EE382N: Principles in Computer Architecture
Parallelism and Locality
Fall 2009

Lecture 24 – Stream Processors Wrapup + Sony (/Toshiba/IBM) Cell Broadband Engine

Mattan Erez

The University of Texas at Austin

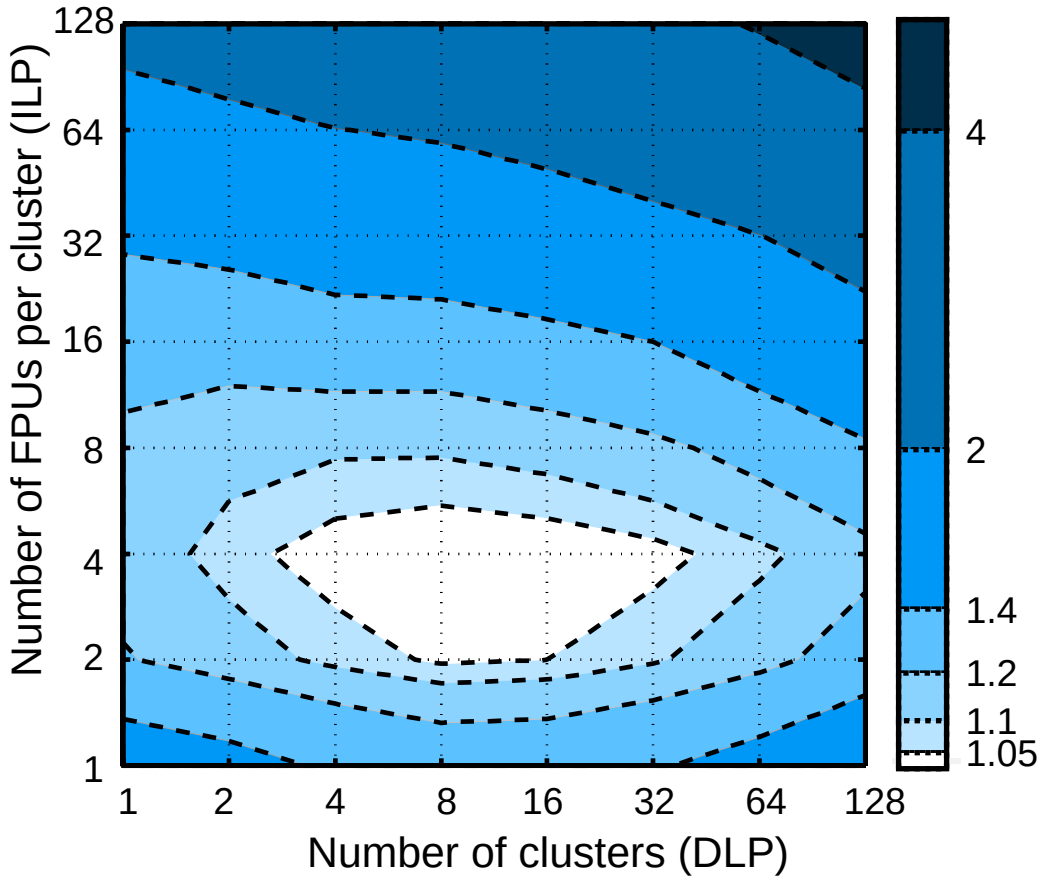
Outline

- Stream Wrapup
  - SRF design
- Motivation
- Cell architecture
  - GPP Controller (PPE)
  - Compute PEs (SPEs)
  - Interconnect (EIB)
  - Memory and I/O
- Comparisons
  - Stream Processors
- Software (probably next time)

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Heat-map (Area per FPU) – 64 bit

