Stream Processing: a New HW/SW Contract for High-Performance Efficient Computation

Mattan Erez

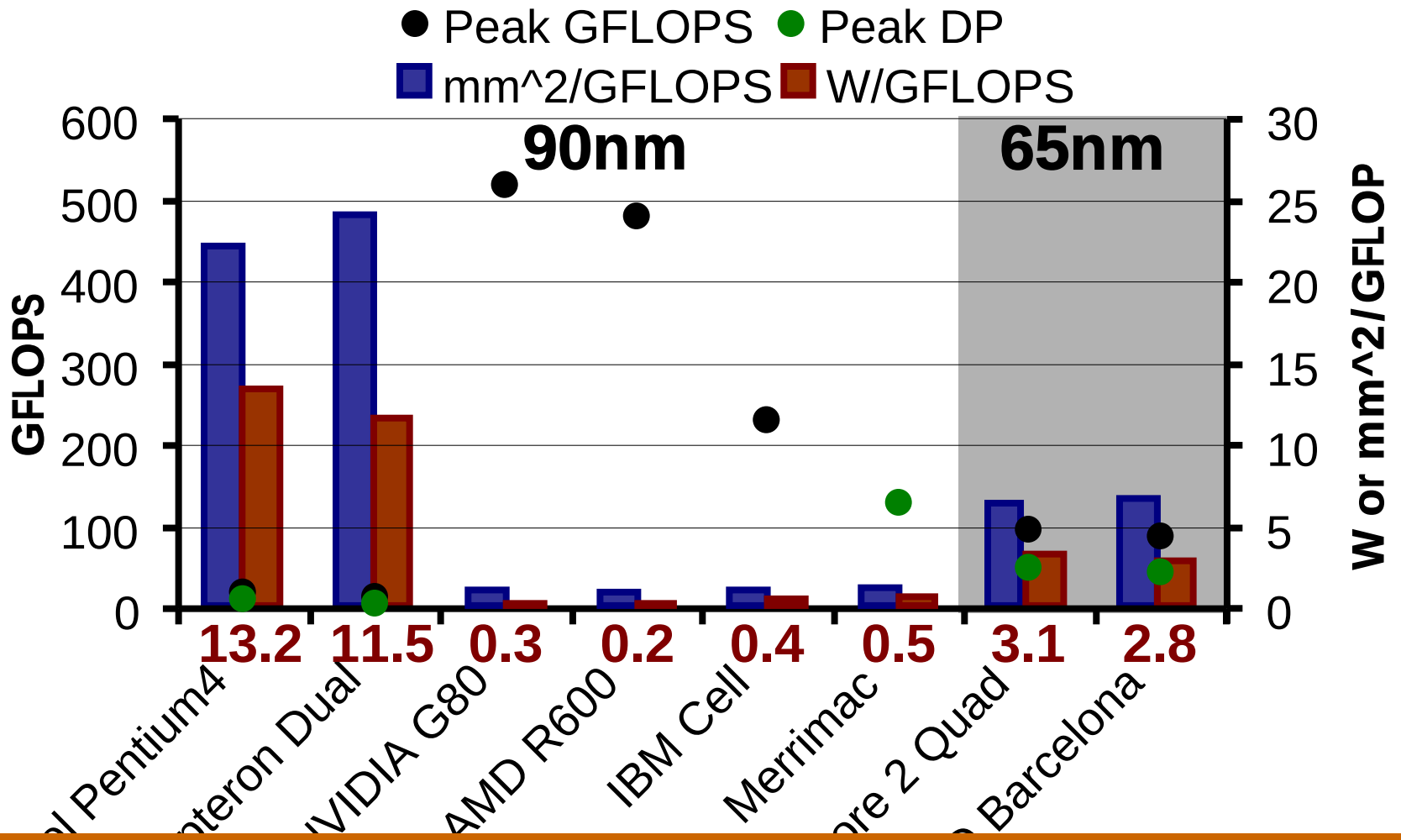
The University of Texas at Austin

CScADS Autotuning Workshop

July 11, 2007

Snowbird, Utah

Stream Processors Offer Efficiency and Performance



Huge potential impact on petaflop system design

Hardware Efficiency → Greater Software Responsibility

- Hardware matches VLSI strengths
  - Throughput-oriented design
  - Parallelism, locality, and partitioning
  - Hierarchical control to simplify instruction sequencing
  - Minimalistic HW scheduling and allocation

- Software **given** more explicit control
  - Explicit hierarchical scheduling and latency hiding (***schedule***)
  - Explicit parallelism (***parallelize***)
  - Explicit locality management (***localize***)

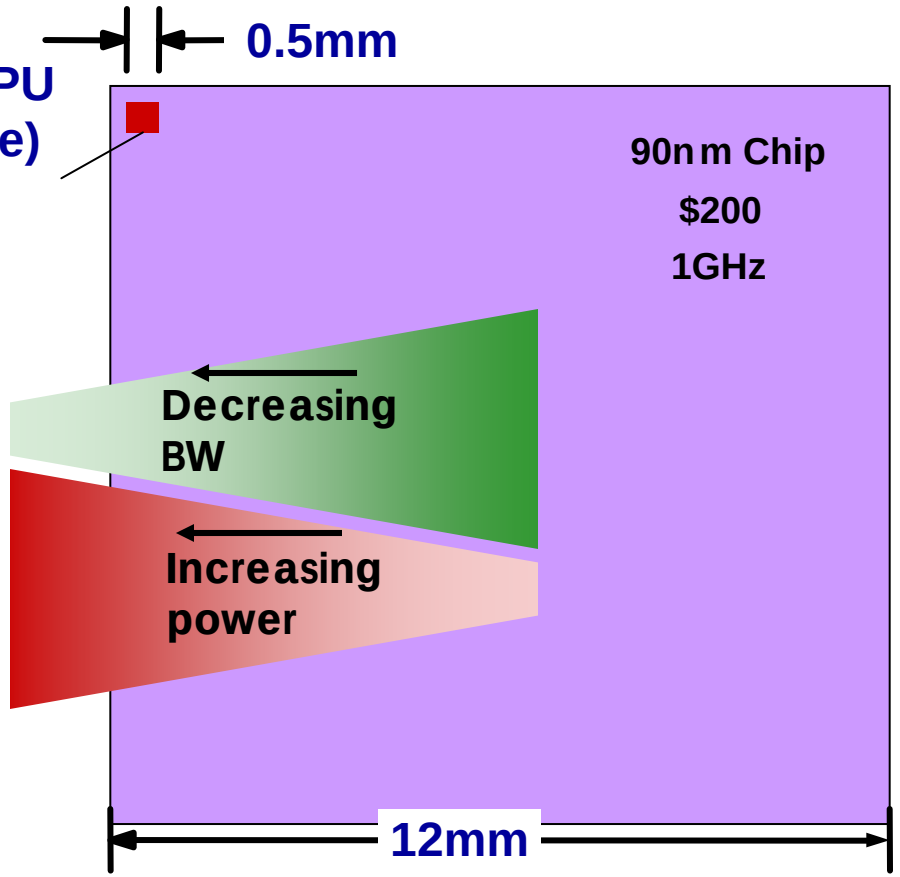
Must reduce HW "waste" but no free lunch

Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

Effective Performance on Modern VLSI

- Parallelism
  - 10s of FPUs per chip
  - Efficient control
- Locality
  - Reuse reduces global BW
  - Locality lowers power
- Bandwidth management
  - Maximize pin utilization
  - Throughput oriented I/O (**latency tolerant**)



Parallelism, locality, bandwidth, and efficient control (and latency hiding)

Bandwidth Dominates Energy Consumption

Operation	Energy (0.13um)	(0.05um)
32b ALU Operation	**5pJ**	**0.3pJ**
Read 32b from 8KB RAM	50pJ	3pJ
Transfer 32b across chip (10mm)	**100pJ**	**17pJ**
Transfer 32b off chip (2.5G CML)	**1.3nJ**	**400pJ**

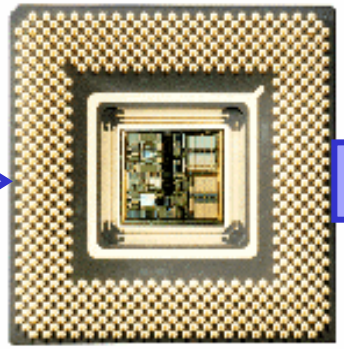
1:20:**260** local to global to off-chip ratio yesterday

1:56:**1300** tomorrow

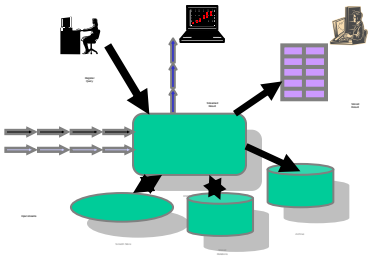
Off-chip >> global >> local >> compute

Stream Execution Model Accounts for Infinite Data

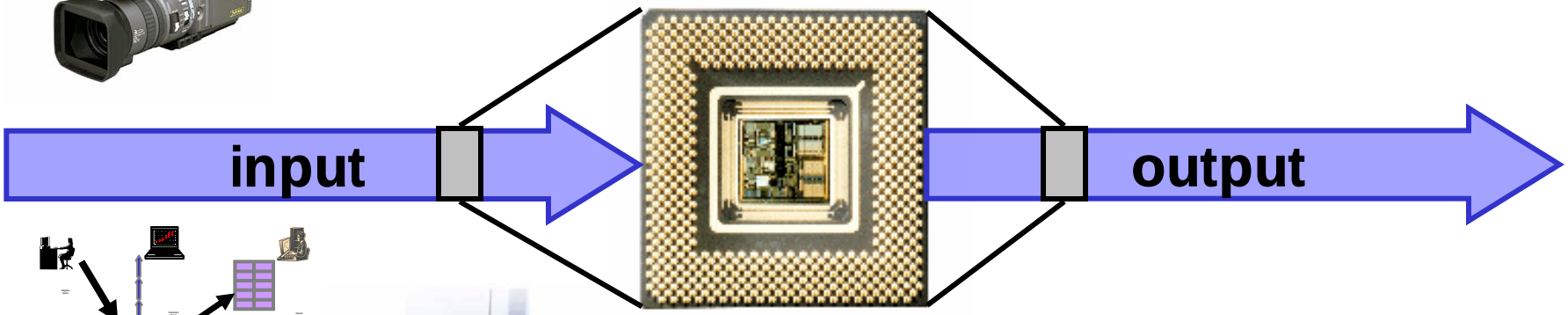
input

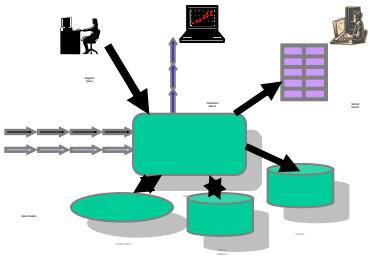


output

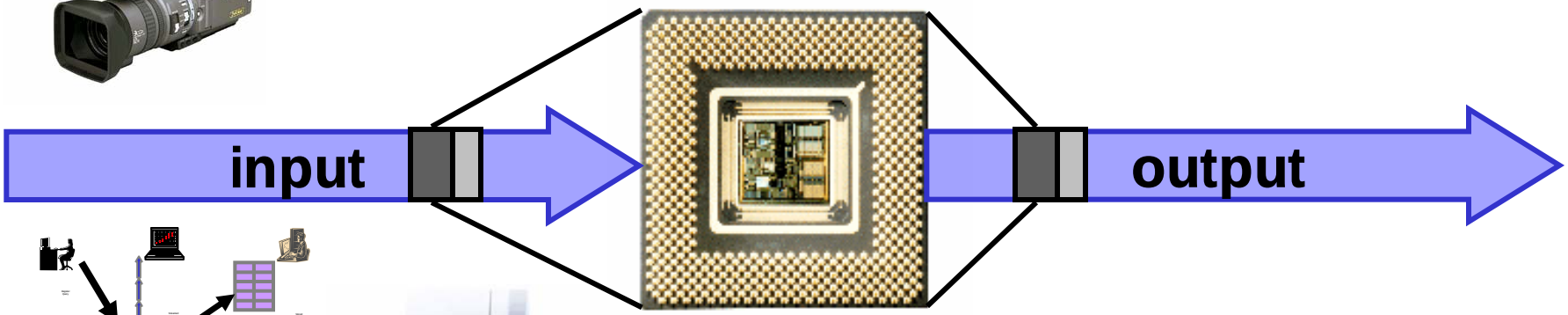


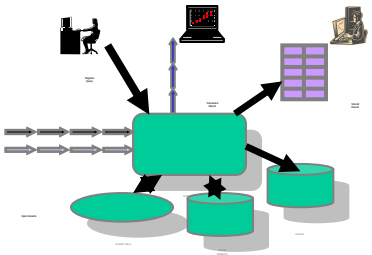
Stream Execution Model Accounts for Infinite Data





