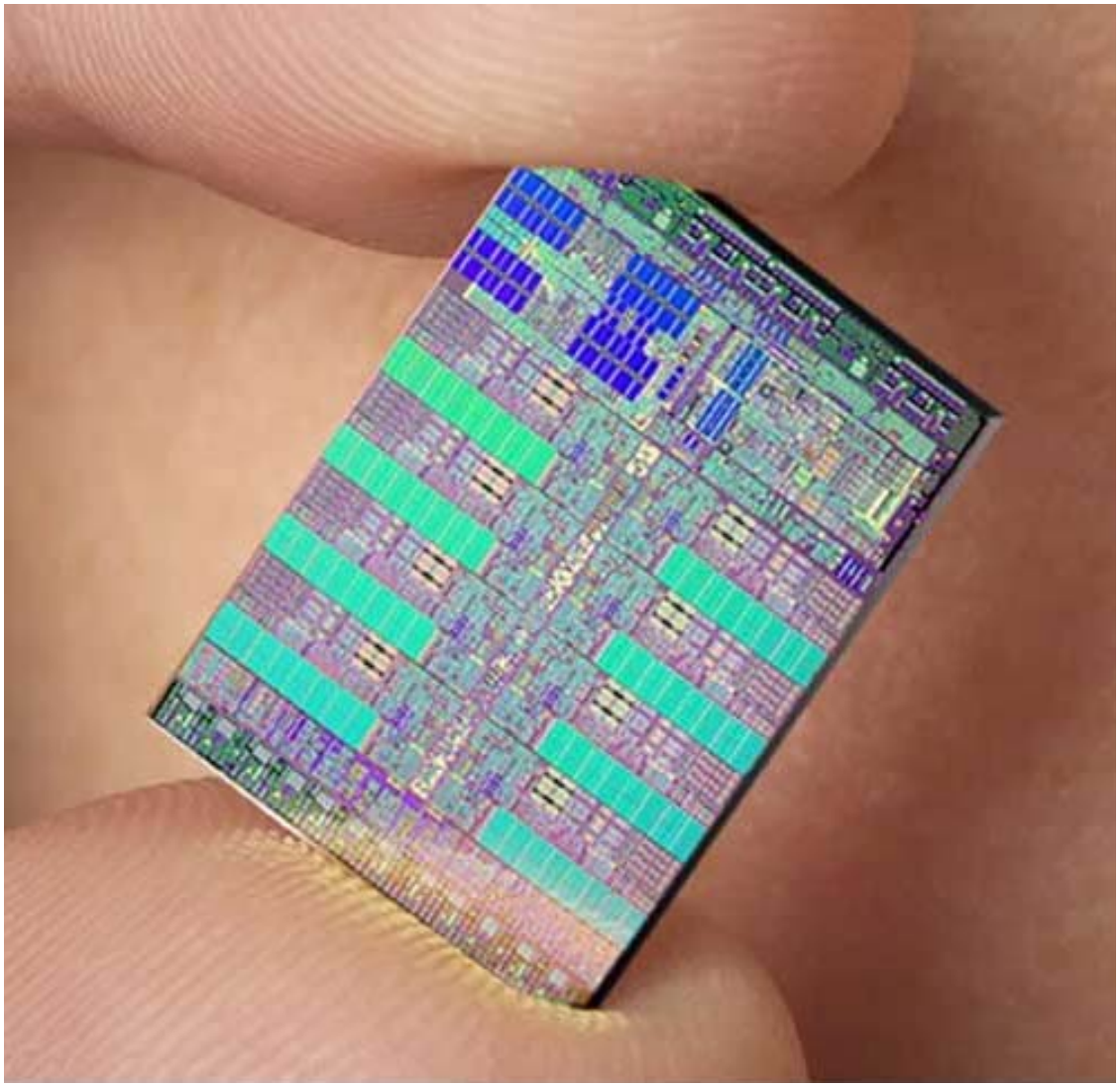
EE382V (17325): Principles in Computer Architecture
Parallelism and Locality
Fall 2007

Lecture 19 – Sony (/Toshiba/IBM) Cell Broadband Engine

Mattan Erez

The University of Texas at Austin

Cell Broadband Engine



SONY

IBM

TOSHIBA

Outline

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

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

Hardware Efficiency → Greater Software Responsibility

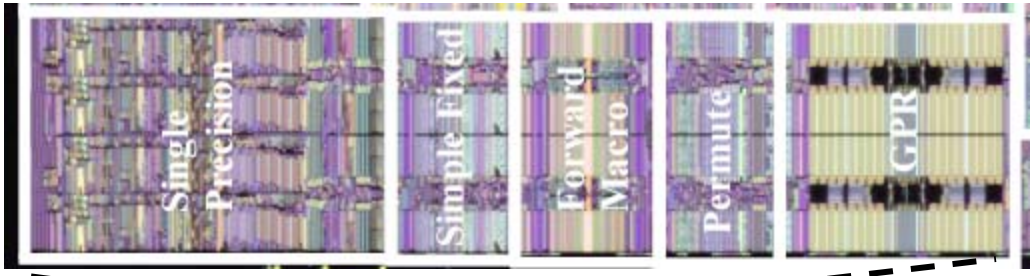
- Hardware matches VLSI strengths
  - Throughput-oriented design
  - Parallelism, locality, and partitioning
  - Hierarchical control to simplify instruction sequencing
  - Minimalistic HW scheduling and allocation