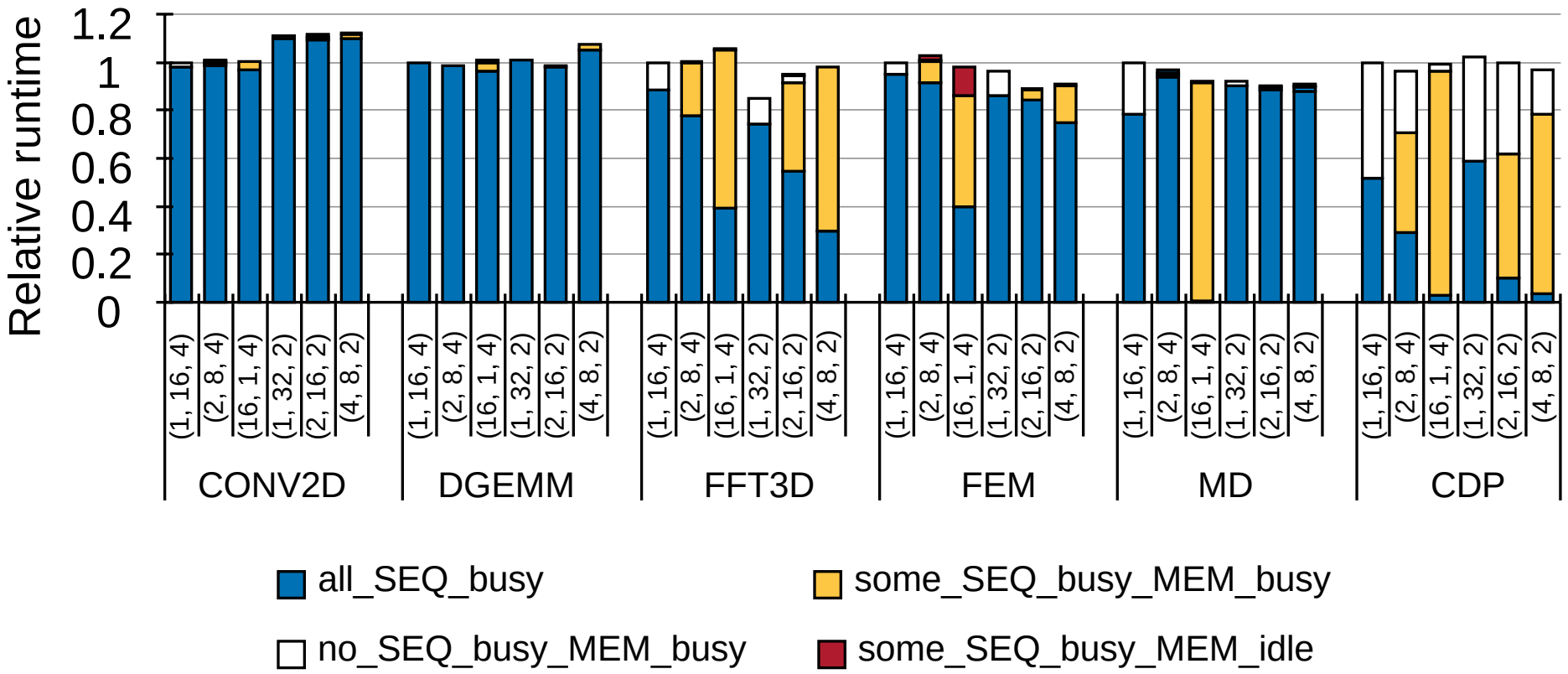
Area overhead of intra-cluster switches



Area overhead of an

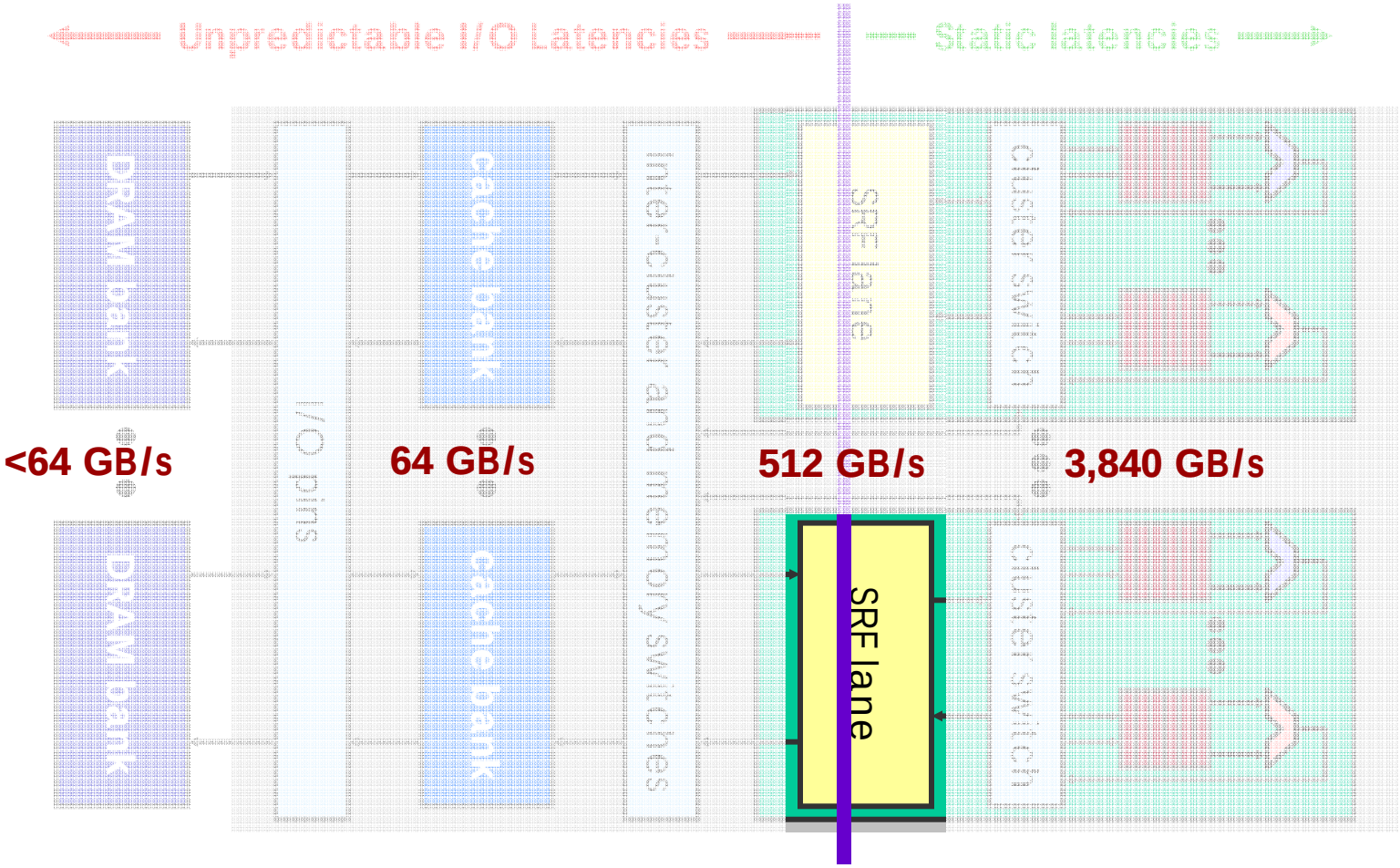
Many reasonable hardware options for 64-bit