Stream Execution Model Accounts for Infinite Data



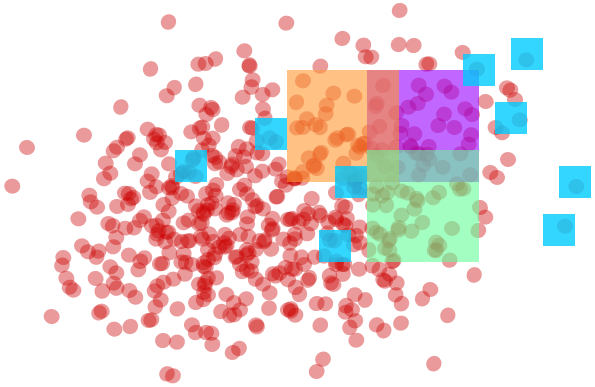


Process *streams* of "bite-sized" data (predetermined sequence)

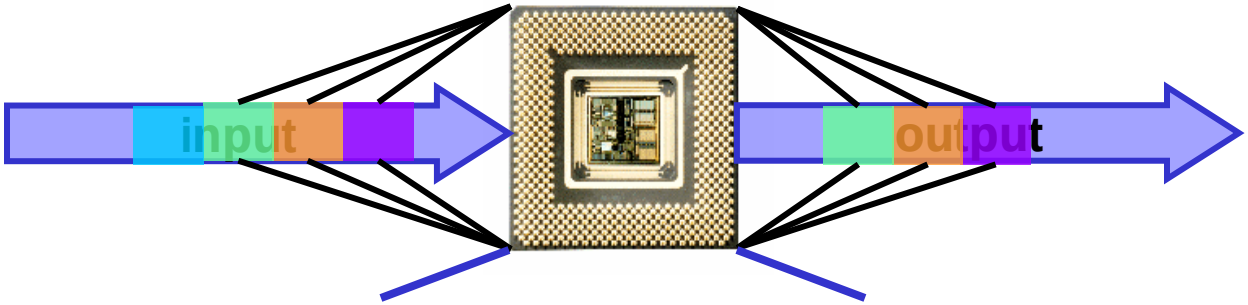
Generalizing the Stream Model

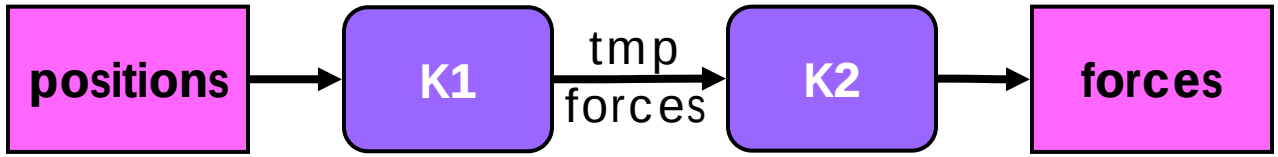
- Data access determinable well in advance of data use
  - Latency hiding
  - Blocking
- Reformulate to ***gather – compute – scatter***
  - Block phases into ***bulk operations***
- "Well in advance": enough to hide latency between blocks and SWP
- Assume data parallelism within compute phase

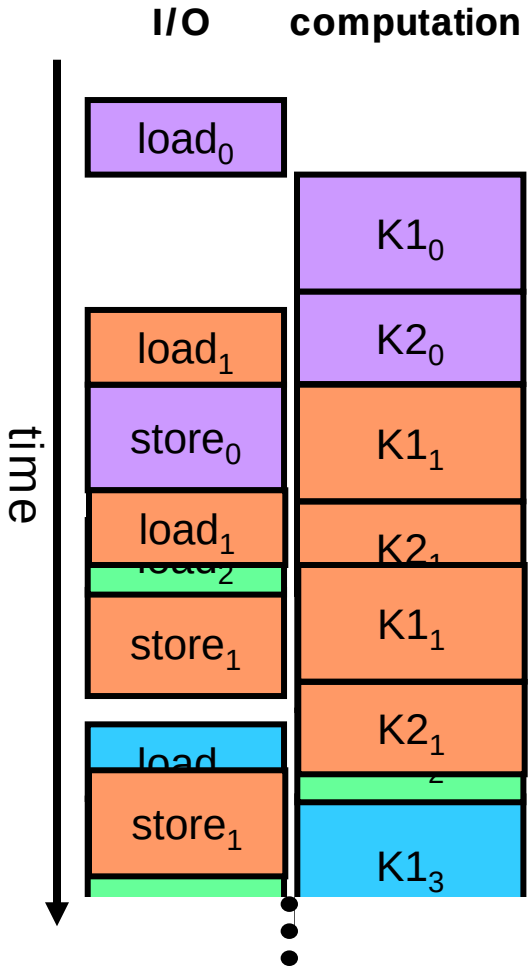
Generalizing the Stream Model



n-body simulation







Latency hiding **– streaming sequence of blocks enables overlapping of I/O and computation**

Generalizing the Stream Model

- Medium granularity bulk operations
  - Kernels and stream-LD/ST
- Predictable sequence (of bulk operations)
  - Latency hiding, explicit communication
- Hierarchical control
  - Inter- and intra-bulk
- Throughput-oriented design
- Locality and parallelism
  - kernel locality + producer-consumer reuse
  - Parallelism within kernels

Generalized stream model matches VLSI requirements

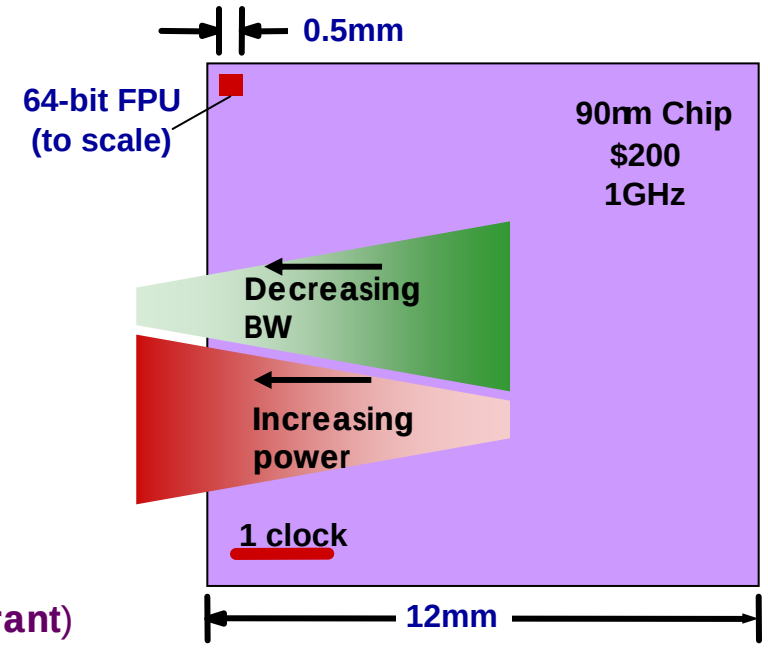
Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

Parallelism and Locality in Streaming Scientific Applications

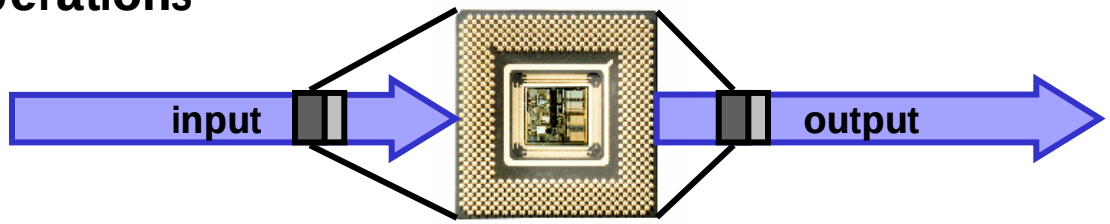
VLSI

- **Parallelism**
  - 10s of FPUs per chip
  - Efficient control
- **Locality**
  - Reuse reduces global BW
  - Locality lowers power
- **Bandwidth management**
  - Maximize pin utilization
  - Throughput oriented I/O (**latency tolerant**)



Streaming model

- **medium granularity bulk operations**
  - **kernels and stream-LD/ST**
- **Predictable sequence**
- **Locality and parallelism**
  - **kernel locality + producer-consumer reuse**

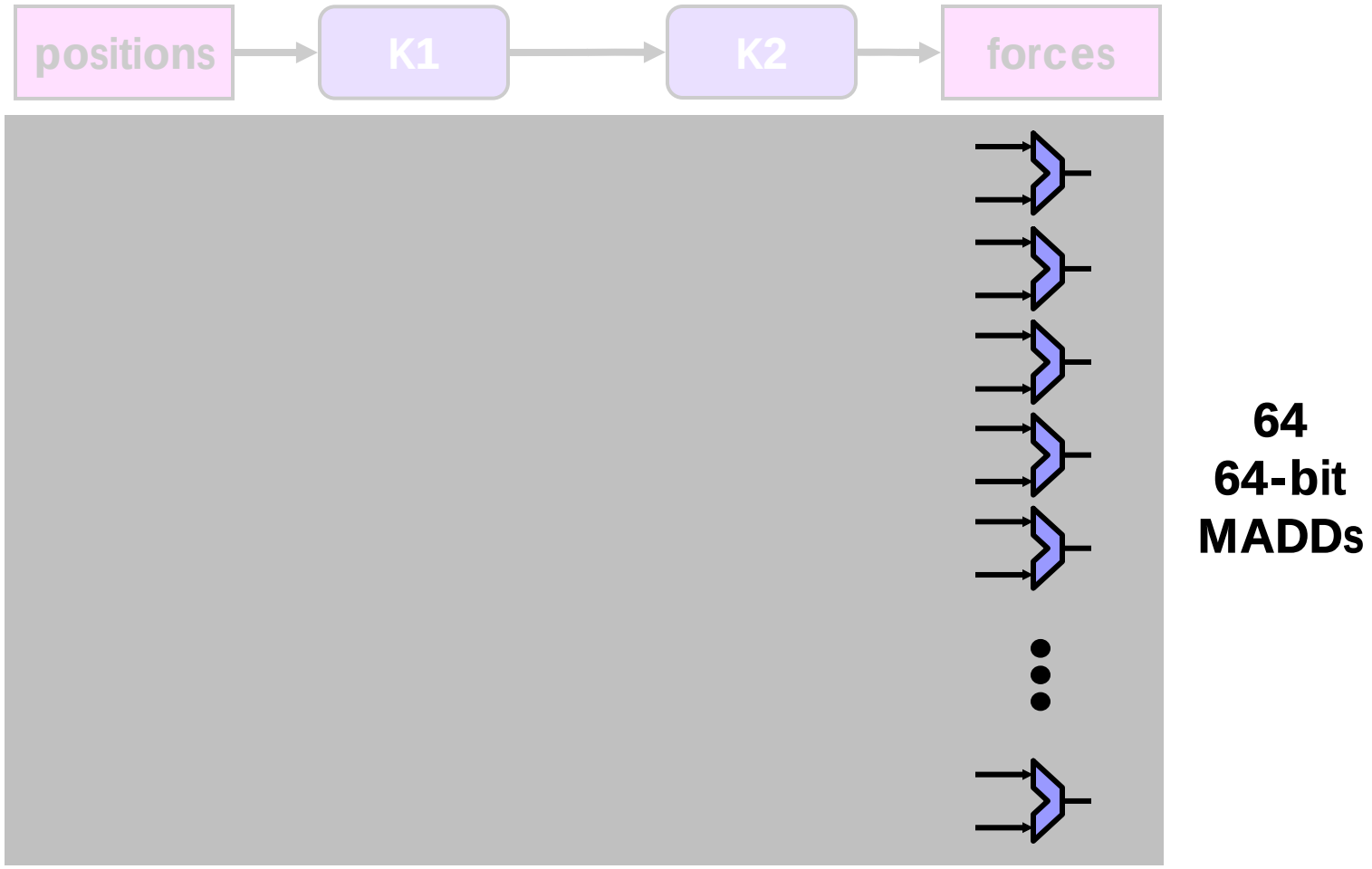


Stream Processor Architecture Overview

- Parallelism
  - Lots of FPUs
  - Latency hiding
- Locality
  - Partitioning and hierarchy
- Bandwidth management
  - Exposed communication (at multiple levels)
  - Throughput-oriented design
- Explicit support of stream execution model
  - Bulk kernels and stream load/stores

