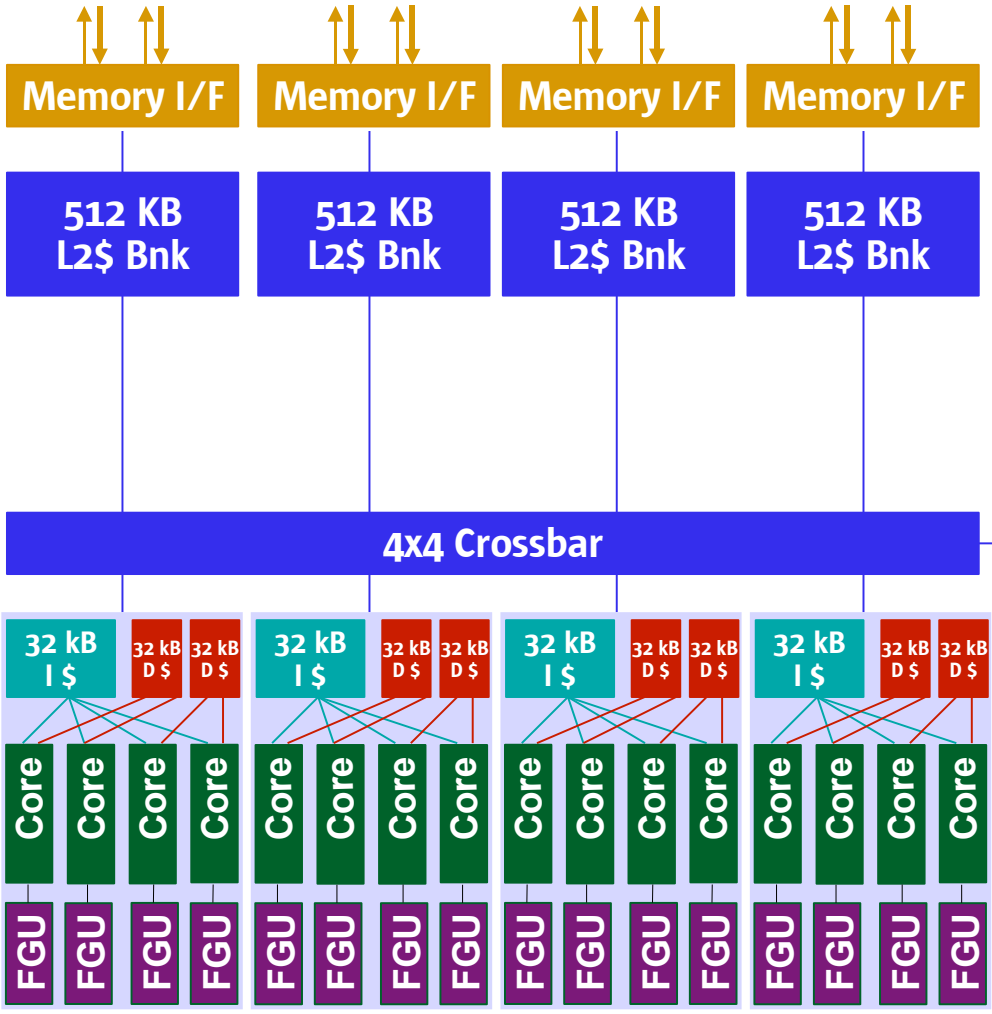
Rock:
A SPARC CMT Processor

Shailender Chaudhry

August 26, 2008

ROCK Block Diagram



8 Memory links
Total 48 GB/s effective BW
plus compression

4 512kB L2$ banks
Each: 64 bytes every other clock
Muxing between 2 banks to keep
4x4 crossbar

IO I/F

1 Gen2 PCIe x16 link
5.32Gbs

4 32kB I$s
Each supplies: 16 instructions/clock

8 32kB D$s
16 banks/D$;2 Rd or 1Wr/clock/bank
Load/use latency: 3 clocks;Write-through