Application Performance



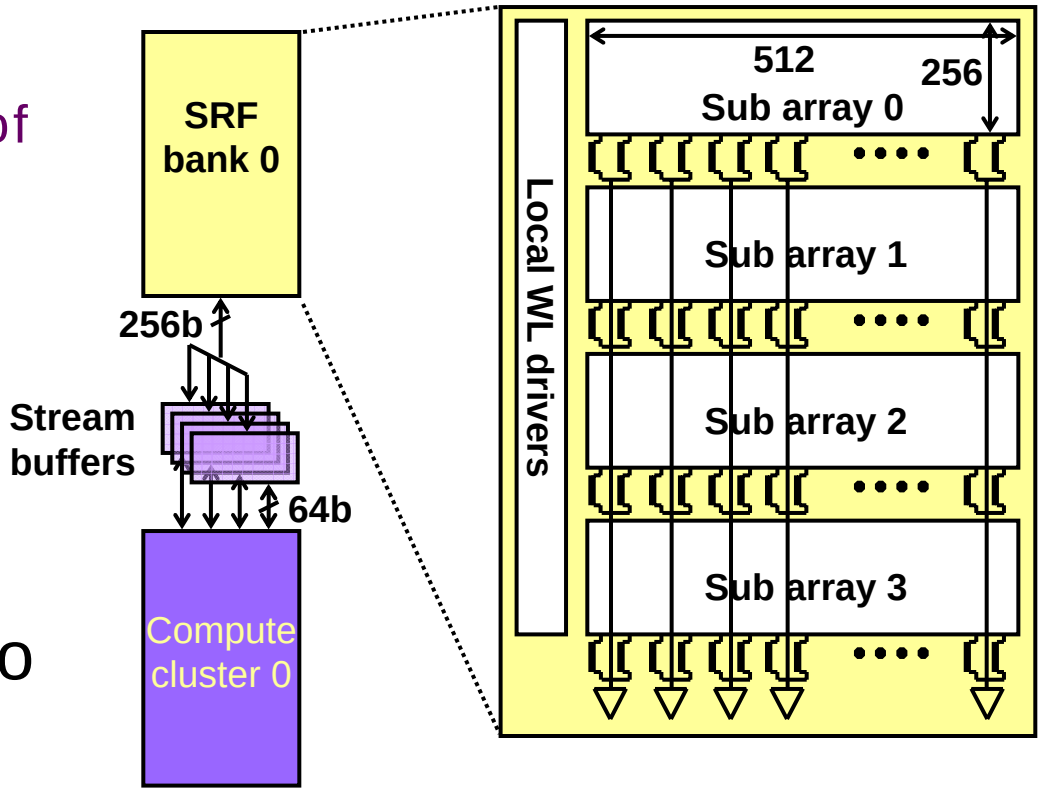
Small performance differences for "good streaming" applications