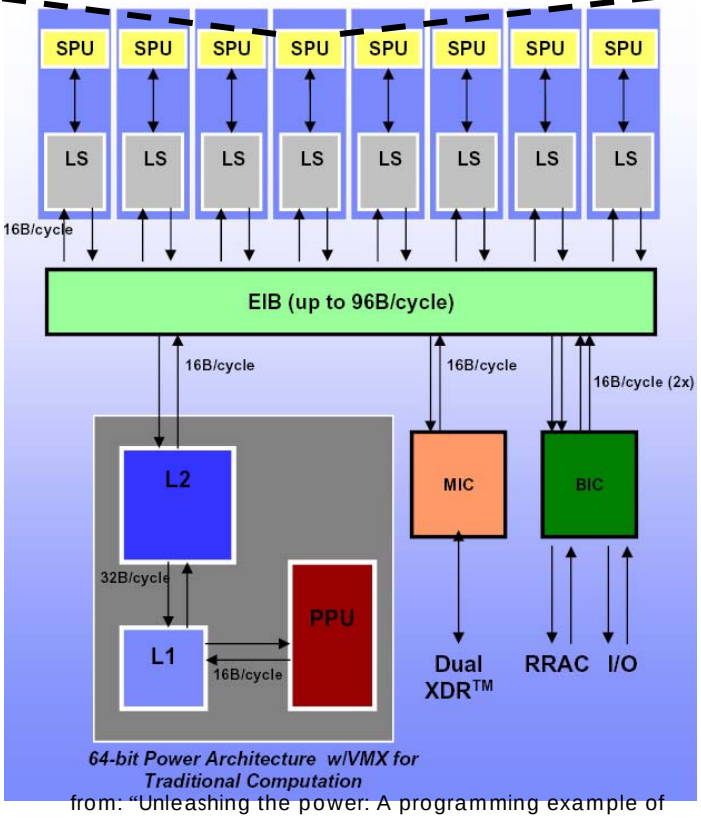
- Software **given** more explicit control
  - Explicit hierarchical scheduling and latency hiding (***schedule***)
  - Explicit parallelism (***parallelize***)
  - Explicit locality management (***localize***)

Must reduce HW "waste" but no free lunch

Locality

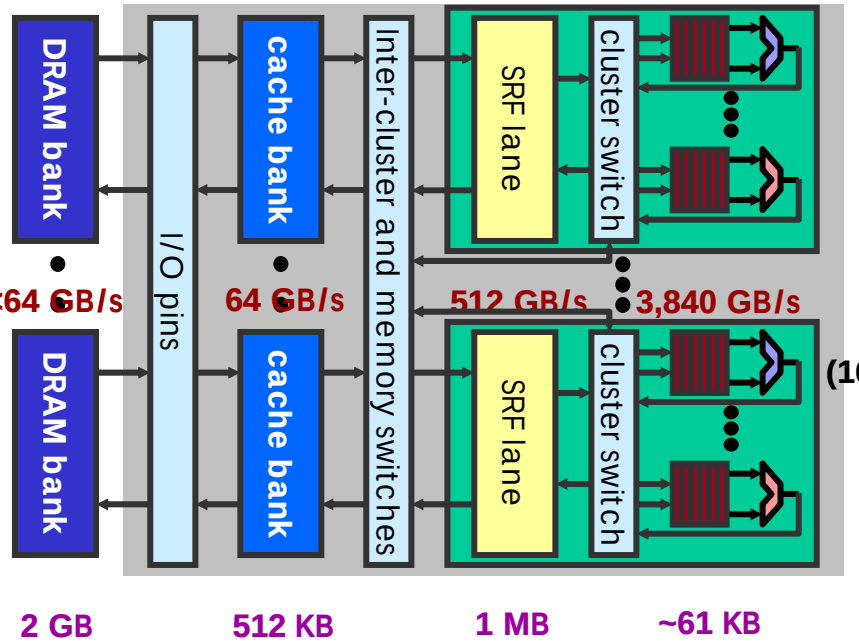
Storage/Bandwidth Hierarchy is Key to Efficient High Performance





64-bit Power Architecture w/VMX for Traditional Computation

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
- Merrimac uses word granularity vs. Cell's 4-word
- Merrimac's SRF has efficient auto-increment access mode
- Cell uses one memory for both code and data
  - Why?

Inter-Cluster Communication Comparison

- Merrimac allows 16 + 8 words per cycle
  - 1 word per cluster using full cross bar
- Cell allows 3x8 words per cycle peak
  - 4 uni-directional rings
  - sustained performance may be somewhat lower
  - scalable architecture requiring little to no re-design

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 (probably next time)**

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


UT
ECE

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