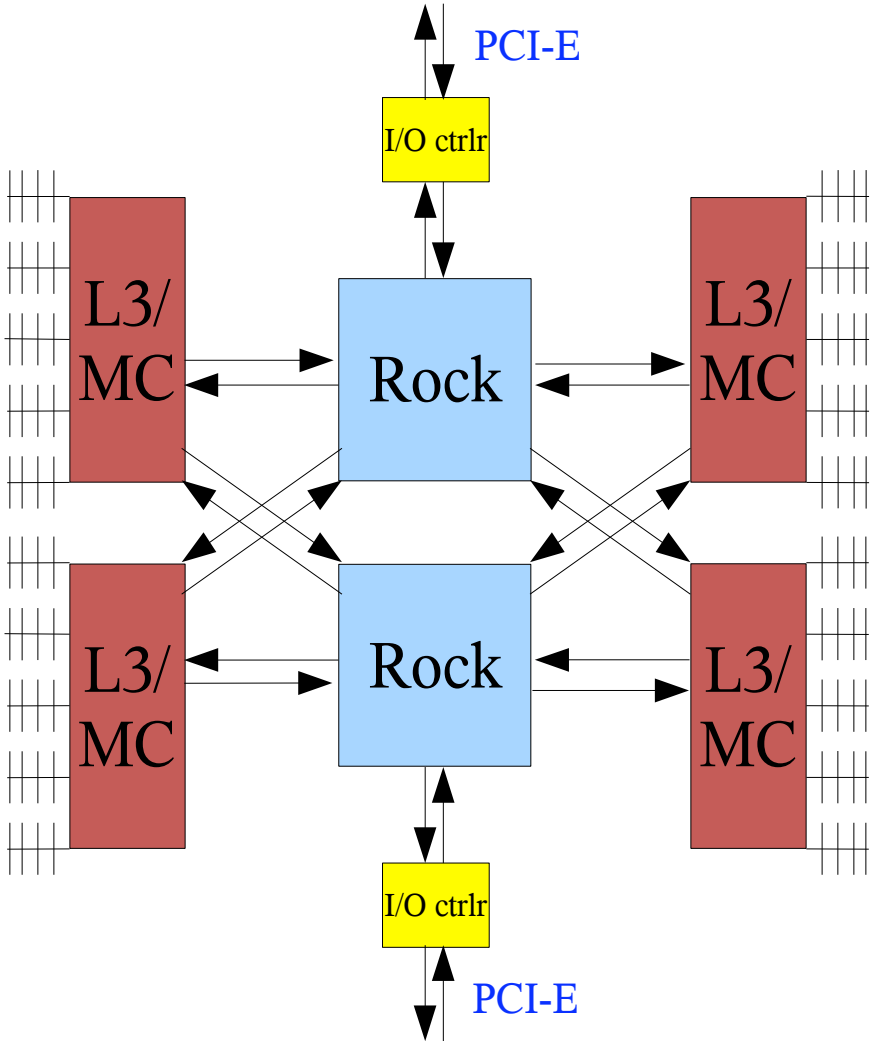
16 Cores and 16 FGUs in 4 Clusters
Each Core: 4 issue superscalar,4 user threads,
each thread with scout threading
Each Core has 6 pipes: load/store, 2 ALU, branch,
FMADD, FMOVE

Rock Features

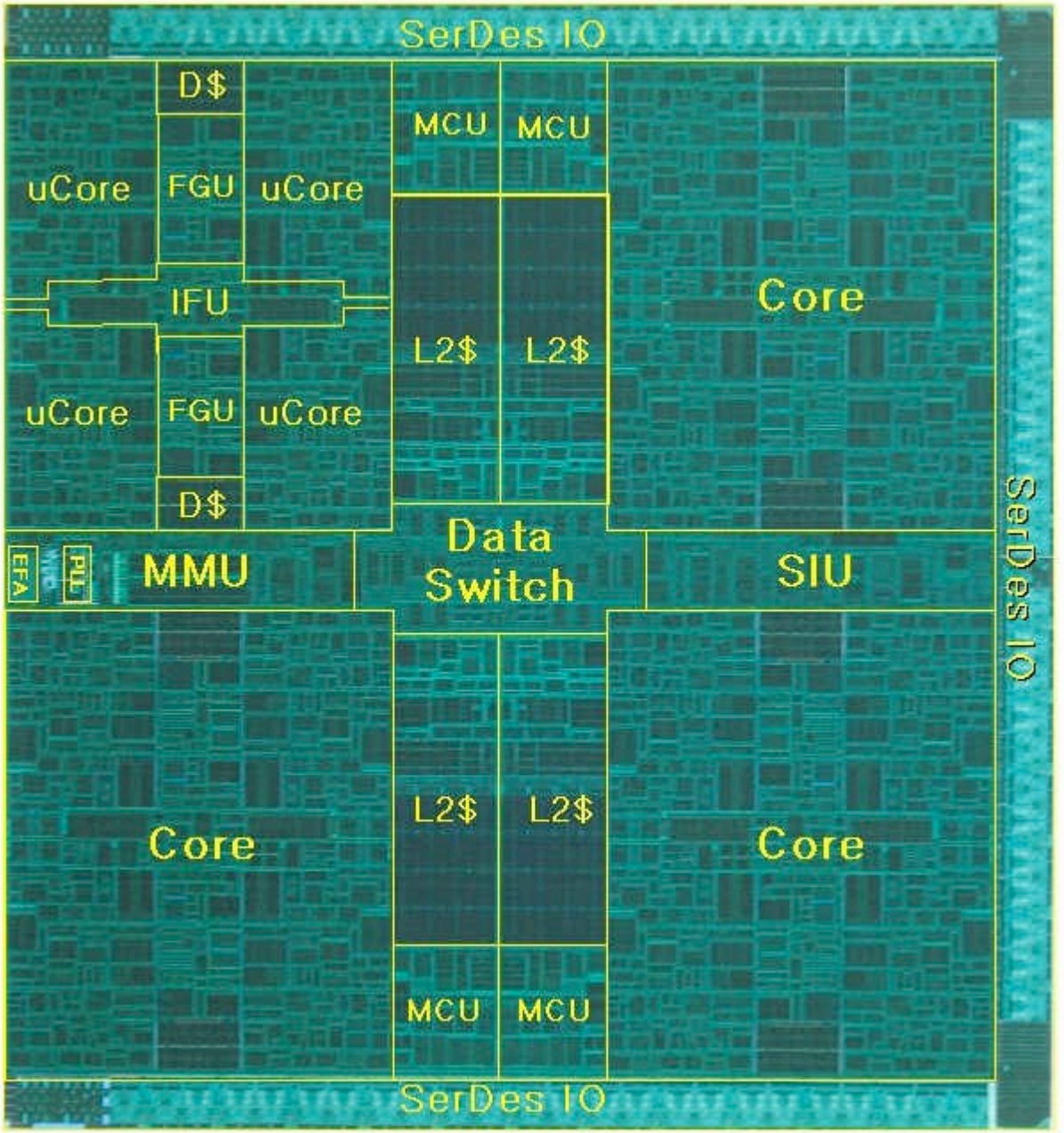
Microcores	16
Strands/microcore	1 or 2
Strands/chip	16 - 32
L1D$	8 x 32KB
L1I$	4 x 32KB
L2$	2MB
L3$ size (off chip)	16MB/ROCK
Peak I/O BW	8 GB/s
Peak Mem BW	48 GB/s
Area	396 mm^2
Frequency	2.1 GHz
Gate Count	5.5 Million
Flop Count	1.1 Million
Transistor Count	321 Million
Power	250W