SRF Decouples Execution from Memory



SRF Sequential Access

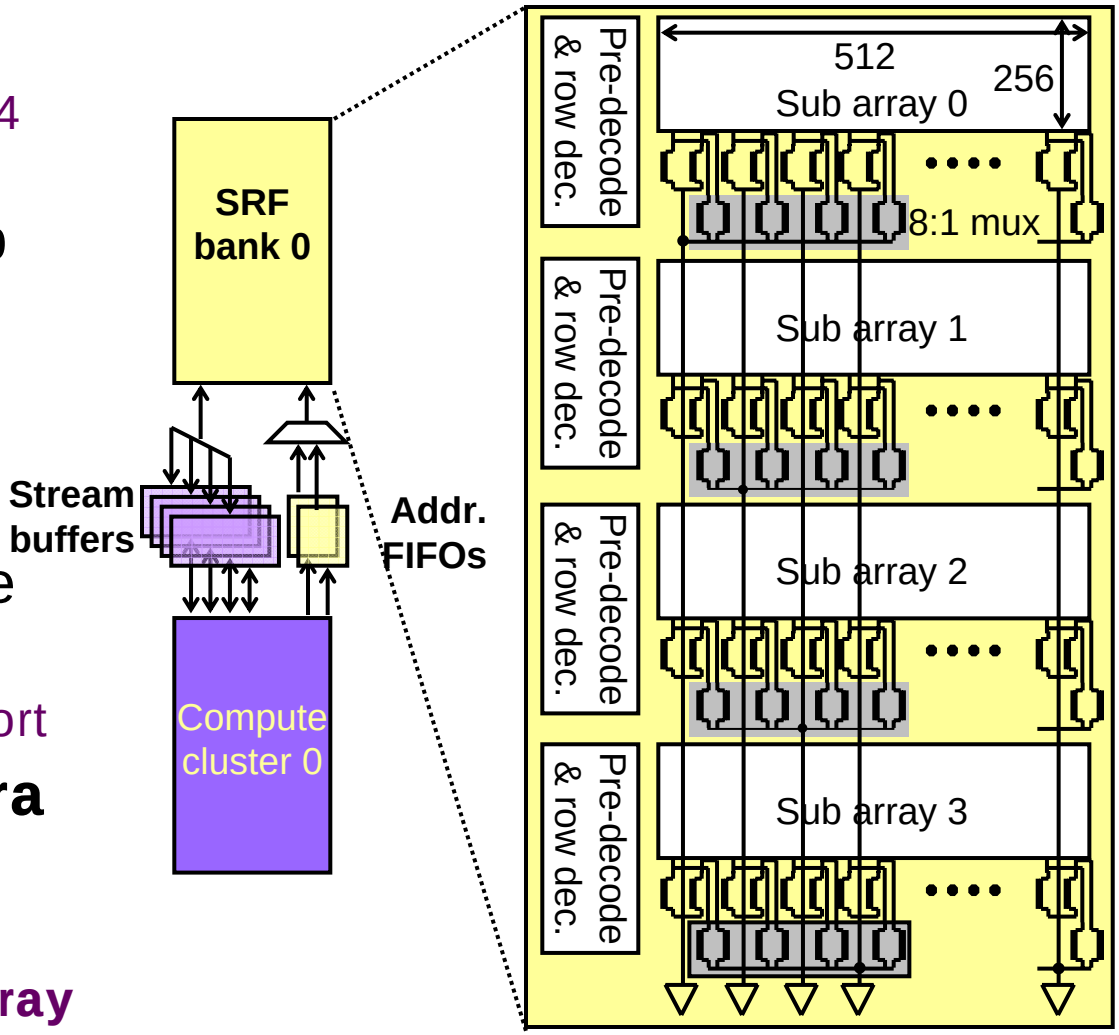
- Single ported memory
  - Efficient wide access of 4 contiguous words
- Implemented using sub arrays
  - Reduced access time
  - Reduced power
- Stream-buffers match bandwidth to compute needs
  - Time multiplex the SRF port



Wide single port time multiplexed by stream buffers

In-lane Indexing Almost Free

- Single ported memory
  - Efficient wide access of 4 contiguous words
- Implemented using sub arrays
  - Reduced access time
  - Reduced power
- Stream-buffers match bandwidth to compute needs
  - Time multiplex the SRF port
- **Indexed SRF at low extra cost**
  - **8:1 MUX in sub-arrays**
  - **Row decoder per sub-array**



Stream Processors and GPUs

Stream Processors

- Bulk kernel computation
  - Kernel uses “scalar” ISA
  - VLIW + SIMD
- Bulk memory operations
  - Software latency hiding
  - Stream mem. System
- HW optimized local mem.
  - Locality opportunities
- Minimize off-chip transfers
  - With capable mem system
- So far mostly load-time parameters