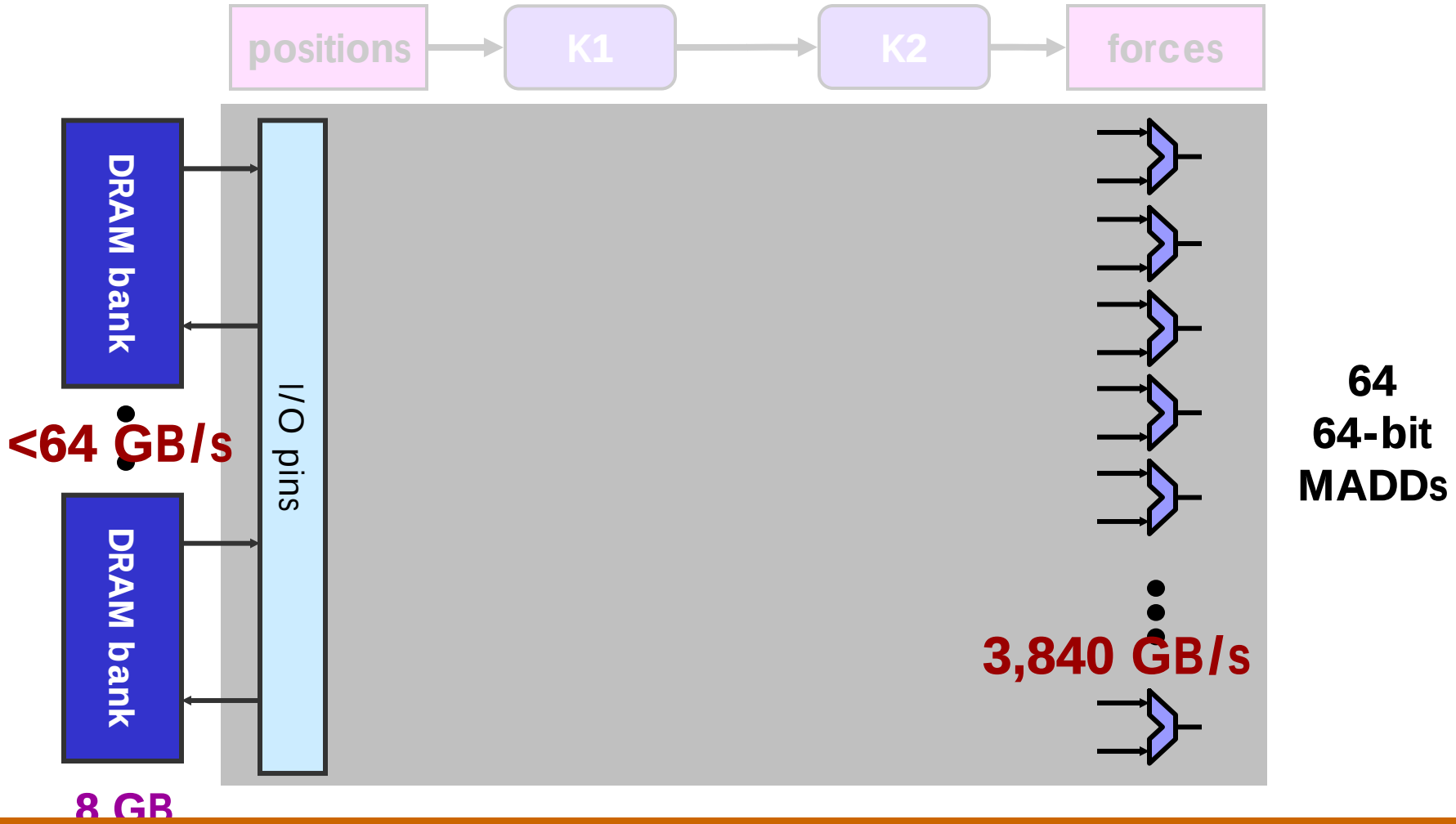
Maximize efficiency:
FLOPs / BW, FLOPs / power, and FLOPs/ area

Stream Processor Architecture (Merrimac)



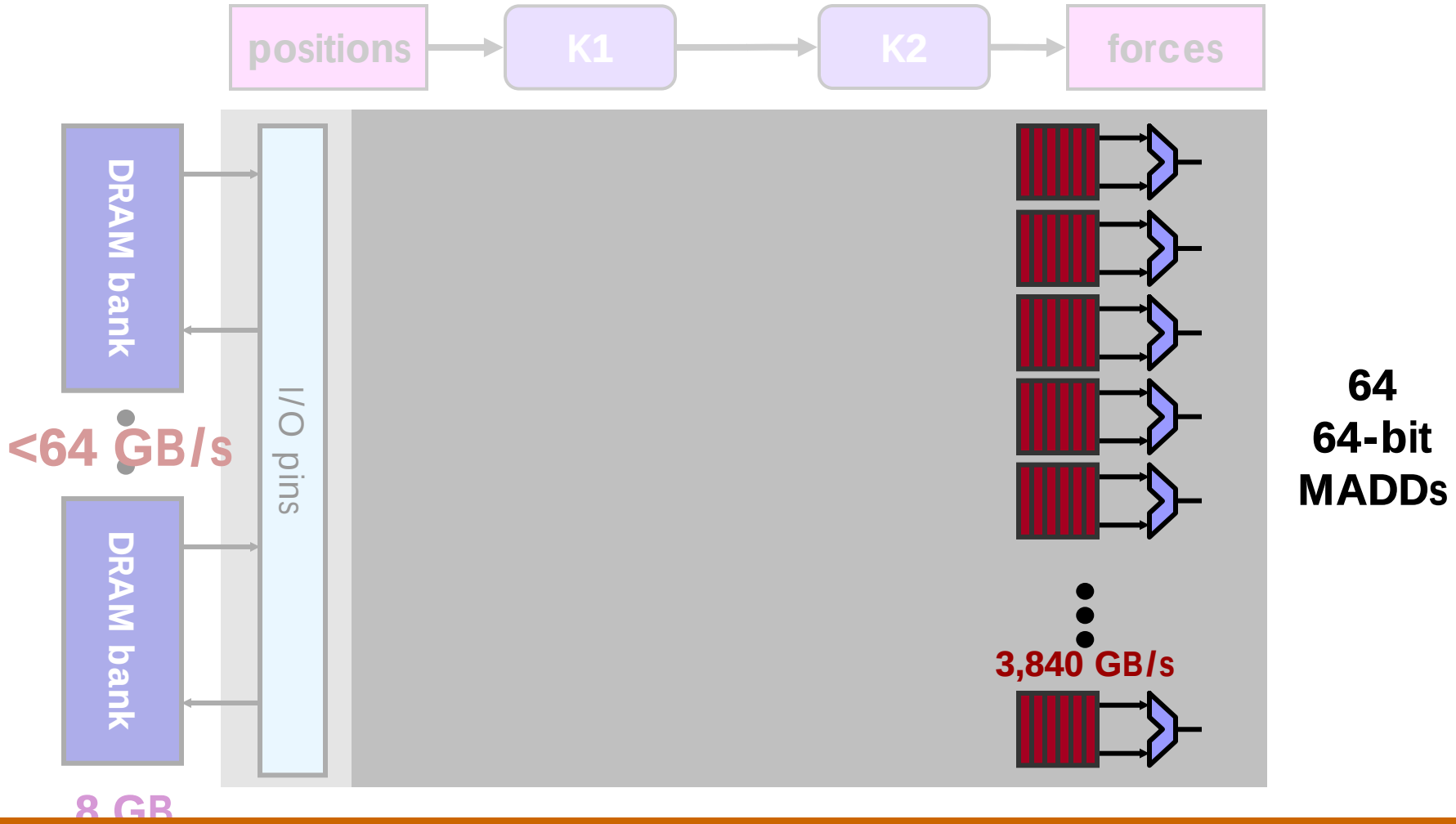
Multiple FPUs for high-performance

Stream Processor Architecture (Merrimac)



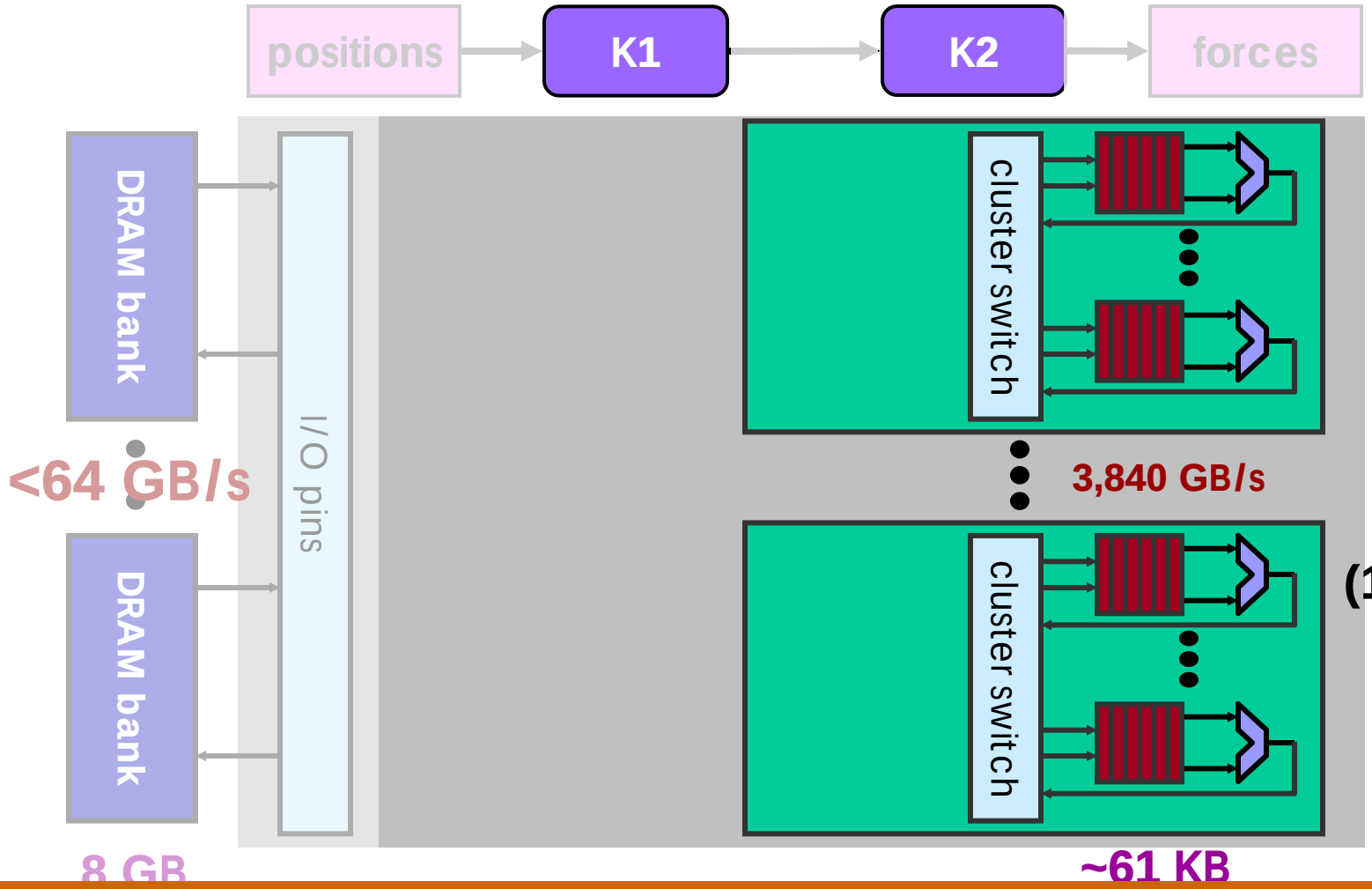
Need to bridge 100X bandwidth gap
Reuse data on chip and build locality hierarchy

Stream Processor Architecture (Merrimac)



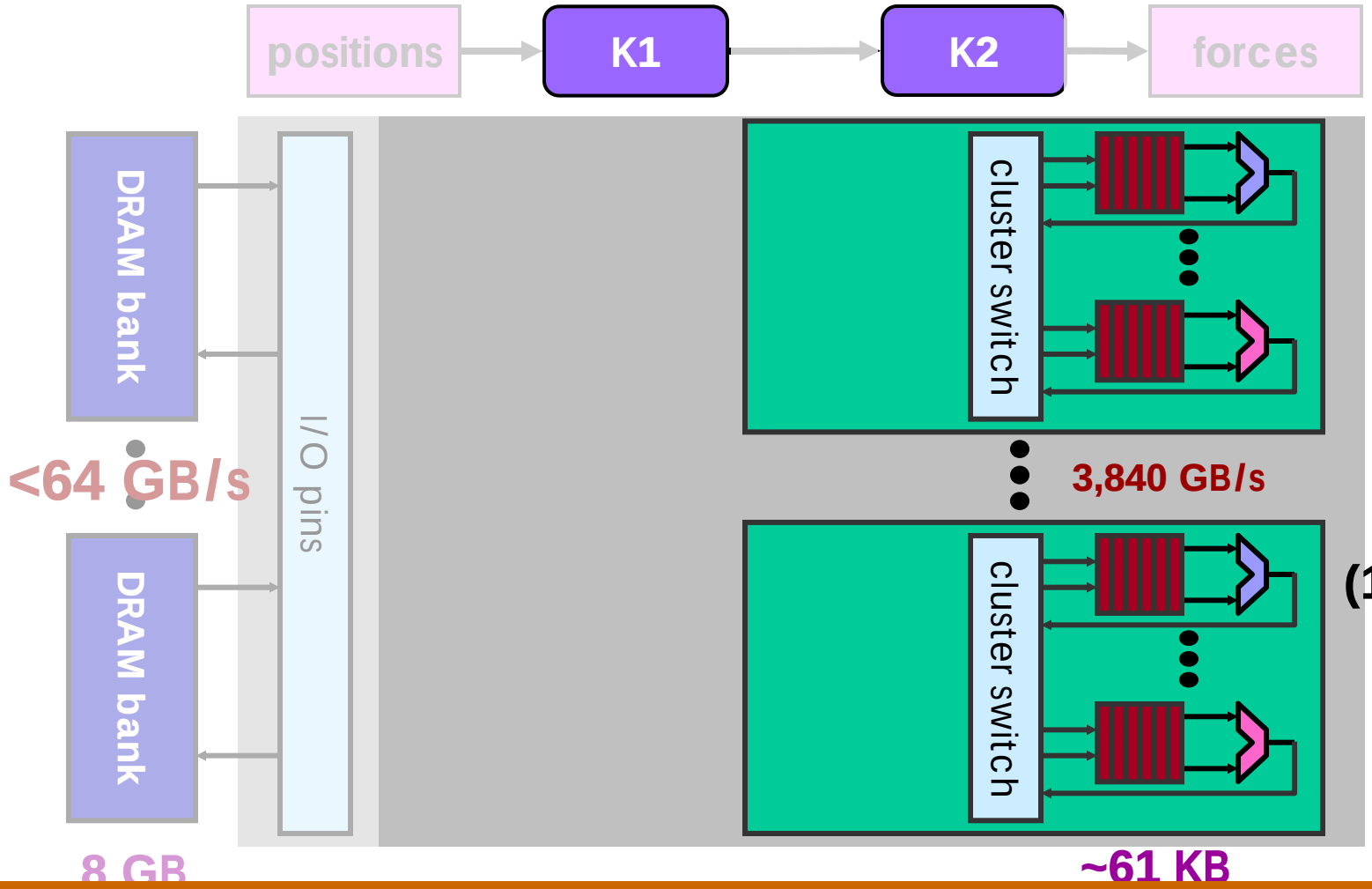
LRF provides the bandwidth through locality
Low energy by traversing short wires