General Availability 2nd half '09 (ROCK 2.0 in Lab today)

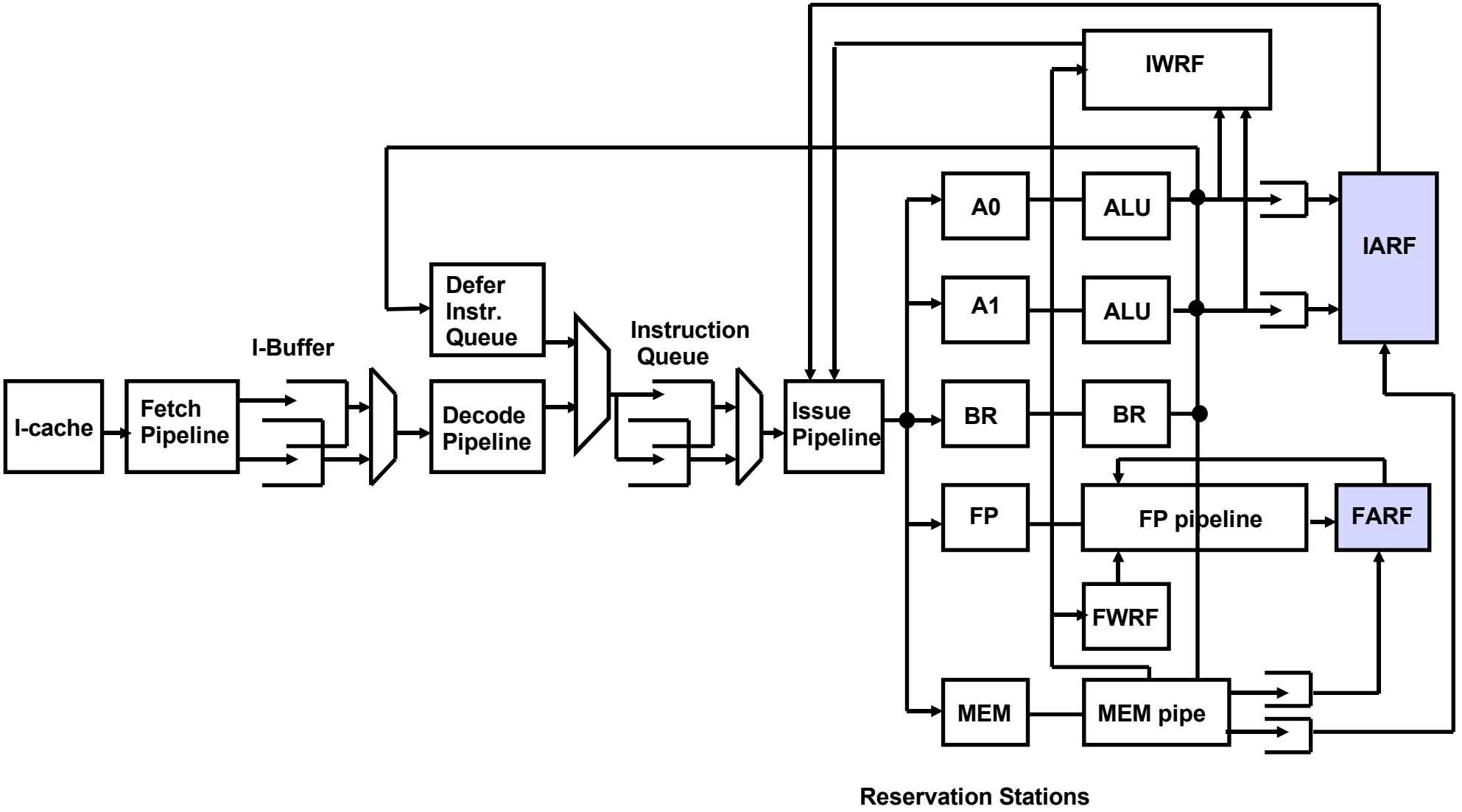
Two Processor Topology



Rock Die Photograph



Rock Pipeline Overview



Rock – Speculation

- **Scout threading**
- **Execute Ahead (EXE)**
  - > Retire independent instructions while scouting
- **Simultaneous Speculative Threading (SST)**
  - > Use two strand resources to parallelize a single thread in HW
  - > Long latency operation and its dependent instructions form a parallel thread of computation
- **Transactional Memory**
- **Parallelization of programs**

