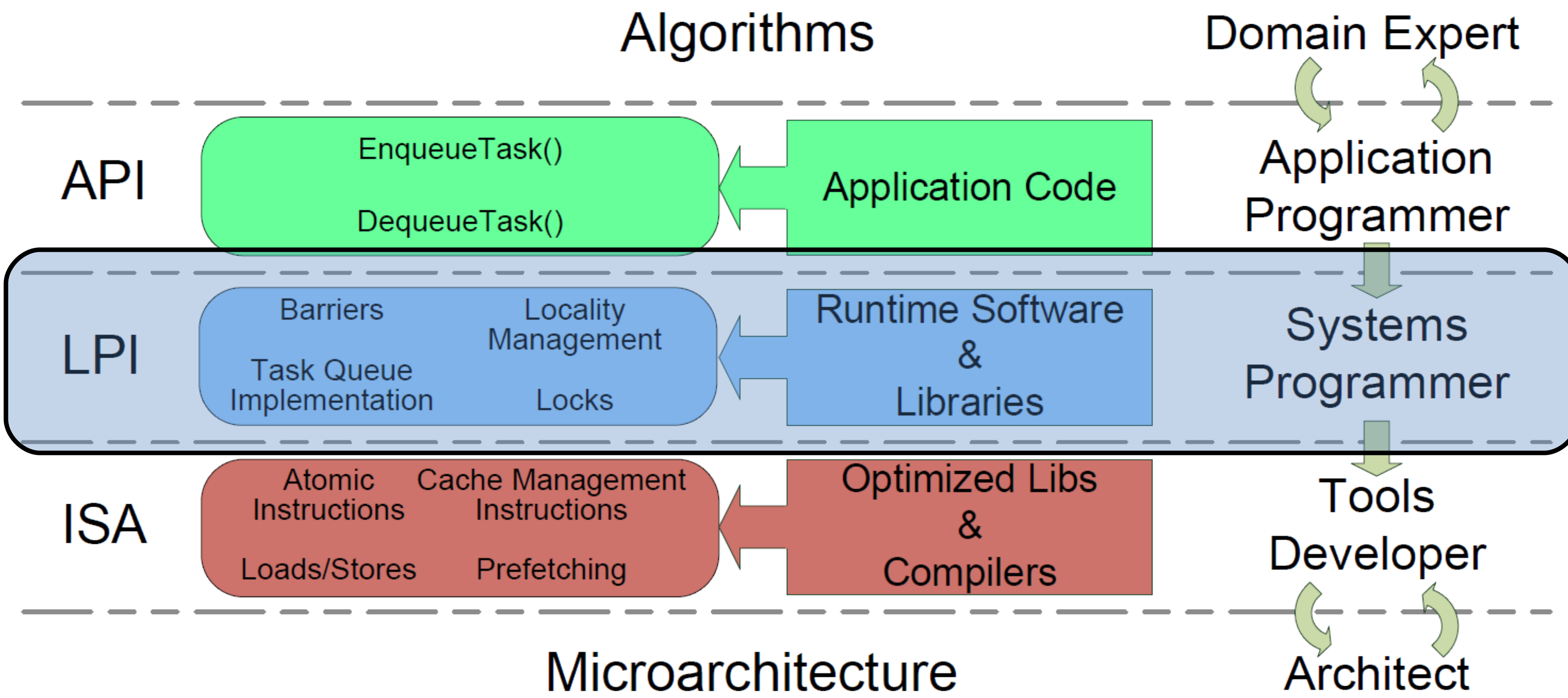
- **Output**: Produced before barrier, read after
- **Conflict**: Written by T1 and read by T2 within an interval
- **Private**: Read/written by only one task
- **Input**: Shared data read by T1, written by T2 in prev. interval

[VISBench: Mahesri et. al, Micro 2008]

Rigel

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Low-level Programming Interface