GPUs

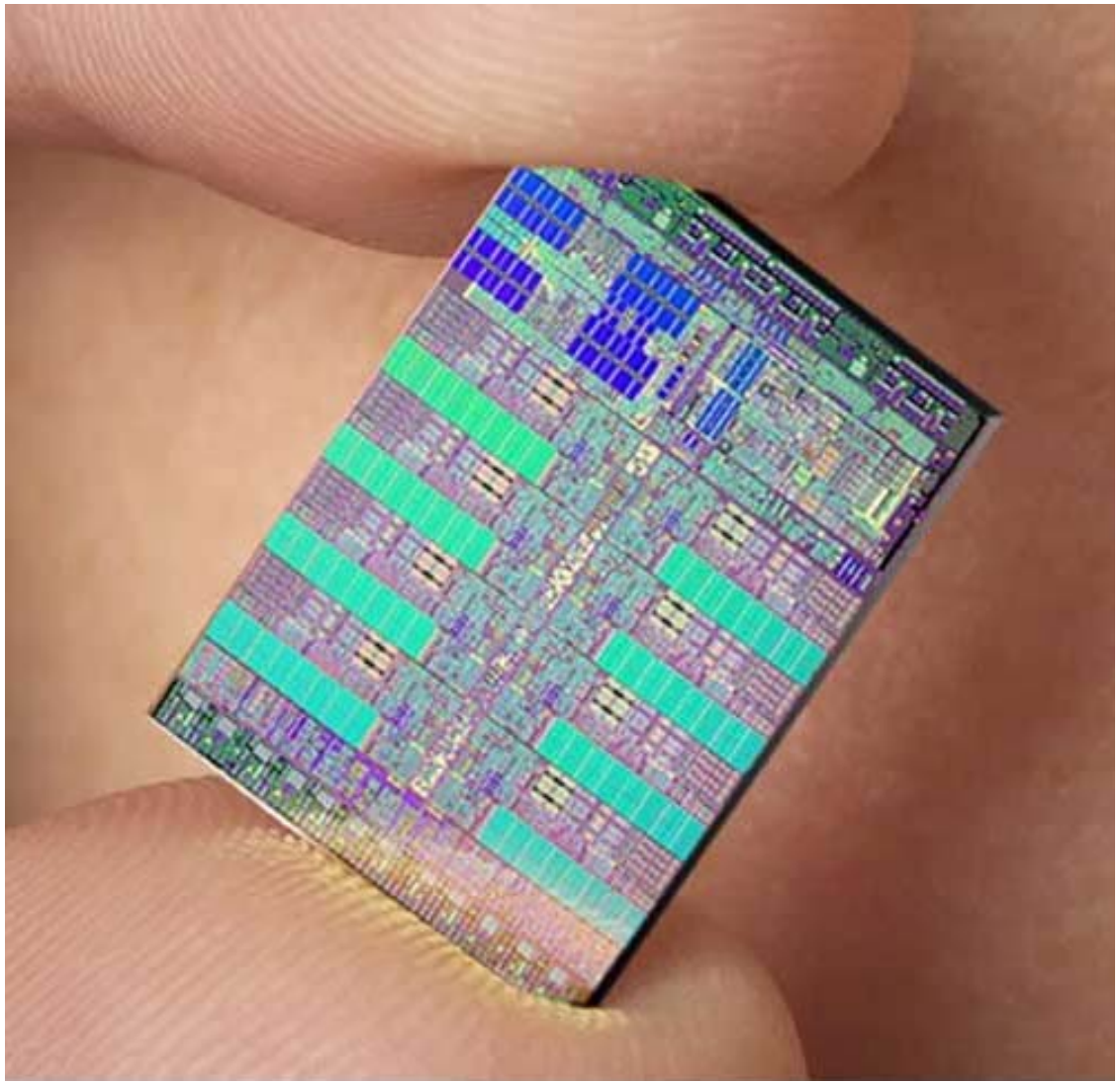
- Bulk kernel computation
  - Kernel uses “scalar” ISA
  - SIMD
- Scalar mem. Operations
  - Threads to hide latency
  - Threads to fill mem. Pipe
- Small shared memory
  - Limited locality
- Rely on off-chip BW
  - Needed for graphics
- Dynamic work-loads
  - Mostly read-only

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
  - Bulk operations and 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

Stream Processors offer programmable and efficiency high performance

Cell Broadband Engine



SONY

IBM

TOSHIBA

Outline

- Motivation
- Cell architecture
  - GPP Controller (PPE)
  - Compute PEs (SPEs)
  - Interconnect (EIB)
  - Memory and I/O
- Comparisons
  - Stream Processors
- Software (probably next time)

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Cell Motivation – Part I

- Performance demanding applications have different characterisitics
  - Parallelism
  - Locality
  - Realtime
- Games, graphics, multimedia …
- Requires redesign of HW and SW to provide efficient high performance
  - Power, memory, frequency walls
- Cell designed specifically for these applications
  - Requirements set by Sony and Toshiba
  - Main design and architecture at IBM

Move to IBM Slides

- Rest of motivation and architecture slides taken directly from talks by Peter Hofstee, IBM
  - Separate PDF file combined from:
    - http://www.hpcaconf.org/hpca11/slides/Cell_Public_Hofstee.pdf
    - http://www.cct.lsu.edu/~estrabd/LACSI2006/workshops/workshop3/Slides/01_Hofstee_Cell.pdf