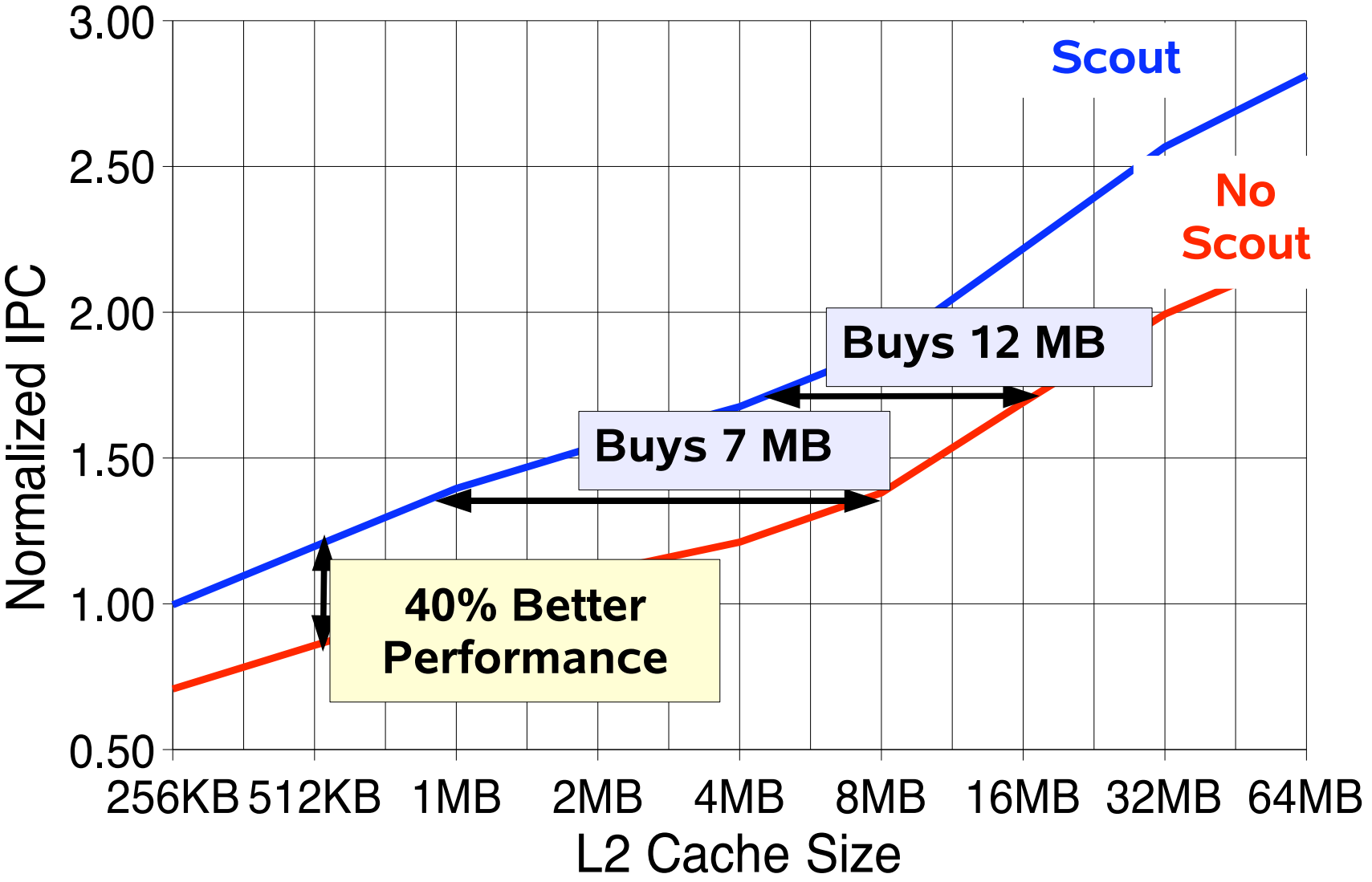
Rock HW Scout

- **HW Scout Idea**
  - > HW scout is invisible to SW
  - > Long-latency instruction starts automatic HW scout:
    - L1D$ miss
    - micro-DTLB miss
    - divide
  - > HW scout:
    - prefetches for loads and stores (Memory Level Parallelism - MLP)
    - warms branch predictor
  - > Long-latency instruction completes
  - > Execution resumes at the long-latency instruction

Rock HW Scout

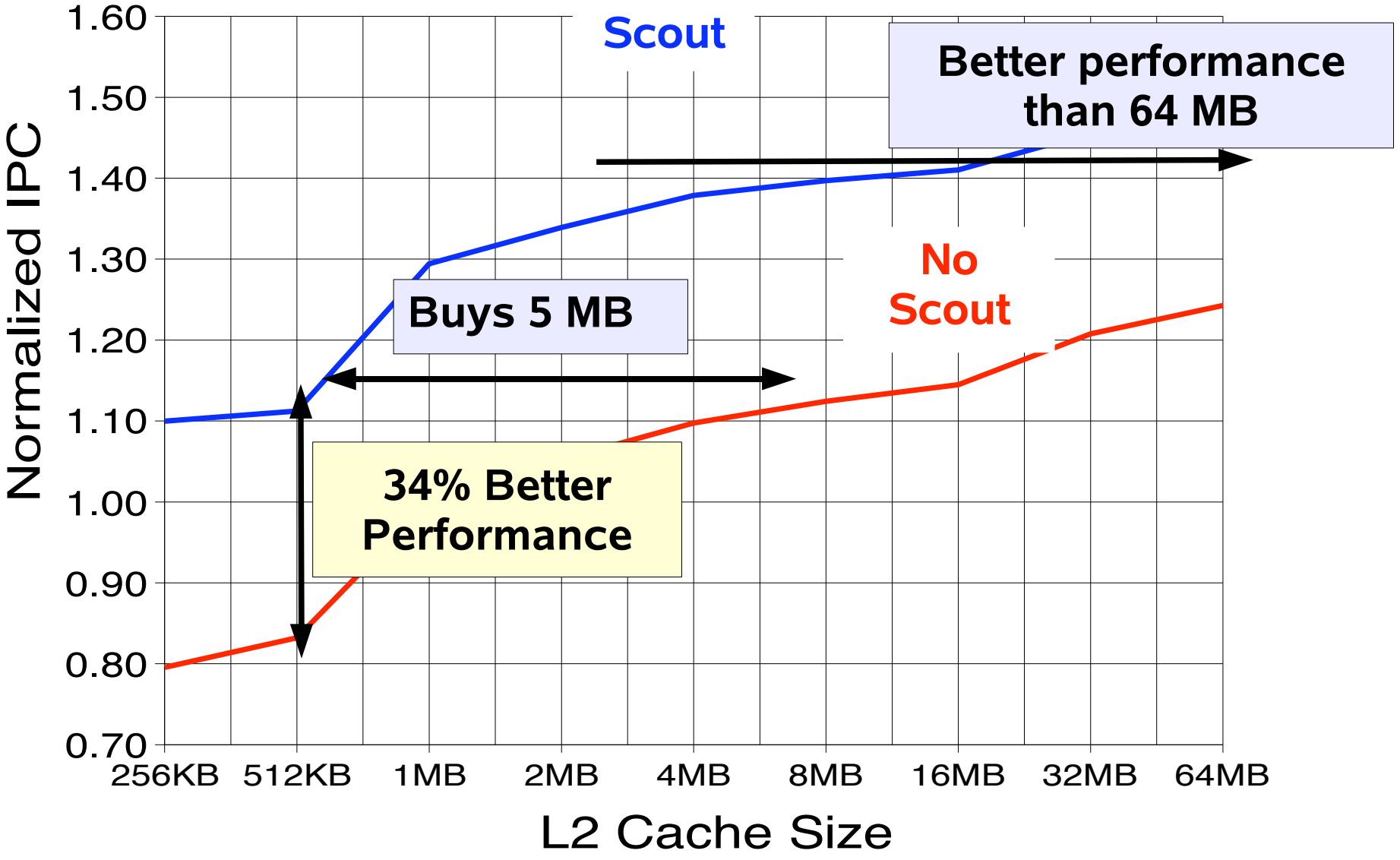
- **HW Scout Implementation**
  - 2 copies of each register:
    - checkpoint (frozen at long-latency instruction)
    - working (includes speculative updates from scout instructions)
  - Each register has NT (NotThere) bit:
    - destination register NT is set iff a source register is NT
    - branches with NT condition codes use predictor
  - Checkpoint operation: freeze checkpoint copies of registers
    - 0-cycle latency
  - Resolved operation: return to execution following checkpoint
    - refetch latency (akin to branch mispredict)