Stream Processor Architecture (Merrimac)



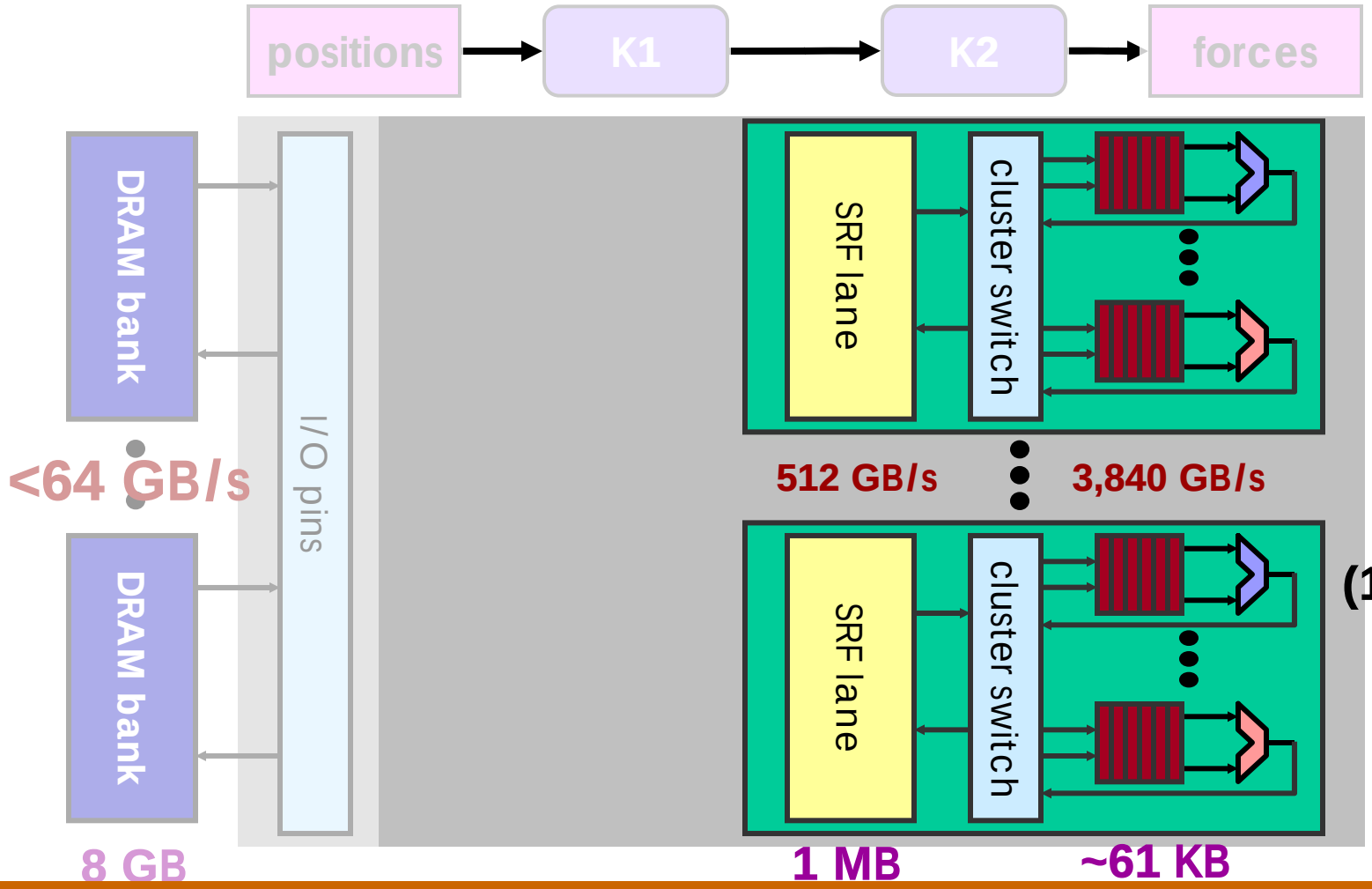
Clustering exploits kernel locality (short term reuse)

Stream Processor Architecture (Merrimac)



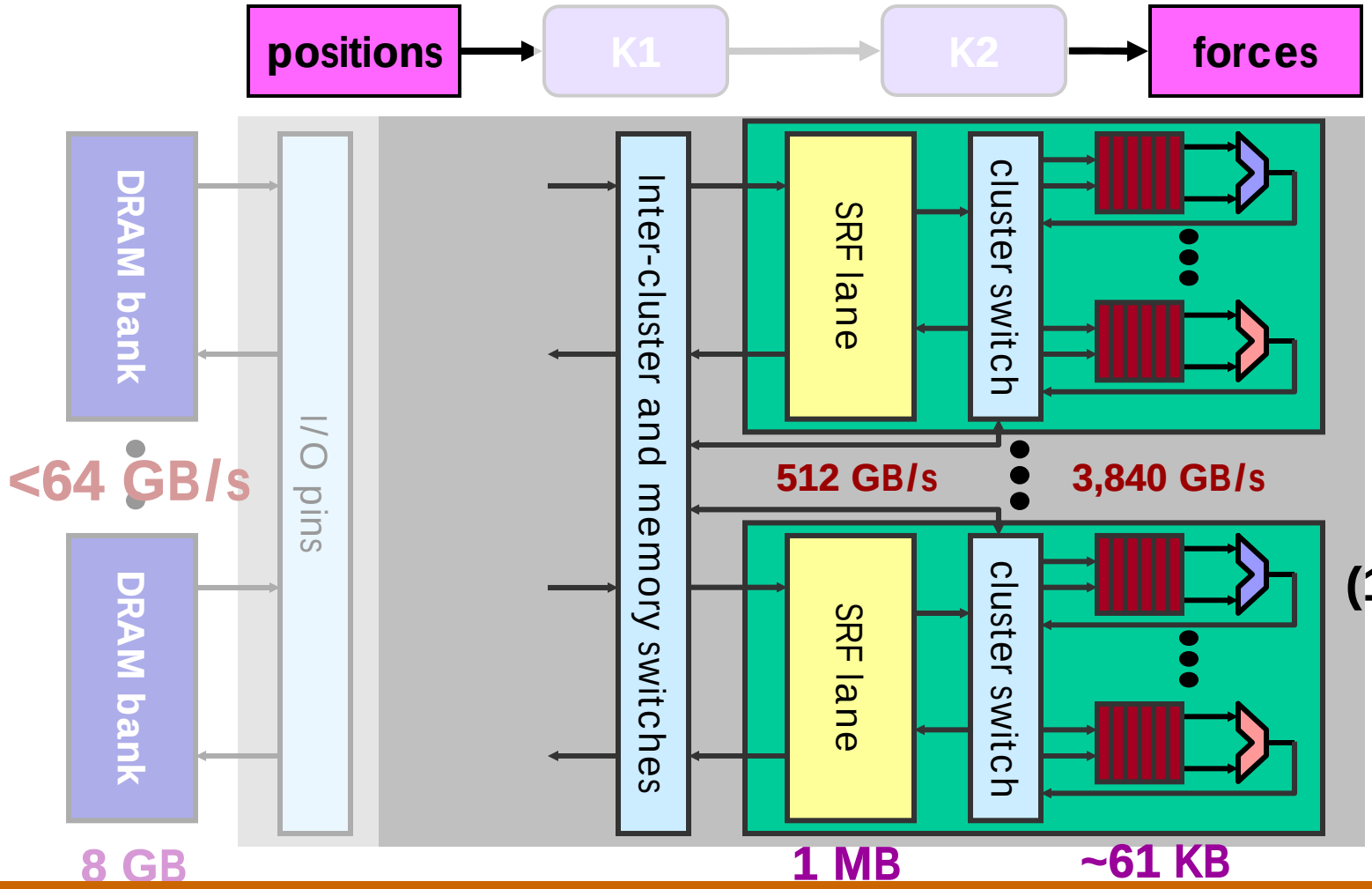
Clustering exploits kernel locality (short term reuse)
Enables efficient instruction-supply

Stream Processor Architecture (Merrimac)



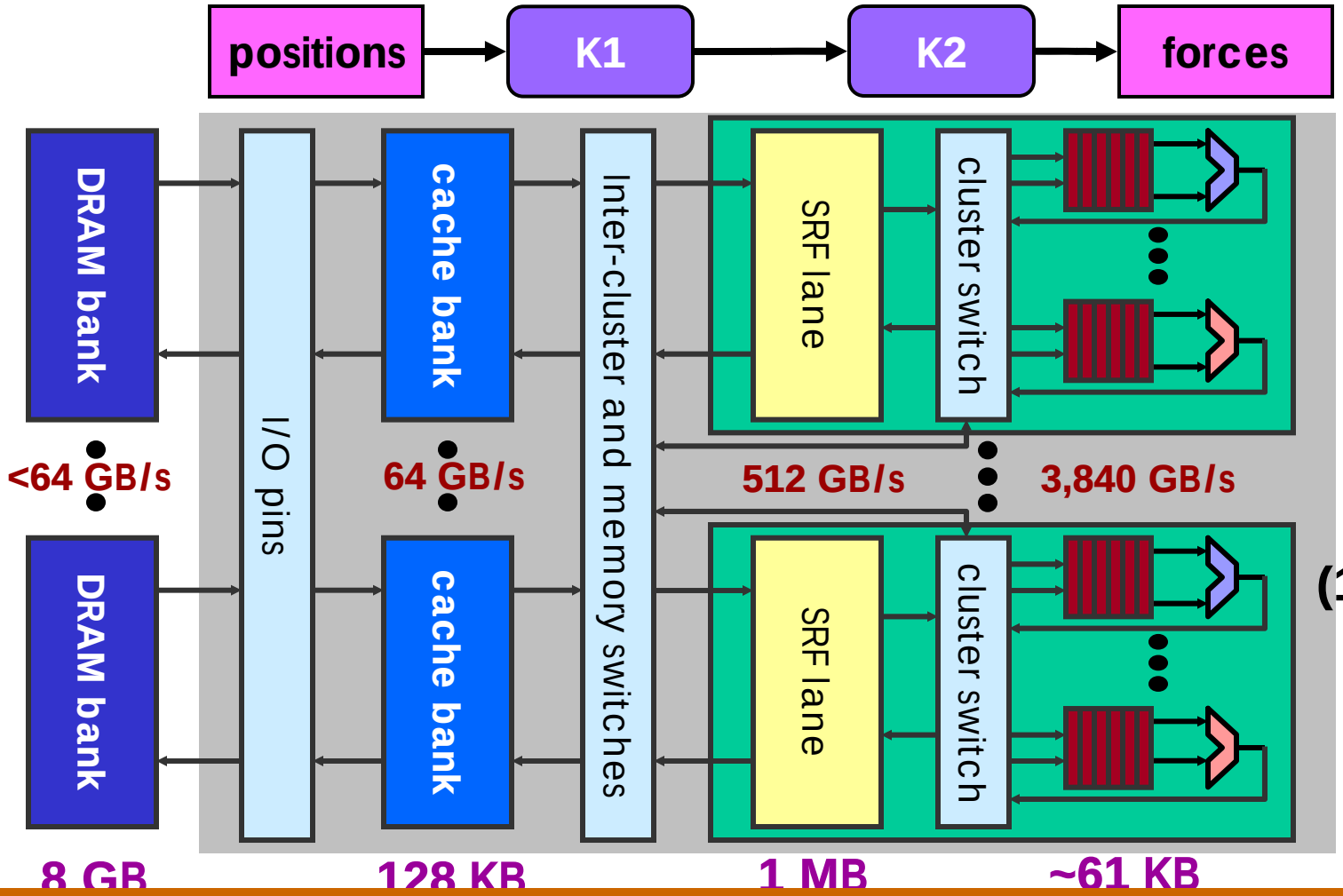
SRF reduces off-chip BW requirements (producer-consumer locality); enables latency-tolerance