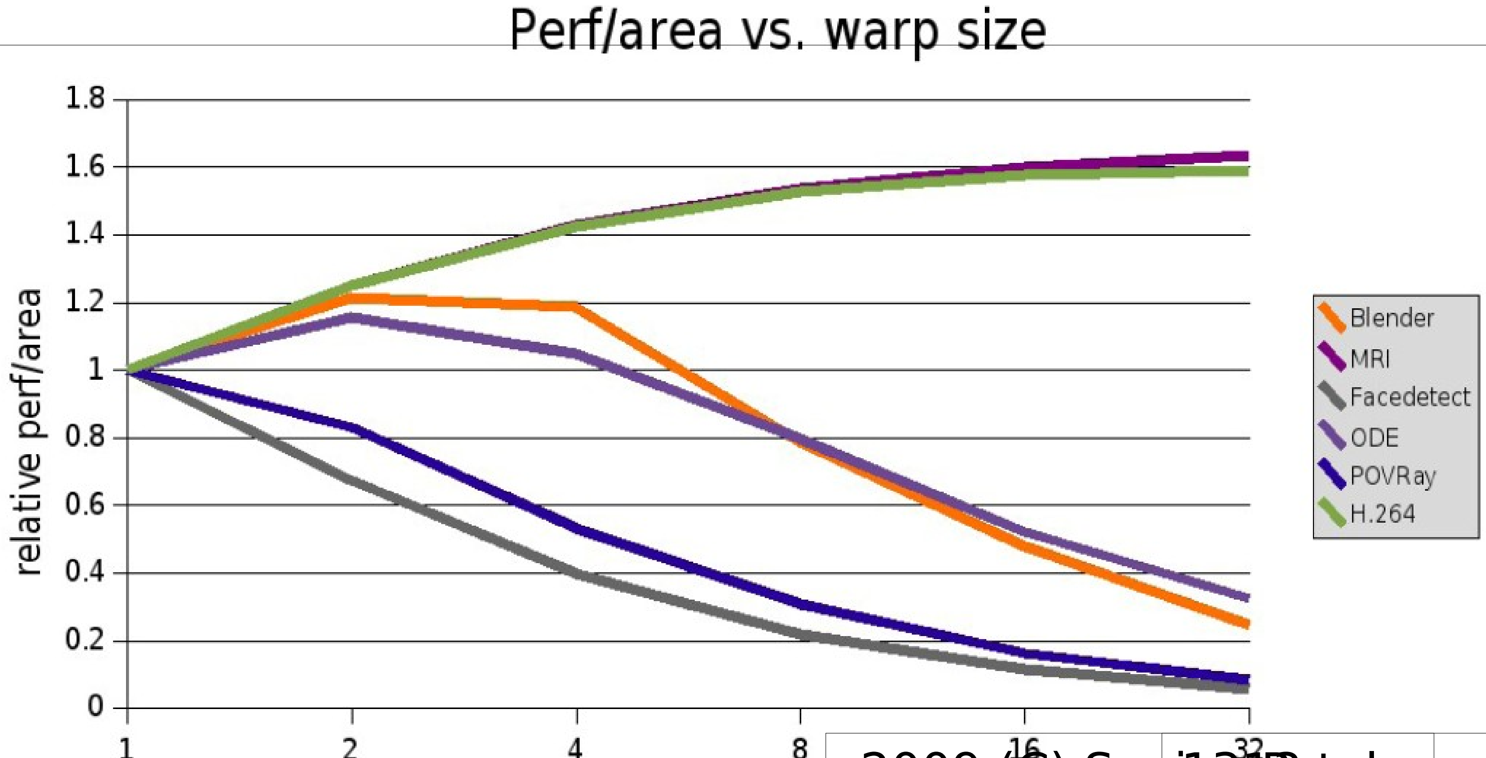
Design Issues: Clean-slate architecture

- Efficiently target wide class of (irregular) parallel apps
- Maximize programmability and productivity

1. **Execution Model**: ISA, SIMD vs. MIMD, VLIW, OoOE, MT
2. **Memory Model**: Caches, scratchpad, ordering, coherence
3. **Work Distribution**: Scheduling, SW/HW spectrum
4. **Synchronization**: Scalability, influence on prog. model
5. **Locality Management:** HW/SW, implicit/explicit

Element 1: Execution Model

- **Tradeoff 1**: **MIMD** vs. SIMD [Maheshi MICRO'08]
  - Additional HW cost vs. SIMD   greater SW flexibility
  - Irregular data parallelism (divergence), task parallelism
  - MIMD: better throughput for irregular apps