Scouting Improvement - Database



Source: ROCK Simulator running TPCC traces

Scouting Improvement - SPECfp2000



Source: ROCK Simulator running SPECfp traces

Rock Execute-Ahead

- **Execute-Ahead (EXE) Idea**
  - > EXE operation is invisible to SW
  - > Long-latency instruction starts automatic EXE strand:
    - L1D$ miss
    - micro-DTLB miss
    - divide
  - > EXE strand:
    - prefetches for loads and stores (MLP)
    - warms branch predictor
    - defers instructions to deferred queue (DQ) that are dependent on the long-latency instruction along with any operand value that is “there”
    - speculatively retires remaining instructions
  - > Long-latency instruction completes
    - only execute deferred instructions from DQ
    - if all deferred instructions complete successfully, "join" moves state to end of EXE strand's execution
    - otherwise, "fail" resumes execution at the long-latency instruction

Rock Execute-Ahead

- **EXE Implementation**
  - > 2 copies of each register:
    - checkpoint (frozen at long-latency instruction)
    - working (includes speculative updates from EXE instructions)
  - > Each register has NT (NotThere) bit:
    - destination register NT is set iff a source register is NT
    - branches with NT condition codes use predictor
    - NT-operand instructions are deferred to DQ
    - deferred instructions use a separate NT field when re-executed
    - original NT field used to resolve WAW hazards
  - > Checkpoint operation: freeze checkpoint copies of registers
    - 0-cycle latency
  - > "Join" operation: continue execution from end of EXE strand's execution
    - 0-cycle latency (unless bubbles introduced in fetch & decode stages)
  - > "Fail" operation: return to execution following checkpoint
    - refetch latency (akin to branch mispredict)

Rock Execute-Ahead

- **EXE Memory Ordering Implementation**
  - Stores placed in store queue by EXE strand
    - used for RAW-bypasses
    - not committed to memory until "join" operation
  - EXE strand's loads performed out of program order
    - each EXE strand's load sets "s-bit" on entry in L1D$ for given strand
    - invalidation of line with "s-bit" set fails speculation
    - replacement of line with "s-bit" set fails speculation
    - "join" or "fail" clears all "s-bits"
- **Instructions independent of long latency operation are not re-executed.**