Stream Processor Architecture (Merrimac)



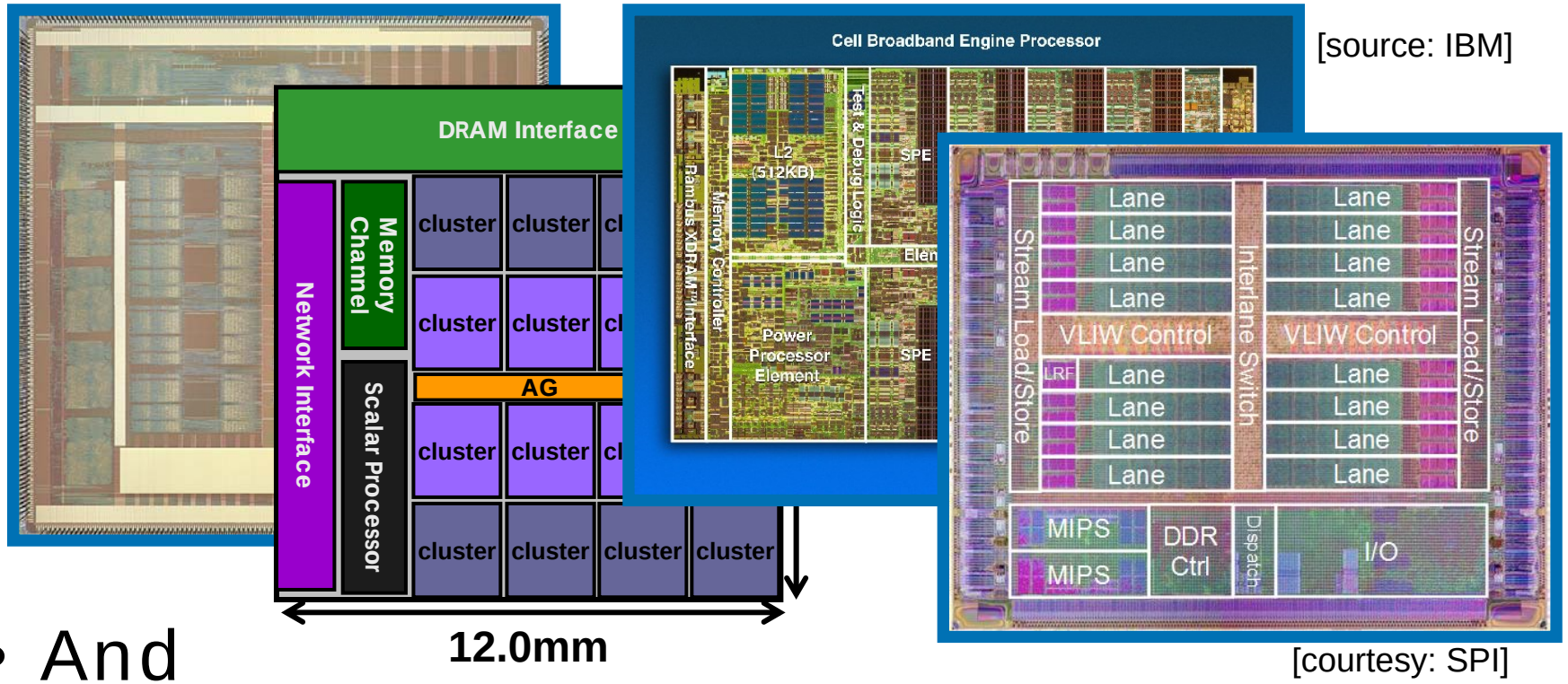
Inter-cluster switch adds flexibility: breaks strict SIMD and assists memory alignment

Stream Processor Architecture (Merrimac)



Cache is a BW amplifier for select accesses

Stream Processors



[source: IBM]

[courtesy: SPI]

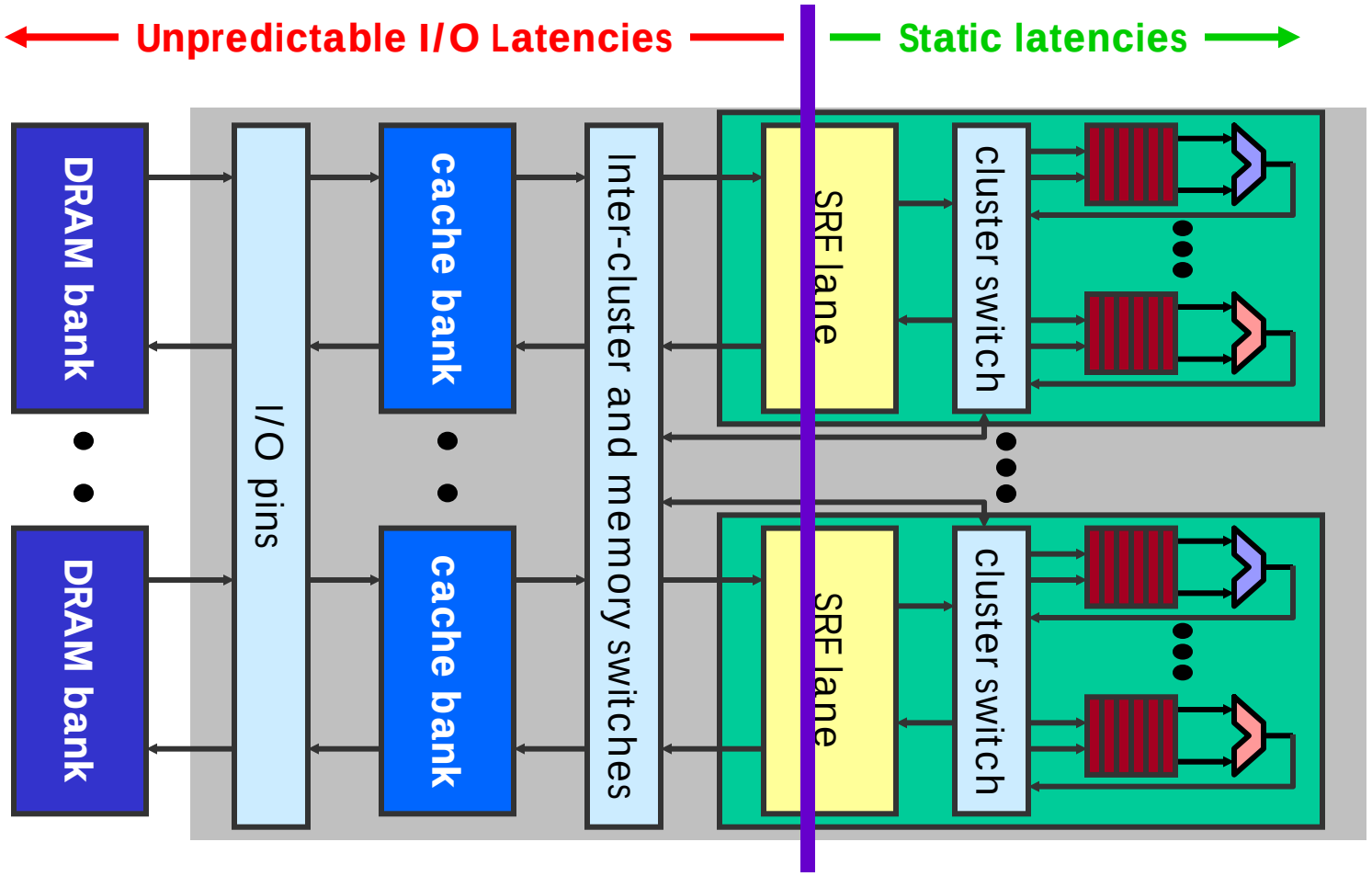
- And
  - ClearSpeed CSX600, MorphoSys, ...
  - GPUs?

Somewhat specialized processors but over a range of applications

Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

SRF Decouples Execution from Memory



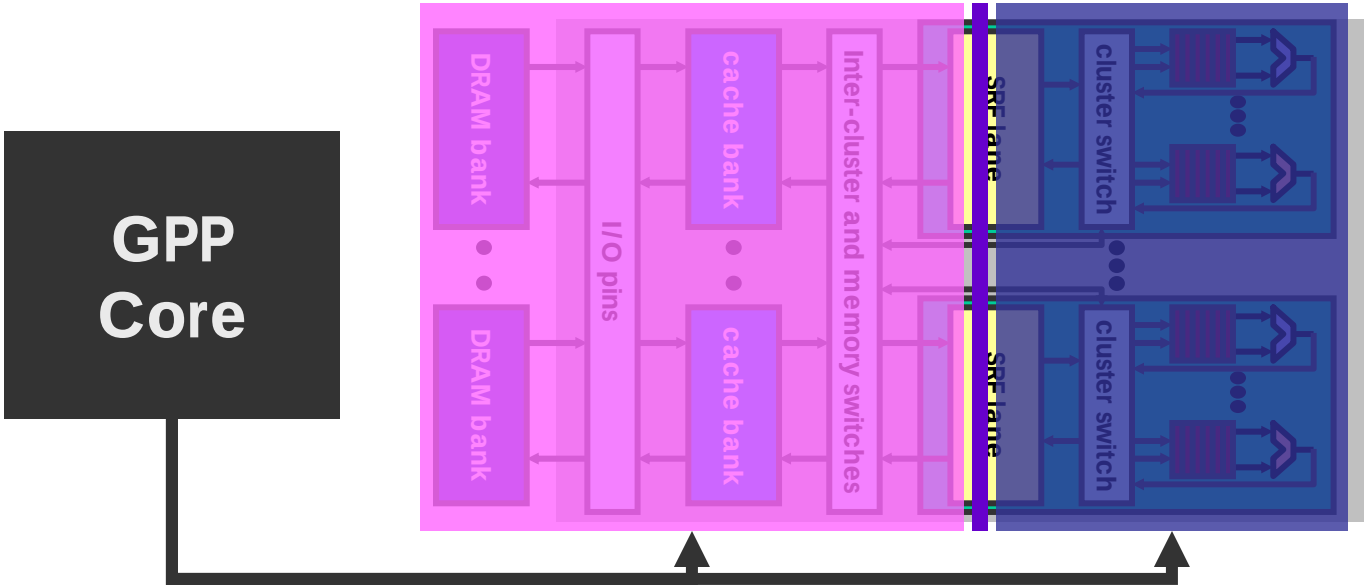
Decoupling enables efficient static architecture
Separate address spaces (MEM/SRF/LRF)