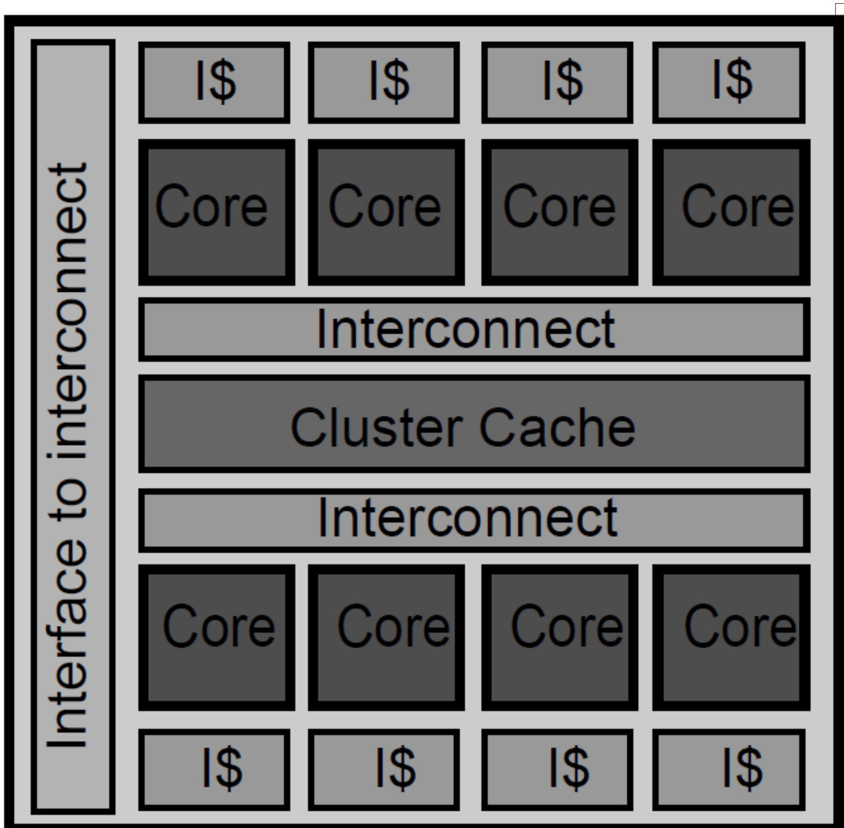
SIMD

Element 1: Execution Model (cont.)

- **Tradeoff 2:** Latency vs. **Throughput**
  - Simple in-order cores [Azizi DasCMP'08]
  - Maximize performance/area (~factor of 2-5x)

- **Tradeoff 3:** **Full RISC ISA** vs. Specialized Cores
  - Complete ISA • conventional code generation
  - No specialized hw ▪ improved compute density
  - Support wide range of apps