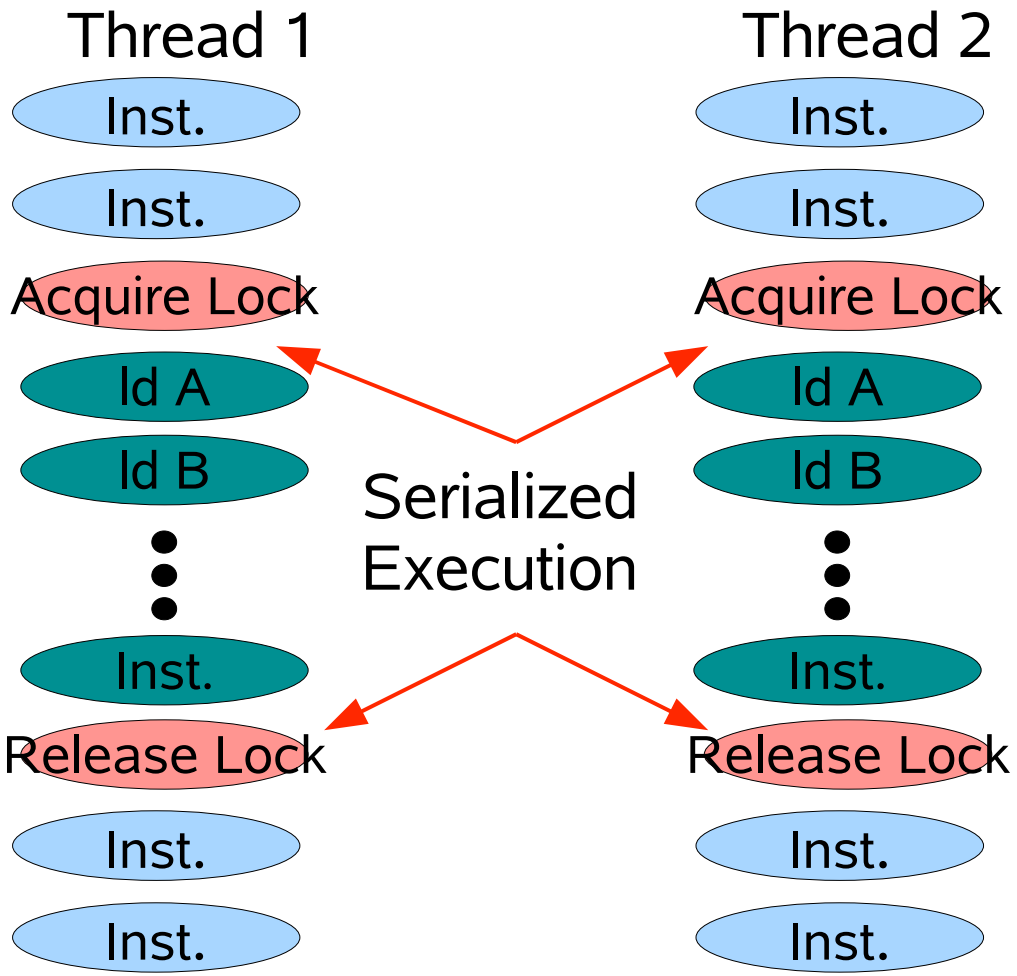
Rock SST

- **Simultaneous Speculative Threading (SST) Idea**
  - SST mode is configured by SW
    - one SW thread per microcore
    - SW thread gets all resources of both HW strands (e.g. store queue, DQ)
  - SST operation is invisible to SW
  - Operate as in EXE mode until long-latency instruction completes
    - 1 HW strand executes deferred instructions ("behind strand")
    - 1 HW strand continues to execute speculatively ("ahead strand")
    - Ahead and behind strands execute completely independently
    - Result is automatic dynamic extraction of ILP and MLP
  - Ahead strand starts deferring instructions to new DQ
  - If behind strand "fails", single strand starts from checkpoint
  - If behind strand "joins", it then starts executing deferred instructions from other DQ

Rock SST

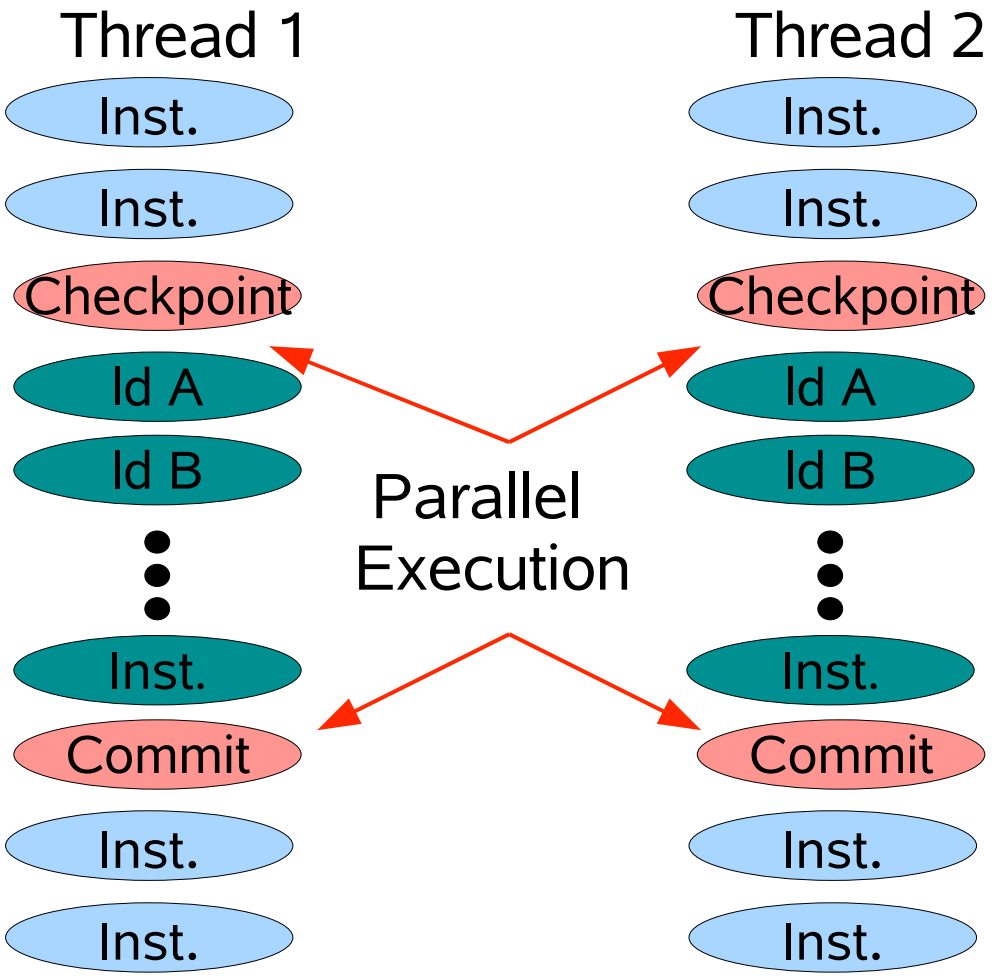
- **SST Implementation**
  - > Use EXE implementation of registers for ahead and behind strands
  - > Use EXE implementation of memory ordering
    - store queue needs to accommodate loads from two points in program order
- **Additional SST Benefit**
  - > Allows Out-of-Order execution around branches
    - checkpoint register state at branch
    - "fail" speculation if branch is mispredicted