Hierarchical Control

"Scalar" operations

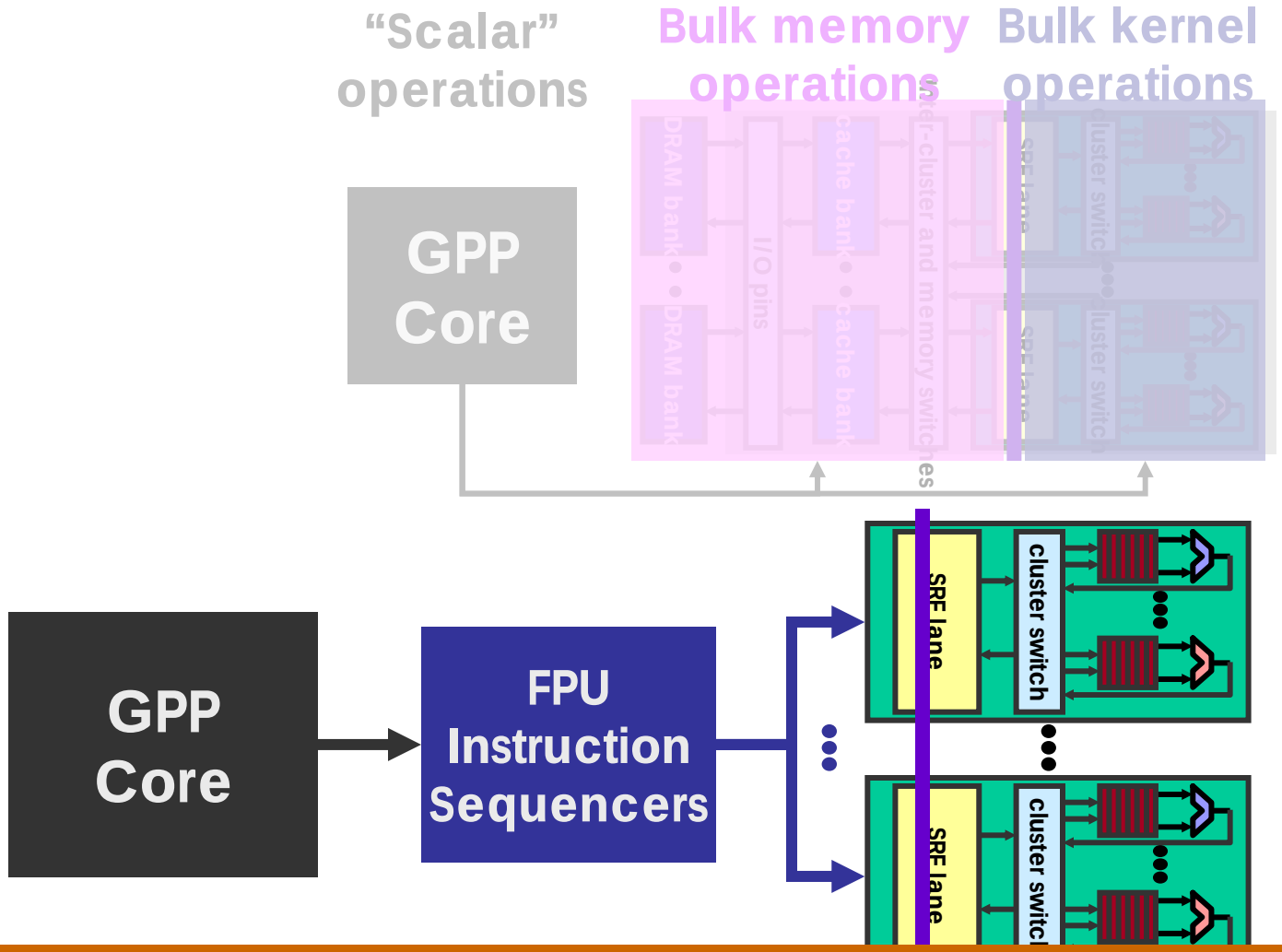
Bulk memory operations

Bulk kernel operations



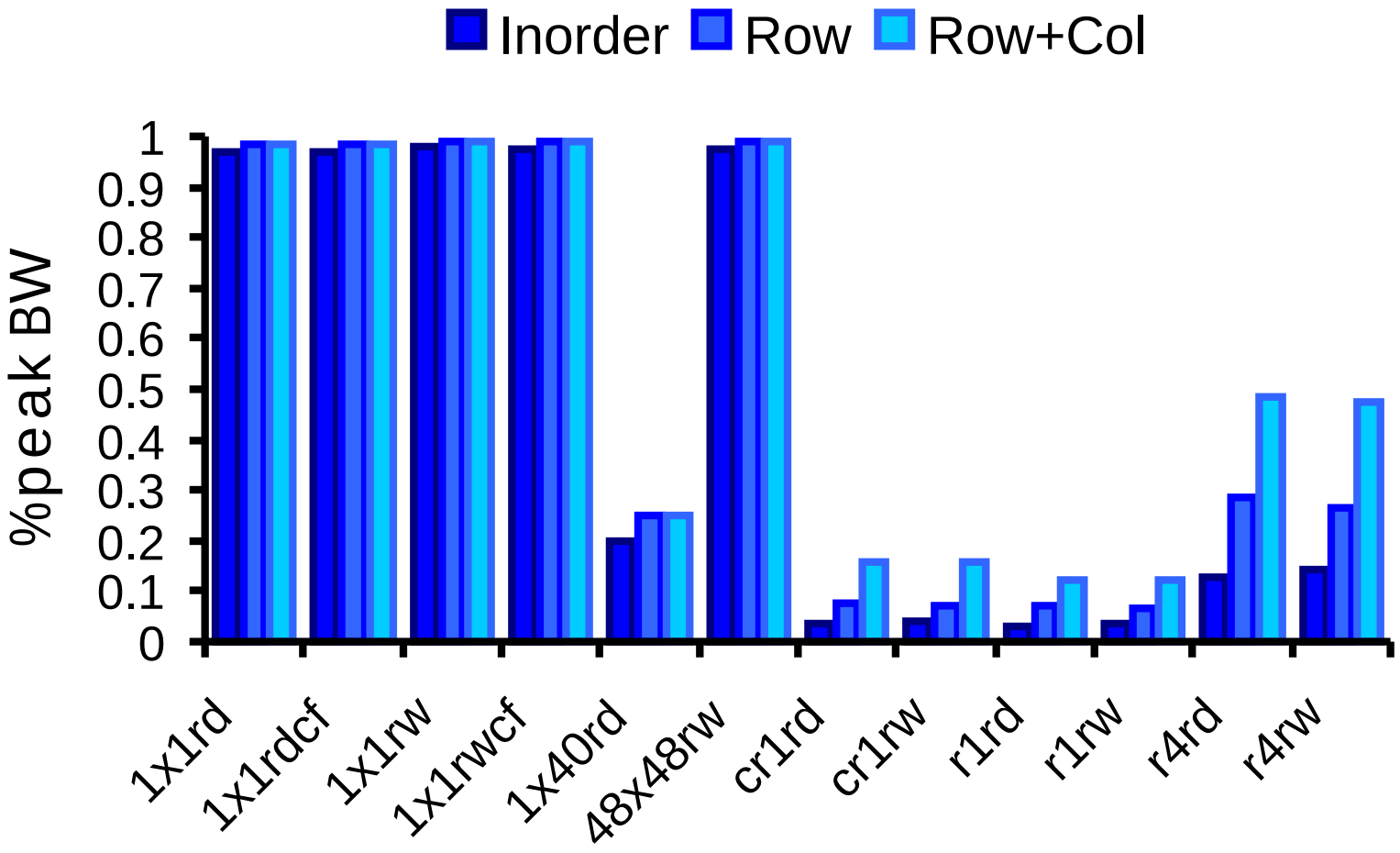
Staging area for bulk operations enables software latency hiding and high-throughput I/O

Hierarchical Control



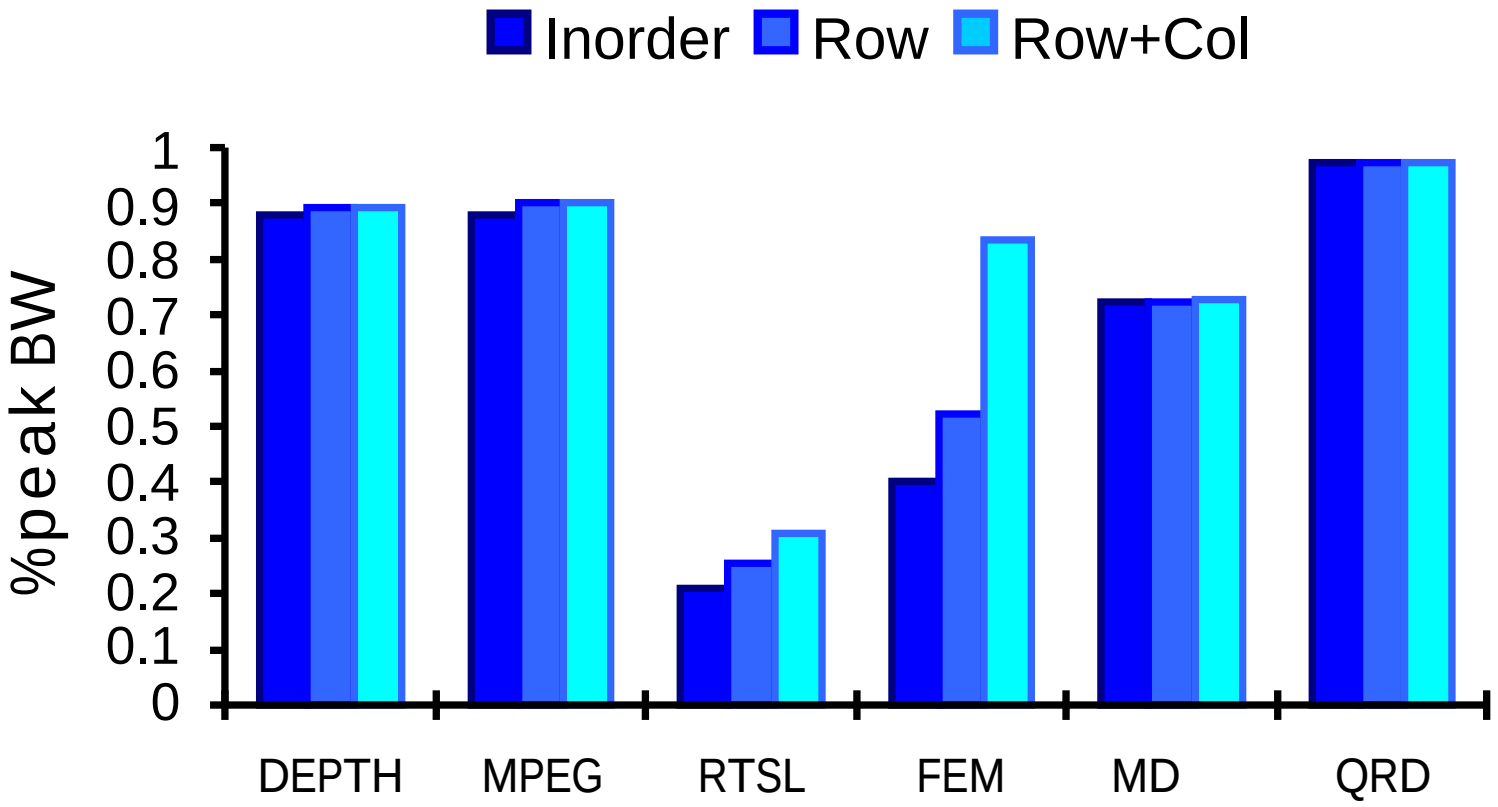
Decoupling allows efficient and effective instruction schedulers

Streaming Memory Systems



DRAM systems are very sensitive to access pattern, Throughput-oriented memory system helps

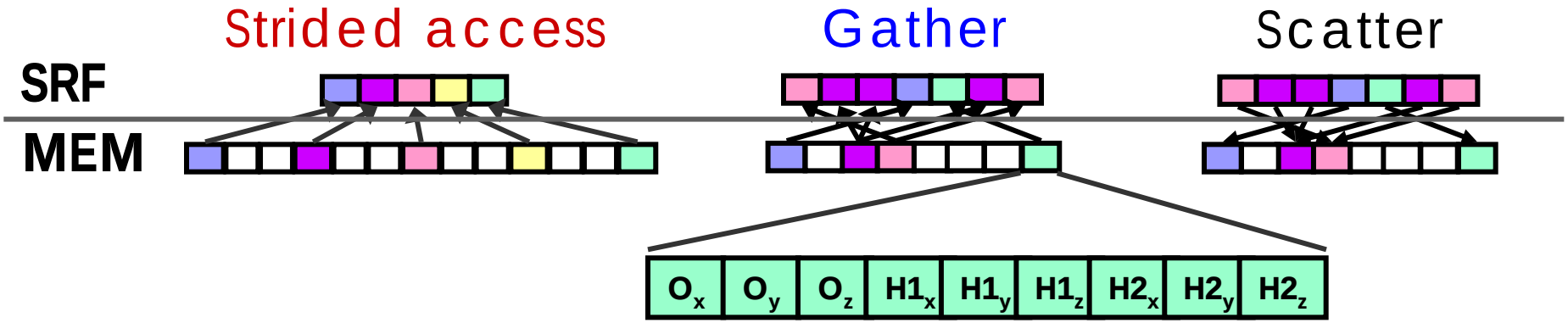
Streaming Memory Systems Help



Capable memory system even more important for applications

Streaming Memory Systems

- Bulk stream loads and stores
  - Hierarchical control
- Expressive and effective addressing modes
  - Can't afford to waste memory bandwidth
  - Use hardware when performance is non-deterministic



- Automatic SIMD alignment
  - Makes SIMD trivial (SIMD ≠ short-vector)

Stream memory system helps the programmer and maximizes I/O throughput

Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

Stream Architecture Features

- Exposed deep locality hierarchy
  - explicit software control over data allocation and data movement
  - flexible on-chip storage for capturing locality
  - staging area for long-latency bulk memory transfers
- Exposed parallelism
  - large number of functional units
  - latency hiding

Stream Architecture Features

- Exposed deep locality hierarchy
  - software managed data movement (communication
- Exposed parallelism
  - large number of functional units and latency hiding

- Predictable instruction latencies
- Optimized static scheduling
- High sustained performance

Stream Architecture Features

- Exposed locality hierarchy
  - software managed data movement
- Exposed parallelism
  - high sustained performance
- **Most instructions manipulate data**
- Minimal hardware control structures
  - no branch prediction
  - no out-of-order execution
  - no trace-cache/decoded cache
  - simple bypass networks
  - …

Efficient hardware → greater software responsibility

Streaming Software Responsible for Parallelism, Locality, and Latency Hiding

- Software explicitly manages locality hierarchy
  - identify bulk transfers and sequence blocks
  - allocate SRF and LRF
- Software explicitly manages parallelism
- Software explicitly manages communication
  - Including pipeline communication
- Schedule for latency hiding (medium-granularity)
- Recode algorithms to stream model
  - expose parallelism
  - expose locality
  - expose structure