Rigel Architecture: Cluster View

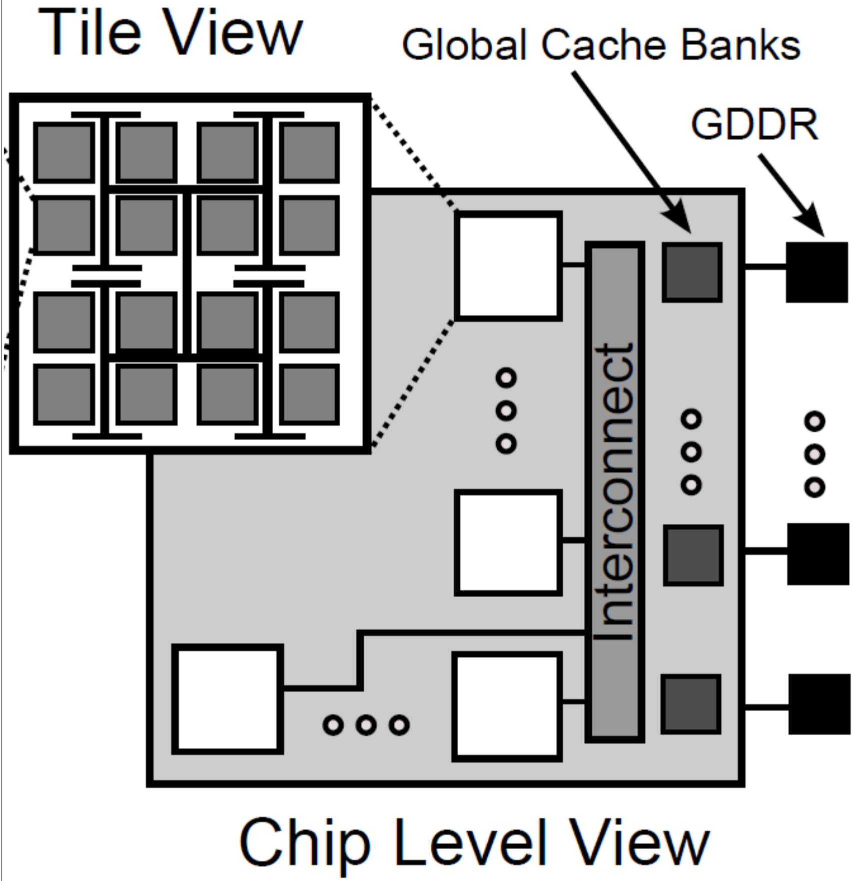


Cluster View

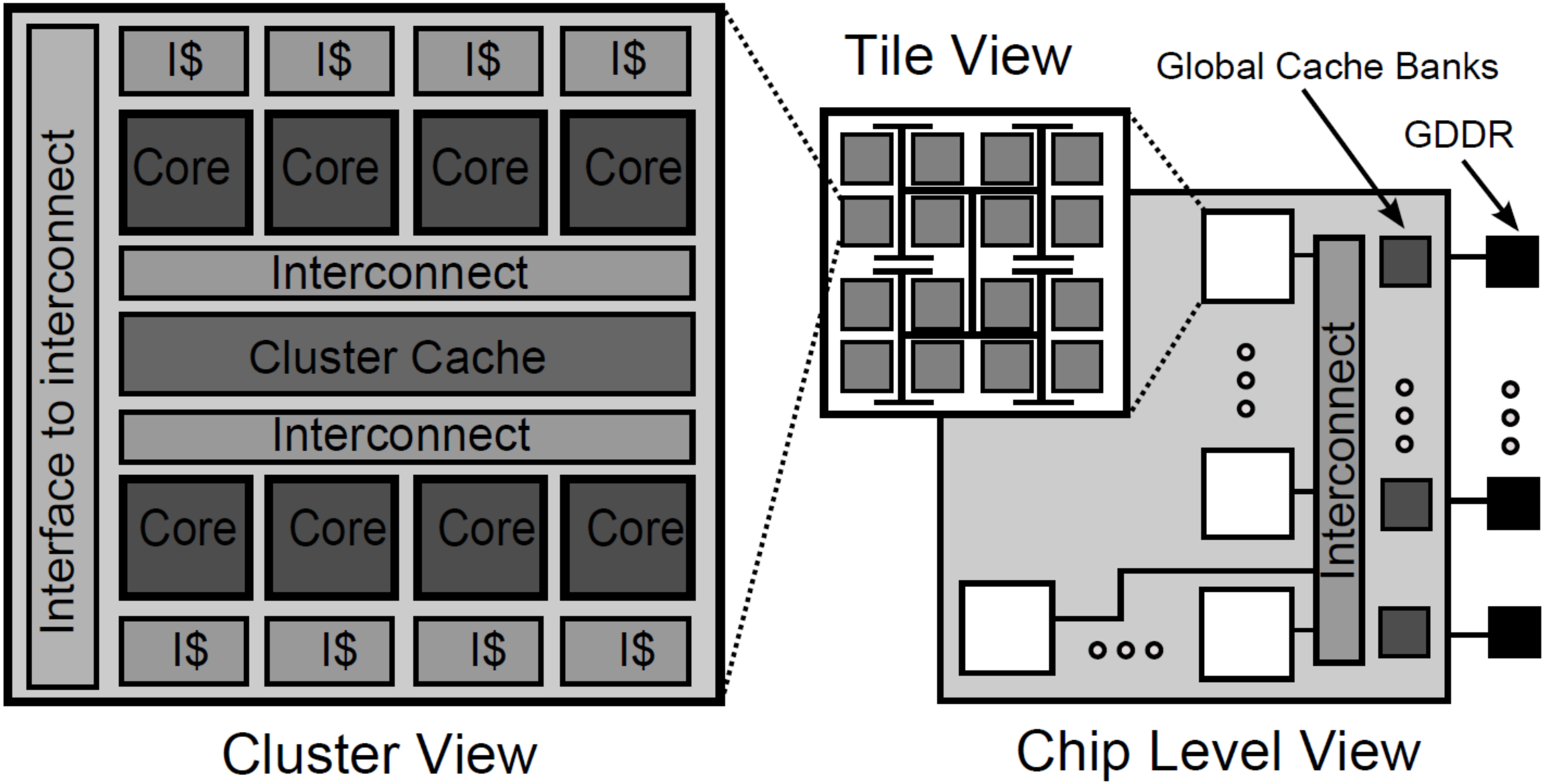
- Rigel's basic building block
- Eight 32b RISC cores
- 64 kB cluster-level shared cache (locally coherent)
- Core Design
  - Simple cores in-order
  - 2-wide issue
  - Per-core SP FPUs
  - Investigating MT granularity