Outline

- Motivation
- Cell architecture
  - GPP Controller (PPE)
  - Compute PEs (SPEs)
  - Interconnect (EIB)
  - Memory and I/O
- **Comparisons**
  - **Stream Processors**
- Software (probably next time)

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Hardware Efficiency → Greater Software Responsibility

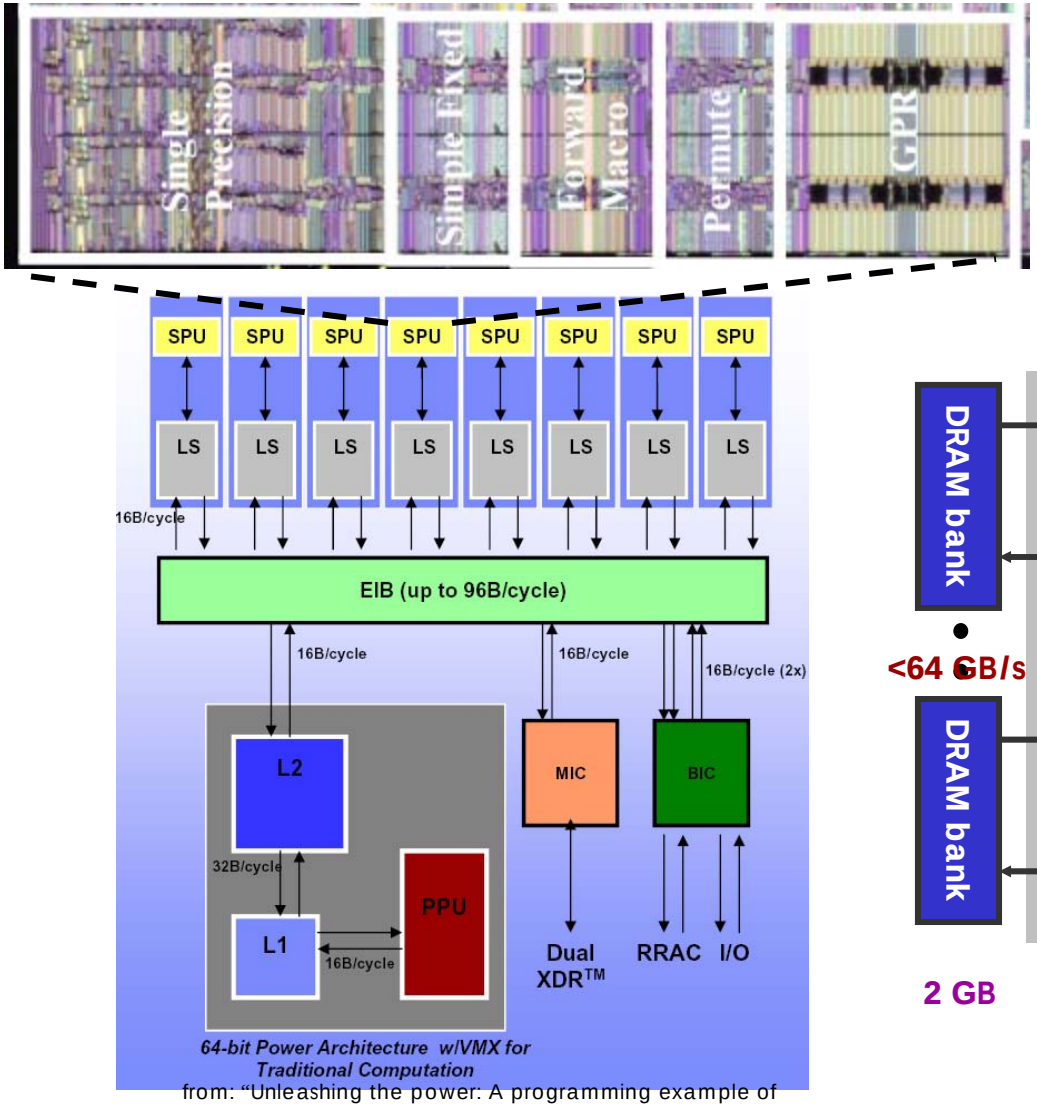
- Hardware matches VLSI strengths
  - Throughput-oriented design
  - Parallelism, locality, and partitioning
  - Hierarchical control to simplify instruction sequencing
  - Minimalistic HW scheduling and allocation