Challenges in user/compiler and compiler/hardware interfaces

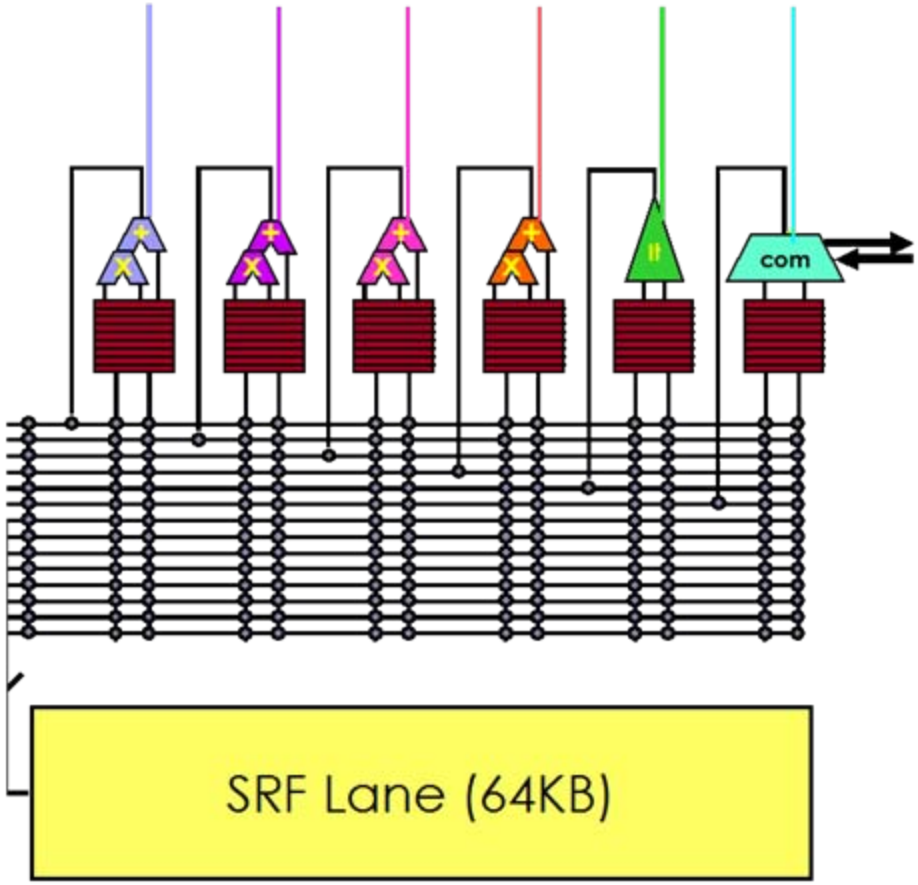
Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

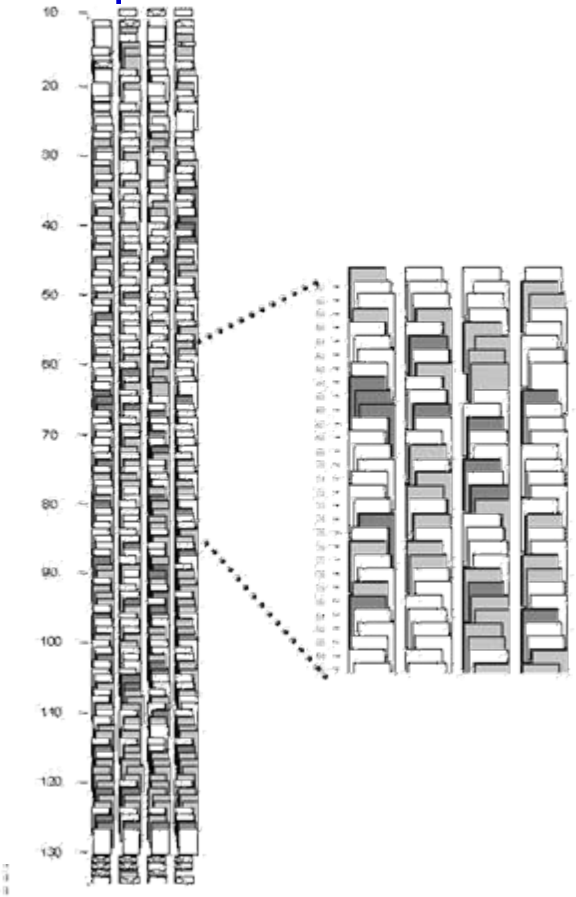
Current State of the Art in Stream Software Systems

- Kernel/Stream 2-level programming model
  - Good kernel scheduling

Compiler Optimizes VLIW Kernel Scheduling



Optimized schedule



Merrimac decouples memory and execution enables static optimization and reduces hardware

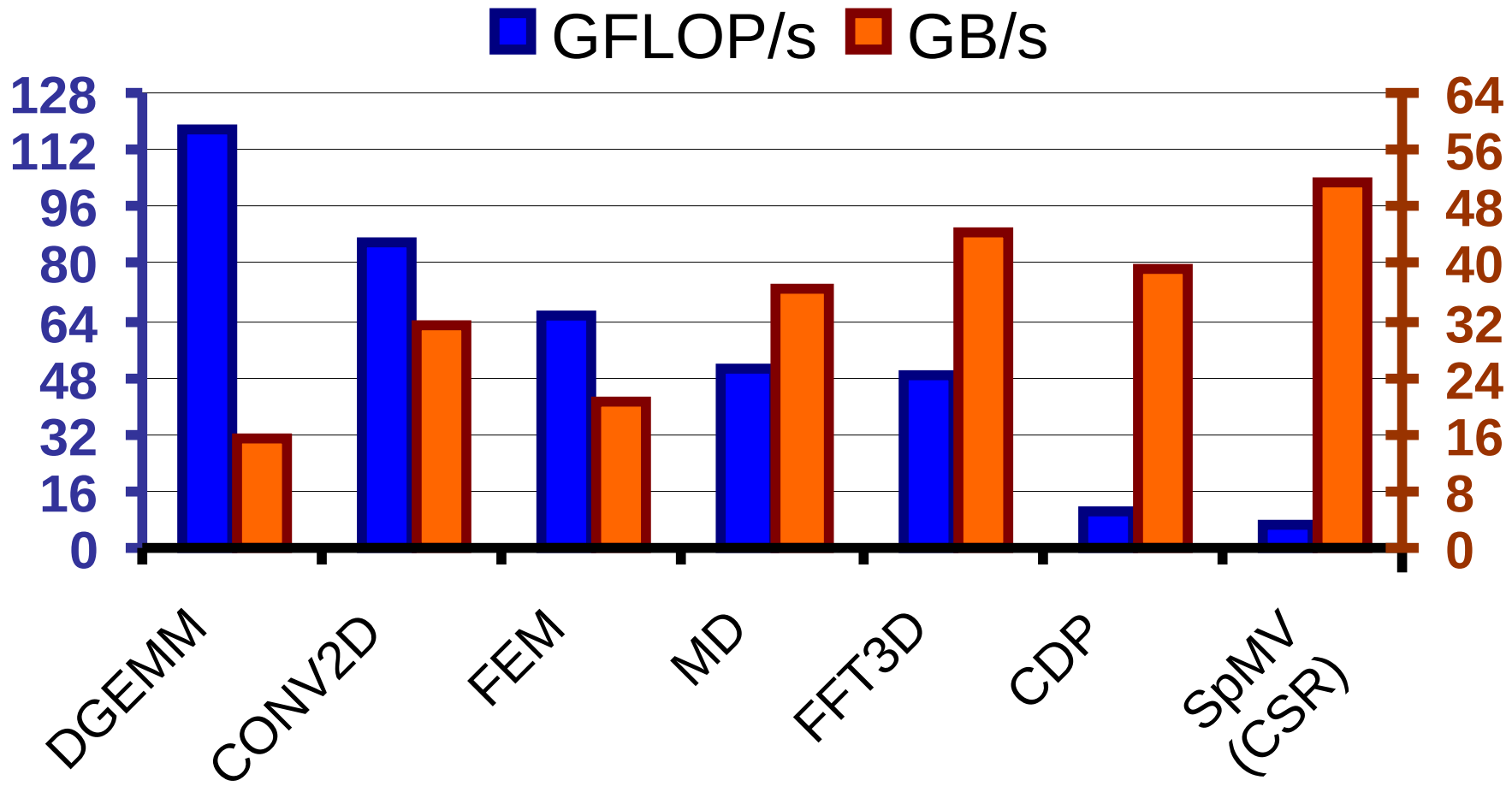
Current State of the Art in Stream* Software Systems

* Stream model as defined earlier

- Kernel/Stream 2-level programming model
  - Good kernel scheduling
  - Decent SRF allocation and stream operation scheduling **IF SIZES KNOWN**
    - Minor success otherwise
- Sequoia
  - Extends to more than 2 levels
- Great auto-tuning opportunities
  - Perfect knowledge of execution pipeline timing
  - Explicit communication
  - Experiments in Sequoia and StreamC

Stream processing simplifies tuning but demands more from the software system and programmer

Results (Simulation)



Explicit stream architecture enables effective resource utilization

What Streams Well?

- Data parallel in general?
- Data – control decoupled algorithms
  - No data→control→data dependence
- Work in progress
  - Traversing data structures in general
  - Dynamic block sizes (data-dependent output rates)
- Later on
  - Building data structures
  - Dynamic data structures

Dynamic mutable data structures will require more tuning over a larger search space

Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

Locality Tradeoffs and Tuning Opportunities

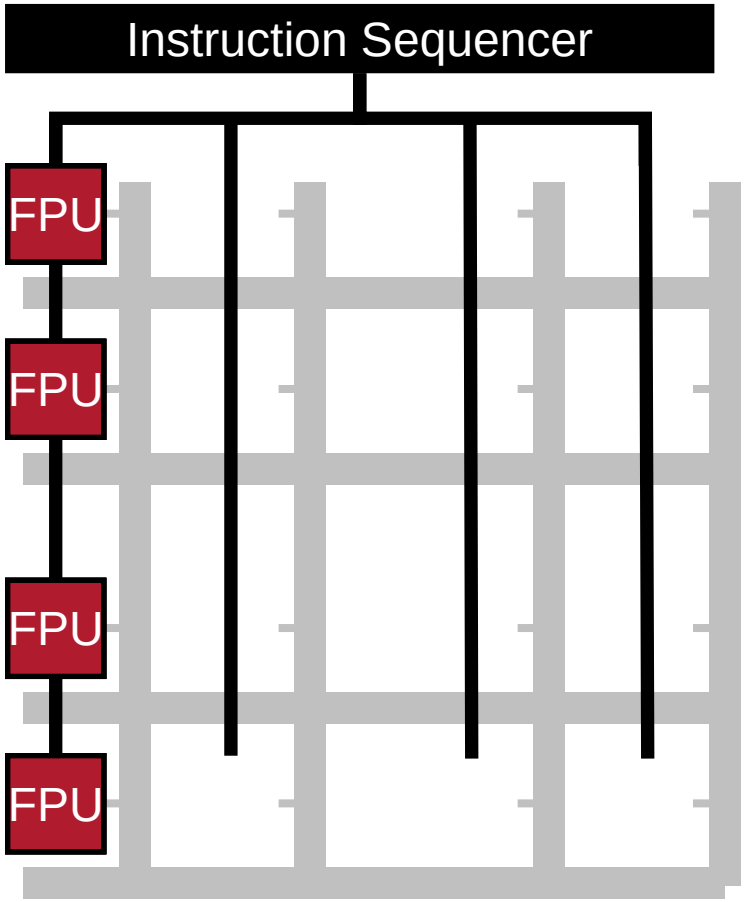
- Register organization
  - Number of registers
  - Connectivity (recall cluster organization)
  - Inter-PE communication
  - Blocking for registers
- Stream register file (local memory)
  - Hardware optimized addressing modes
  - Cross-PE accesses
  - Blocking
- Reactive caching
  - Hardware?
  - Software only?
  - Prefetch and wasted bandwidth issues