Rigel Architecture: Top Level

- 16 clusters per tile
- Simple tree interconnect within tiles
- Multistage crossbar between tiles and Global Cache
- Optimized for shared memory (no point-to-point communication)



Rigel Architecture: Full Chip View



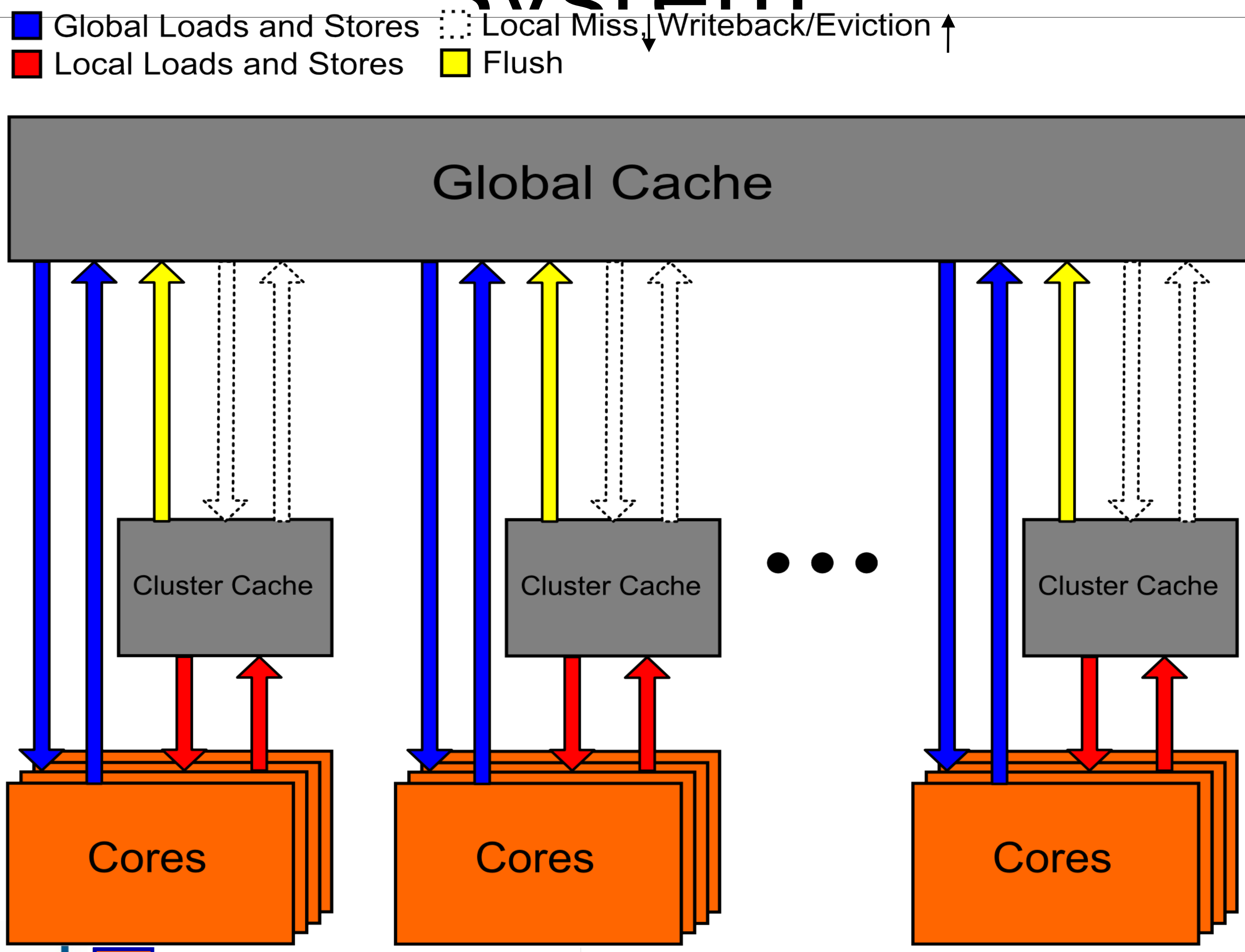
Research Directions: Minimized MIMD

- Convert horizontal vectors into vertical vectors
  - Code compaction, pipeline efficiency
- Superblock, linearized code
  - Reduces number of independent icaches
- Task synchronization to recover locality in data stream