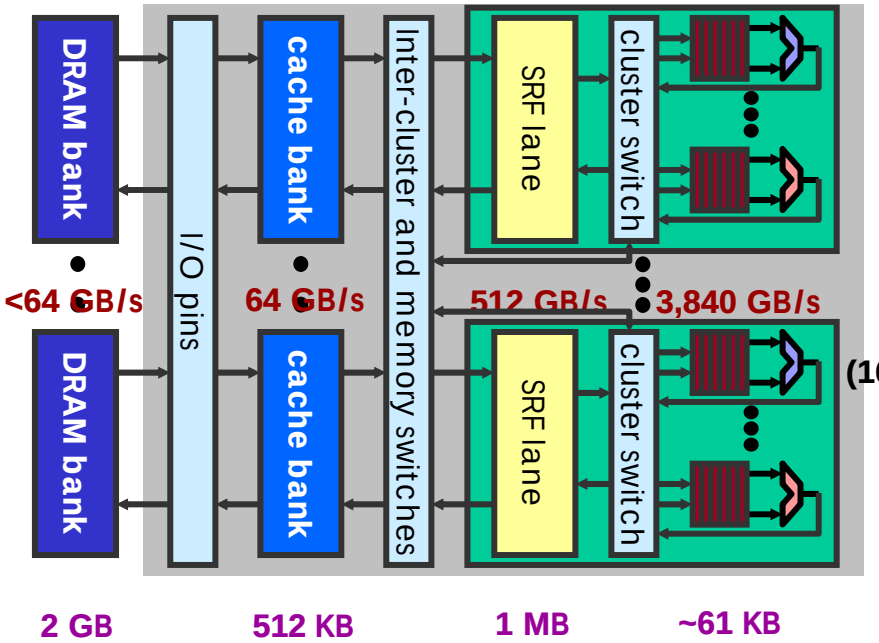
- Software **given** more explicit control
  - Explicit hierarchical scheduling and latency hiding (***schedule***)
  - Explicit parallelism (***parallelize***)
  - Explicit locality management (***localize***)

Must reduce HW "waste" but no free lunch

Locality

Storage/Bandwidth Hierarchy is Key to Efficient High Performance





from: "Unleashing the power: A programming example of large FFTs on Cell" given by Alex Chow at power.org on 6/9/2005

SRF/LS Comparison

- Serve as staging area for memory
- Capture locality as part of the storage hierarchy
- Single time multiplexed wide port
  - kernel access
  - DMA access
  - instruction access
- SPs uses word granularity vs. Cell's 4-word
- SP's SRF has efficient auto-increment access mode
- Cell uses one memory for both code and data
  - Why?

Parallelism

Three Types of Parallelism in Applications

- Instruction level parallelism (ILP)
  - multiple instructions from the same instruction basic-block (loop body) that can execute together
  - true ILP is usually quite limited (~5 - ~20 instructions)
- Task level Parallelism (TLP)
  - separate high-level tasks (different code) that can be run at the same time
  - True TLP very limited (only a few concurrent tasks)
- Data level parallelism (DLP)
  - multiple iterations of a “loop” that can execute concurrently
  - DLP is plentiful in scientific applications

Taking Advantage of ILP

- Multiple FUs (VLIW or superscalar)
  - Cell has limited superscalar (not for FP)
  - Merrimac has 4-wide VLIW FP ops
- Latency tolerance (pipeline parallelism)
  - Cell has 7 FP instructions in flight
  - Merrimac expected to have ~24 FP
  - Merrimac uses VLIW to avoid interlocks and bypass networks
  - Cell also emphasizes static scheduling
    - not clear to what extent dynamic variations are allowed

Taking Advantage of TLP

- Multiple FUs (MIMD)
  - Cell can run a different task (thread) on each SPE + asynchronous DMA on each SPE
    - DMA must be controlled by the SPE kernel
  - Merrimac can run a kernel and DMA concurrently
    - DMAs fully independent of the kernels
- Latency tolerance
  - concurrent execution of **different** kernels and their associated stream memory operations

Taking Advantage of DLP

- Multiple FUs
  - SIMD
    - very (most?) efficient way of utilizing parallelism
    - Cell has 4-wide SIMD
    - Merrimac 16-wide
  - MIMD
    - convert DLP to TLP and use MIMD for different "tasks"
  - VLIW
    - convert DLP to ILP and use VLIW (unrolling, SWP)
- Latency tolerance
  - Overlap memory operations and kernel execution (SWP and unrolling)
  - Take advantage of pipeline parallelism in memory

Memory System

High Bandwidth Asynchronous DMA

- Very high bandwidth memory system
  - need to keep FUs busy even with storage hierarchy
  - Cell has ~2 words/cycle (25.6GB/s)
  - Merrimac designed for 4 words/cycle
- Sophisticated DMA
  - stride (with records)
  - gather/scatter (with records)
- Differences in granularity of DMA control
  - Merrimac treats DMA as stream level operations
  - Cell treats DMA as kernel level operations

Design Choices Discussion

Is VLIW a Good Idea?

- Want to take advantage of all available ILP
  - Merrimac has a shallower pipeline than Cell
- Efficiency
  - allows exposed communication at the kernel level
  - no need for interlocks (or at least cheaper)
  - no bypass network
- Manage a larger register space
  - distributed/clustered register file
  - allows relaxing SIMD restrictions with communication
- VLIW code expansion can be a problem
  - especially for MIMD
  - requires recompilation
    - interlocks allow correct (possibly inefficient) execution

Is MIMD a Good Idea?