So far very ad-hoc decisions in both hardware and software

Parallelism Tradeoffs and Tuning Opportunities

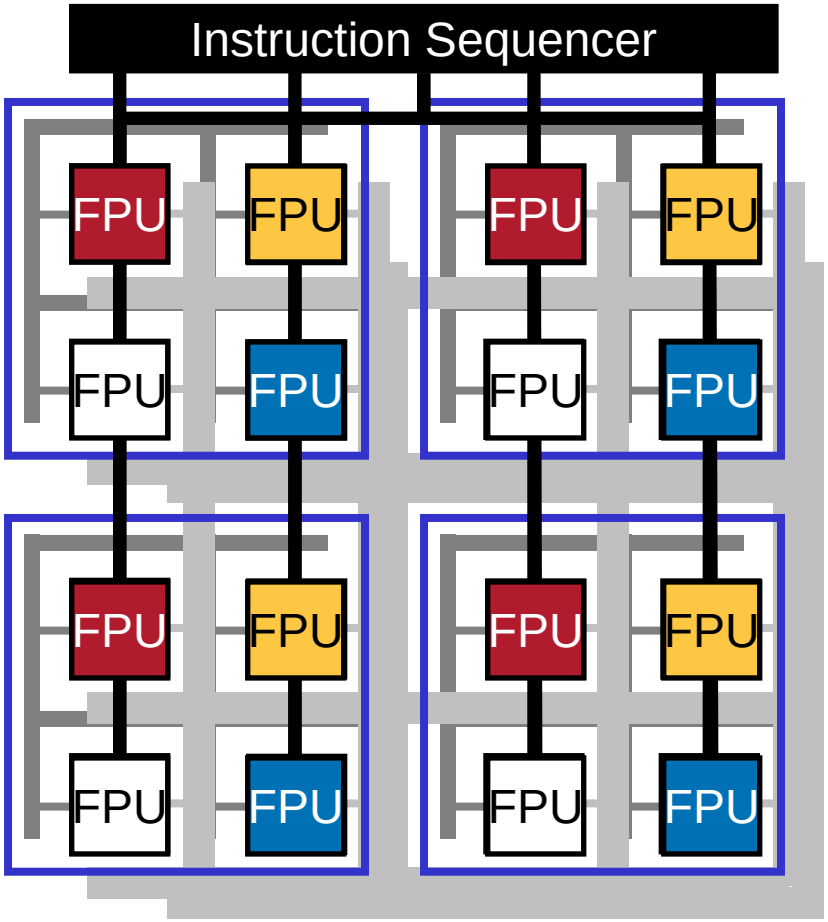
- 3 types of parallelism
  - Data Level Parallelism
  - Instruction Level Parallelism
  - Thread (Task) Level Parallelism

Data-Level Parallelism in Stream Processors



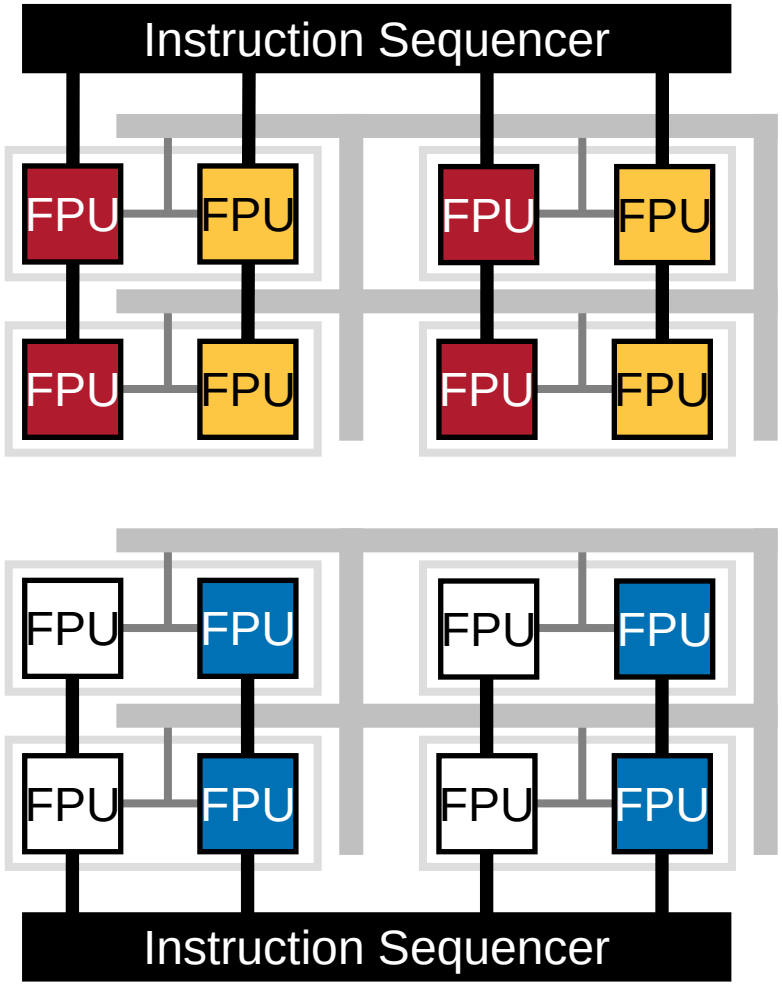
- SIMD
- Independent indexing per FPU
- Full crossbar between FPUs
- No sub-word operation

Data- and Instruction-Level Parallelism in Stream Processors



- A group of FPUs = A Processing Element (PE) = A Cluster
- VLIW
- Hierarchical switch provides area efficiency

Data-, Instruction- and Thread-Level Parallelism in Stream Processors



- Sequencer group
  - Each instruction sequencer runs different kernels

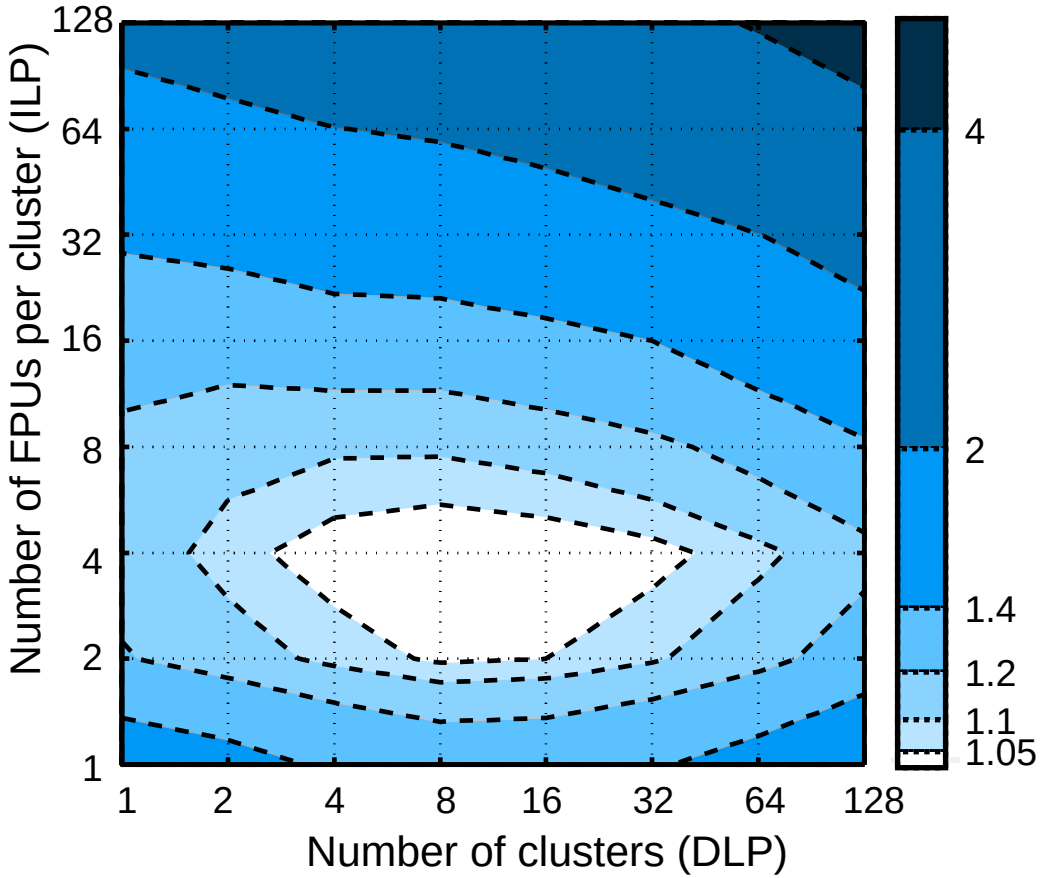
Parallelism Tradeoffs and Tuning Opportunities

- Applications
  - **Throughput oriented** vs. real-time constraint
  - Strong vs. **weak** scaling
  - **Regular vs. irregular**
  - Dynamic / **(practically-)static** datasets
- Hardware
  - DLP: **SIMD**, short vectors
  - ILP: **VLIW / execution pipeline**, OoO
  - TLP: **MIMD**, SMT (style)
  - Communication options
    - Partial switches, direct sequencer-sequencer switch

Hardware models for some options, active research on other options and performance models

Heat-map (Area per FPU) – 64 bit

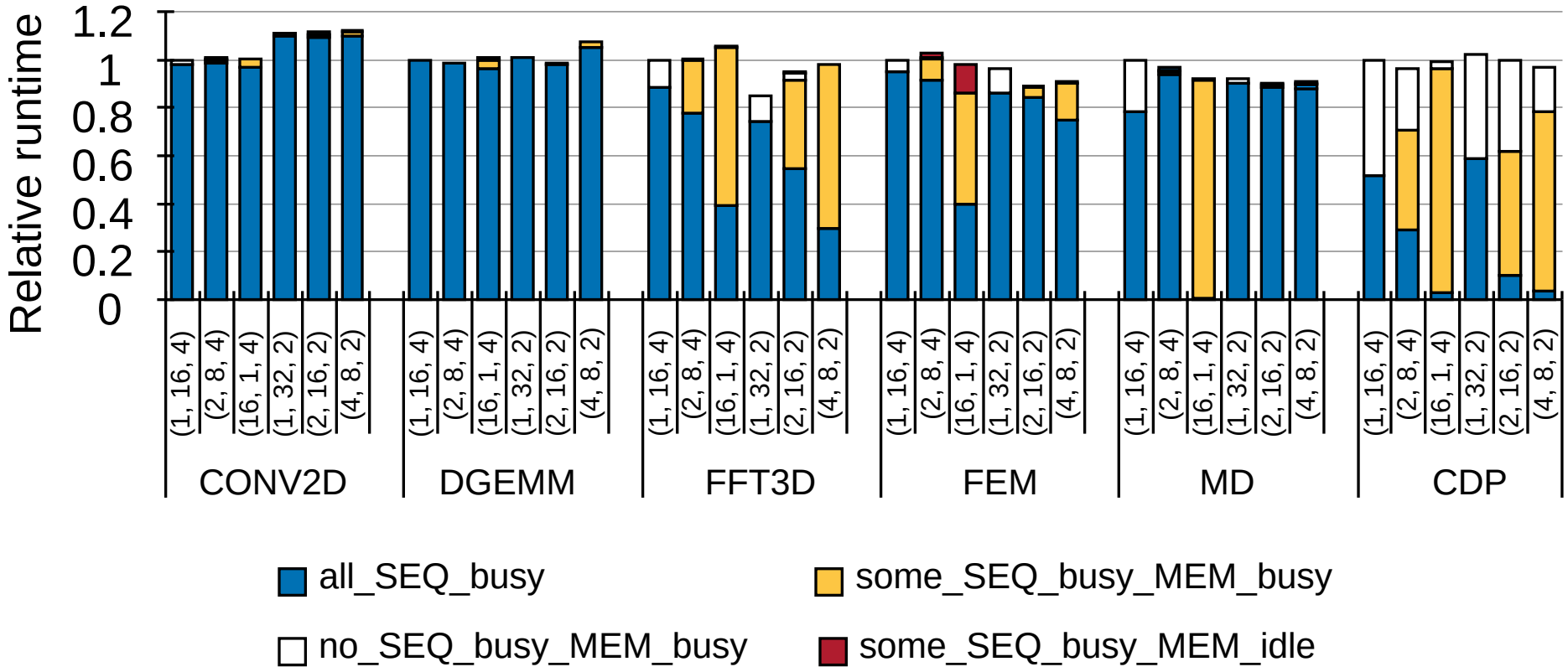
Area overhead of intra-cluster switches



Area overhead of an

Many reasonable hardware options for 64-bit

Application Performance



Small performance differences for "good streaming" applications

Scheduling Tradeoffs and Tuning Opportunities

- Hierarchical scheduling
  - Better suited for auto-tuning
- Hardware support for bulk scheduling?
  - Any software control?
  - Merrimac and Imagine use a hardware score-board at stream instruction granularity (loads/stores or kernels)
- Interaction between scheduling and allocation in general
  - Stream and local register allocation

Very little study on effect of hardware support for bulk scheduling – active research direction

Outline

- Hardware strengths and the stream execution model
- Stream Processor hardware
  - Parallelism
  - Locality
  - Hierarchical control and scheduling
  - Throughput oriented I/O
- Implications on the software system
  - Current status
- HW and SW tradeoffs and tuning options
  - Locality, parallelism, and scheduling
- Petascale implications

Petascale Implications

- Power / energy
  - Closer is better
- Interconnect
  - Smaller diameter
  - Shorter distances?
  - Simpler network topology?
- Reliability
  - Fewer components
  - Interesting fault-tolerance opportunities
- Cost
  - Fewer chips
  - More efficient use of off-chip bandwidth

Hardware makes more sense, tuning makes more sense, recoding is a problem

Conclusions

- Stream Processors offer extreme performance and efficiency
  - rely on software for more efficient hardware
- Empower software through new interfaces
  - Exposed locality hierarchy
  - Exposed communication
  - Hierarchical control
  - Decouple execution pipeline from unpredictable I/O
- Help software when it makes sense
  - Aggressive memory system with SIMD alignment
  - Multiple parallelism mechanisms (can skip short-vectors ☺)
  - Hardware assist for bulk operation dispatch
- Software system heavily utilizes auto-tuning/search

Stream Processors offer path to petascale;
rely on, and are better targets for, automatic tuning