Rock TM - Lock Example



- **A thread acquires a lock prior to entering critical section**
- **Uses atomic operation**
  - > Expensive
- **Serializes execution**
  - > Pessimistic
- **Complex**
  - > Requires ordering to avoid deadlocks

Rock TM - TM Example



- **A thread uses a transaction for a critical section**
- **No atomic operation required**
- **Allows parallel execution**
  - > Optimistic
- **Less complex**
  - > No ordering to avoid deadlocks

Rock TM

- **Transactional Memory (TM) Implementation**
  - "RISC" approach:
    - most small transactions executed efficiently by HW
    - best effort: transaction can fail due to contention or implementation reasons
      - The same checkpoint mechanism is used to update arch state or fail
  - Transaction is initiated by SW
    - loads and stores to multiple locations are atomic in the memory order
    - checkpoint: starts transaction and gives PC of failure path
    - commit: completes a successful transaction
  - Enables:
    - lock elision: better performance from unchanged source code
    - atomic replacement: better performance for atomic instructions
    - high-performance transactions: good performance for new code using small transactions
    - SW TM: better performance for SW-implemented TM
    - hybrid TM: HW transactions when possible, SW transactions otherwise

Rock TM

- **TM Implementation**
  - > Transaction is executed speculatively by EXE strand
    - checkpoint is "long-latency instruction"
  - > Use EXE implementation of registers
  - > Use EXE implementation of memory ordering
    - "s-bits" track lines read by transaction and detect conflicts (provides load atomicity)
    - L2$ buffers stores and tracks conflicts until commit, then locks lines until they are updated (provides store atomicity)

Rock Parallelization

- **Autopar/Transactional Parallelization (TP)/SW Scout:**
  - Use multiple SW threads to speed up a single target SW thread
  - Invisible to user SW
    - C/C++: implemented by compiler
    - Java: implemented by JVM
- **Autopar and Transactional Parallelization:**
  - Nonspeculative and speculative thread-level parallelization
  - Use Rock's HW TM support to detect conflicts with parallelized code
    - The program can have dependencies and still be parallelized
- **SW Scout:**
  - Separate SW thread performs prefetching for target thread
  - Uses Rock's shared L1D$ to prefetch to L1
  - Can use Rock's BRNR (branch-on-register-not-ready) to report L1D$ misses to scout thread

Rock's Strengths

- **Area-Efficient High-Performance Core**
  - Reasonable frequency
  - 2-strand support (SMT)
  - OOO 4-issue pipeline
  - 14 mm^2 in 65nm
  - <10W
- **Cache-Miss Tolerance**
  - Scout thread (MLP without reorder buffer area or limitations)
  - EXE thread (ILP under cache miss)
- **Near-Linear Scalability to multiple Sockets**

Rock's Strengths

- **Use of CMT Resources for Single-Thread Performance**
  - SST mode
    - resource sharing (store queue, DQ)
    - automatic dynamic parallelization by HW
    - OOO issue around branches
  - Low-latency communication and shared L1D$
  - Transactional Parallelization (TP)
- **HW Transactional Memory**

Abbreviations

- I/F - interface
- FGU - Floating Point and Graphics Unit
- MMU - Memory Management Unit
- MCU - Memory Control Unit
- IFU - Instruction Fetch Unit
- SIU - System Interface Unit
- A0 - Simple ALU
- A1 - Complex ALU
- BR - Branch Pipe
- FP - Floating Point Pipe
- MEM - Memory Operations Pipe
- FWRF - Floating Point Working Register File
- FARF - Floating Point Architectural Register File
- IWRF - Integer Working Register File
- IARF - Integer Architectural Register File

Rock

Shailender Chaudhry