- Probably yes for limited parallelism
  - true for PS3, probably not for scientific applications
- May help irregular applications
  - work well on Merrimac with relaxed-SIMD but there are overheads
    - padding, extra node copies, SIMD conditionals overhead
- Multiple sequencers and extra instruction storage
  - instructions storage might be saved by pipelining kernels
    - not always applicable, might increase bandwidth
  - Cell sequencer seems to be ~12% of SPE

Is Kernel-Level DMA Control a Good Idea?

- MIMD (almost) dictates kernel level control
- Consumes issue slots
- Works well for fixed-rate streams
- Requires multi-threading for irregular stream access
  - stream-level DMA lets hardware handle synchronization more efficiently
- May allow pointer-chasing?
  - only real advantage is making the way-too-long memory latency loop a little shorter by localizing control

Outline

- Motivation
- Cell architecture
  - GPP Controller (PPE)
  - Compute PEs (SPEs)
  - Interconnect (EIB)
  - Memory and I/O
- Comparisons
  - Merrimac
- **Software**

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Parallelism in Cell

- PPE has SIMD VMX instructions and is 2-way SMT
- 8 SPEs
  - Each SPE operates exclusively on 4-vectors
  - Odd/even instruction pipes
    - Odd for branching and memory / Even for compute
  - Pipelining for taking advantage of ILP
- Asynchronous DMAs
  - SWP of communication and computation
  - Memory-level parallelism

Locality in Cell

- PPE has L1 and L2 caches
  - L2 cache is 512KB
- SPE have lots of registers and local store
  - 128 registers per SPU
  - 256KB LS per SPU

Communication and Synchronization in Cell

- Element Interconnect Bus (EIB)
  - Carries all data between memory system and PPE/SPEs
  - Carries all data between SPE→SPE, SPE→PPE, and PPE→SPE
- PPE L2 cache is coherent with memory
- SPEs are not coherent (LS is not a cache)
  - SPE DMA coherent with L2
- SPEs can DMA to/from one another
  - Memory map LS space into global namespace
  - Each SPE has local virtual memory translation to do this
  - No automatic coherency – explicit synchronization
- Synchronization through memory or mailboxes

Cell Software Challenges

- Separate code for PPE and SPEs
  - Explicit synchronization
- SPEs can only access memory through DMAs
  - DMA is asynchronous, but prep instructions are part of SPE code
  - SW responsible for consistency and coherency
- SPEs must be programmed with SIMD
  - Lots of pipeline challenges left up to programmer / compiler
    - Deep pipeline with no branch predictor
    - 2-wide scalar pipeline needs static scheduling
    - LS shared by DMA, instruction fetch, and SIMD LD/ST
    - No memory protection on LS (Stack can "eat" data or code)
- Most common programming system is just low-level API and intrinsics – Cell SDK
  - Luckily, other options exist

Separate Code for PPE and SPEs

- PPE should be doing top-level control and I/O only
  - Trying to compute on the PPE is problematic
    - In order 2-way SMT with deep pipeline
  - Busy enough just trying to coordinate between the SPEs
- SPEs are the parallel compute engines
  - PowerPC architecture, but really different ISA and requirements
  - Vector only, 2-way superscalar, …
- Two processor types really have two different software systems
- Synchronization can be tricky to get efficient
  - PPE → SPE should use mailbox
  - SPE → PPE should use L2 cache location
  - Why?
- Spawning threads on SPEs painfully slow

Memory System Challenges

- No arbitrary global LD/ST from SPE
  - Everything must go through DMA
- Strict DMA alignment and granularity restrictions
  - All DMAs **must** be aligned to 16-bytes and request at least 16 bytes
  - All DMAs ***should*** be aligned to 128-bytes and request at least 128 bytes of contiguous memory
  - **Failing to maintain alignment will cause a bus error!**
- Banked DRAM structure
  - Consecutive DMA requests should span 2KB (8 banks)
- DMA commands issued from within SPE
  - Strided data access
  - Chain of stride commands to have arbitrary gather
    - Chained list of commands prepared by SPE and stored in LS
- Need to explicitly synchronize with DMA completion

SPE Programming Challenges

- Vector only
  - Scalars must be placed in LSW of vector and some instructions will take this into account
  - All casts and operations are aligned to 16 bytes (4 words)
  - Poor integer→float casting
- LS is only SRAM in SPE
  - Careful with growing stack
  - Sometimes need to force instruction prefetch to avoid stalls
    - Priority given to DMA first, then LD/ST, and only then fetch
- No branch prediction (predict not-taken?)
  - Can have branch hint instructions inserted explicitly
- Some help from XLC compiler (and an occasional bug)

Overall

- Lots to deal with
- Explicit communication and synchronization
- Explicit locality
- Explicit parallelism
- Explicit pipeline scheduling

- Need some help from programming tools