Element 2: Memory Model

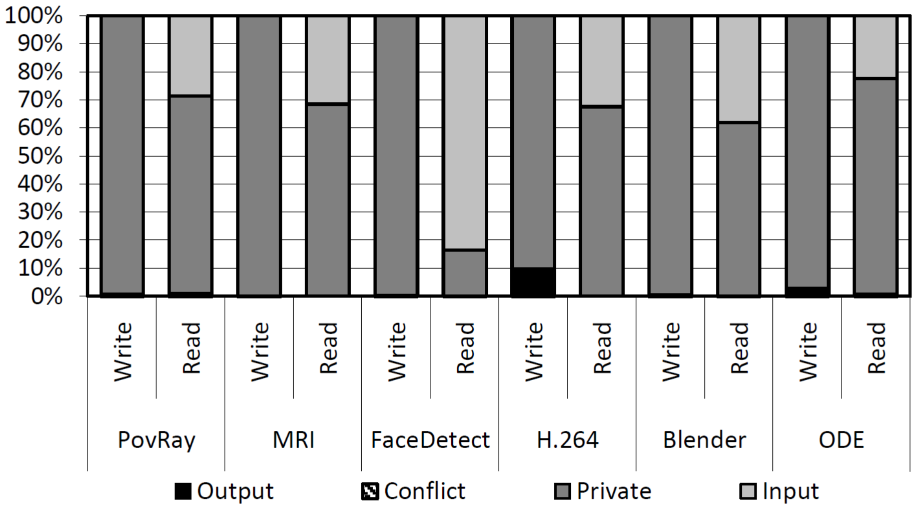
- **Tradeoff 1: Single** vs. multiple address space
  - Ease of programmability
- **Tradeoff 2: Hardware caches** vs. scratchpads
  - Locality management implicit with caches
  - Software manages global sharing
  - Save memory bandwidth
- **Tradeoff 3: Hierarchical** vs. Distributed
  - Cluster cache / global cache hierarchy
  - Local and global memory operations (ISA)
- **Tradeoff 4: C**oherence: HW vs. SW vs. **HW/SW Hybrid**

"Incoherent" Memory System

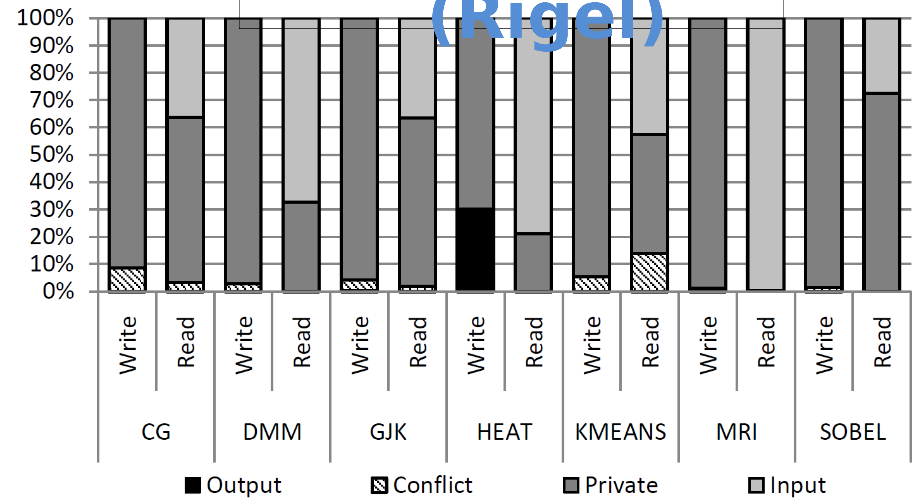


Motivation: Sharing Patterns

VISBench (x86)



RTM Kernels (Rigel)



- **Output**: Produced before barrier, read after
- **Conflict**: Written by T1 and read by T2 within an interval
- **Private**: Read/written by only one task
- **Input**: Shared data read by T1, written by T2 in prev. interval

[VISBench: Mahesri et. al, Micro 2008]

Rigel Memory System

- Stronger memory model on demand
  - Start with global operations (**correctness**)
  - Optimize with local operations (**performance**)
- Global Operations
  - **Visibility**: *All cores using global ops (globally coherent)*
  - Bypass the cluster caches
  - Higher latency, potential contention
- Local Operations
  - **Visibility**: *Only **guaranteed** within cluster*
  - May be incoherent with Global Cache
  - Low latency and high (distributed) bandwidth

SW Coherence

- SW Coherence on Rigel:
  - Based on prominent BSP pattern
  - Programmer annotates shared data
  - Use barrier for reasoning about coherence actions
- At barrier:
  1. Flush modified shared data from cluster cache
  2. Invalidate shared data that was read
  3. Synchronize with other cores
- **[To appear: PACT 2009]**

Element 3: Work Distribution

- **Tradeoff (Spectrum):** HW vs. **SW Implementation**
  - Speed (HW) vs flexibility (SW)
  - Hierarchical task queues: SW task management
  - Flexible policies + small amount of specialized HW
  - Programmable Scheduler

- **Task Management Overheads (@1024 cores)** **[ISCA 2009]**
  - Overheads: enqueue, dequeue, barriers
  - **< 5%** overhead for most regular data-parallel workloads
  - **< 15%** for most *irregular data-parallel* workloads
  - Task lengths: 100’s-100k instructions

Element 4: Synchronization

- Need to accelerates common use patterns, which would normally be supported by coherence mechanisms:
  1. Control synchronization
  2. Data sharing
- Broadcast update (signaling mechanism)
  - Use cases: flags and barriers
  - Reduce contention: poll locally
- Atomic primitives (global cache)
  - Reductions, histograms

Element 5: Locality Management

- Explicit cache management instructions
  - **Goal:** approach control/performance of scratchpad
  - **Optional** management of memory for **performance**
    - Multi-level block prefetch (to various cache levels)
    - Explicit flush, writeback, invalidate
    - Complement local and global operations
    - Automatic generation of management instructions
- Continuing and future work

Rigel vs. Contemporary Accelerators

	Rigel	GPU	Cell	Larrabee
Vectors	**1x (MIMD)**	**32x (SIMD)**	**4x (SIMD)**	**32x (SIMD)**
Memory	**Fully cached**	**Special Purpose**	**DMA+ Scratch**	**Fully Cached**
Address Space	**Single**	**Multiple**	**Multiple**	**Single**
Thread Count	**Some(1-4)**	**Heavy (10s-100s)**	**None**	**Some(~4)**
“Core” Count	**1000s**	**10s-100s**	**10s**	**10s**
Coherence	**HW/SW hybrid**	**None**	**None**	**HW**
Work Distribution	**Software**	**Hardware**	**Software**	**Software**
Specialized HW	**None**	**Significant(gfx)**	**None**	**Some (texture)**

Results: **Scalability**

- Baseline: 8 core cluster
- Based on cycle-accurate, execution-driven simulation
- Library, run-time system code simulated
- Regular C code, standard C compiler, no recompilation

Feasibility: **Area and Power**

- Targeting 45nm process @ 1.2 GHz, 1024 cores
- RTL synthesis results + memory compiler
- Can build this **today:** **320 mm2** die area, **<100W** average power
- Estimated FLOPS/W and FLOPS/mm2 **match or exceed GPUs**

Co-Contributors

- Wen-mei Hwu, Thomas Huang, Mark Horowitz, Minh Do, Steve Lumetta, Matt Frank, Shobha Vasudevan, Sara Baghsorki, Neal Crago, Danny Johnson, Matt Johnson, John Kelm, Aqeel Mahesri, Quang Nguyen, Bill Tuohy, Voytek Truty, Simion Venshtain, and others

Conclusions

- Rigel addresses programmable accelerators challenges
  1. Inflexible programming models
  2. Lack of conventional memory model
  3. Difficulty scaling irregular parallel apps
  4. **FLOPS/Dev. Effort**
- Software coherence viable approach
- Task management requires little HW
- 1024-core accelerator is feasible today
  - **Programmability**: Task API + MIMD execution
  - **Area/performance**: 8 GFLOPS/mm2 @ ~100W
  - GPU-like density, CPU-like programmability

- Thank you!

- Questions?