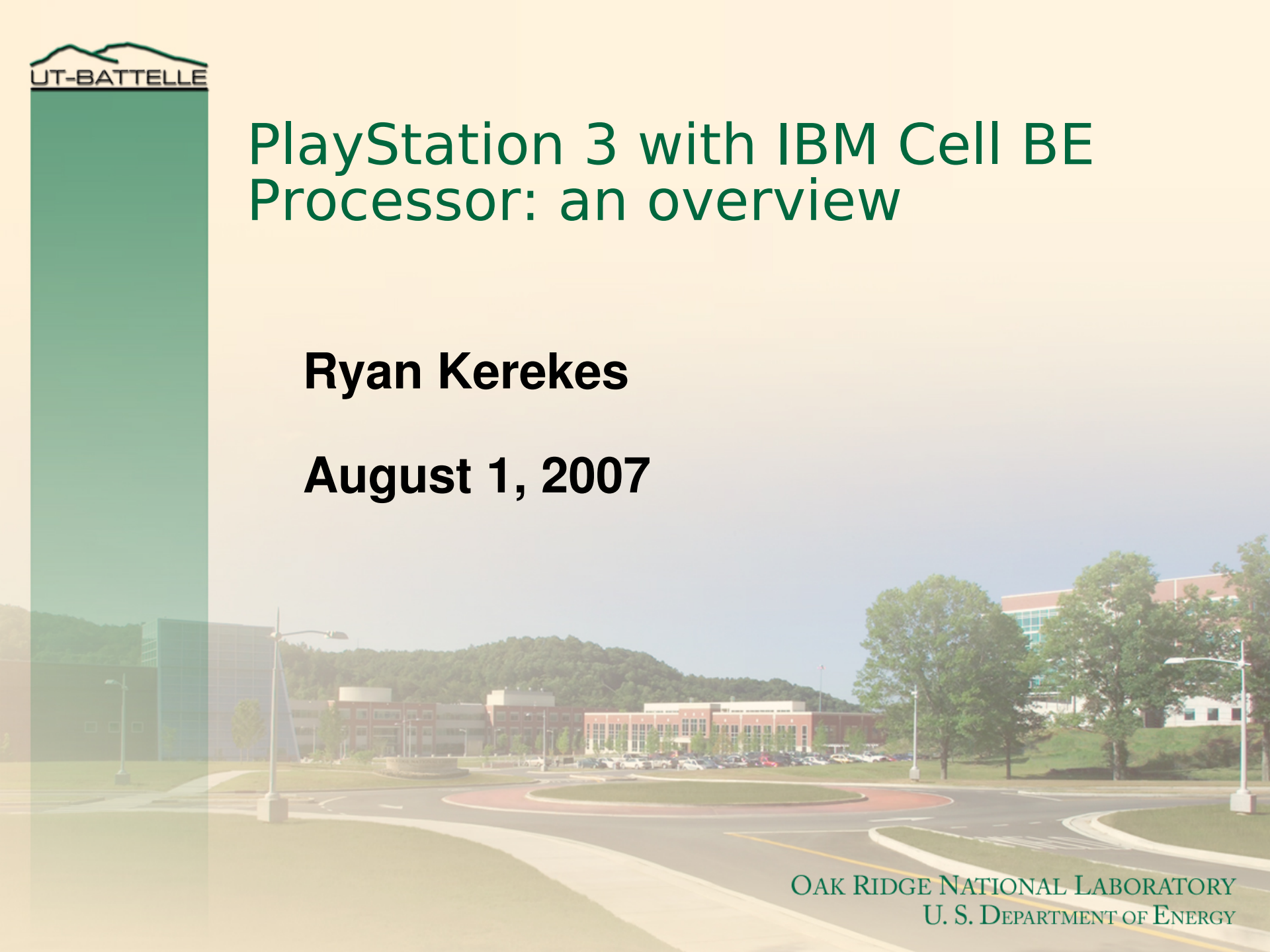
UT-BATTELLE

PlayStation 3 with IBM Cell BE Processor: an overview

Ryan Kerekes

August 1, 2007



OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

Outline

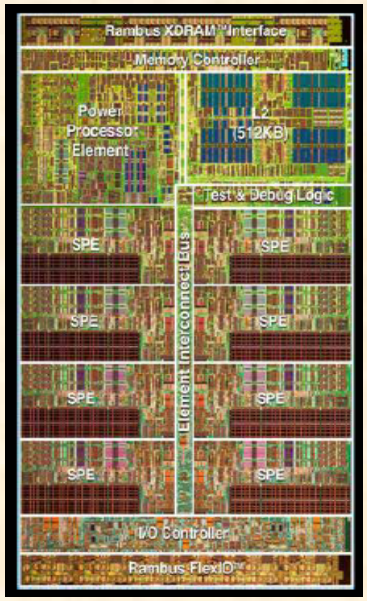
- **What is the Cell BE processor?**
  - **Architecture, speed, etc.**
- **How do we write programs for Cell?**
  - **SDK (helpful functions)**
  - **Development environment**
- **What are some potential applications?**

What is Cell BE?

- **Cell "Broadband Engine" (BE) processor introduced by IBM in 2005**
  - **"Supercomputer-on-a-chip"**
- **1 Power-PC processor (called the PPU) @ 3.2 GHz**
- **"8" SIMD processors (called SPUs) @ 3.2 GHz**
  - **256K local memory each**
  - **"synergistic processing elements"**

Where is Cell BE?

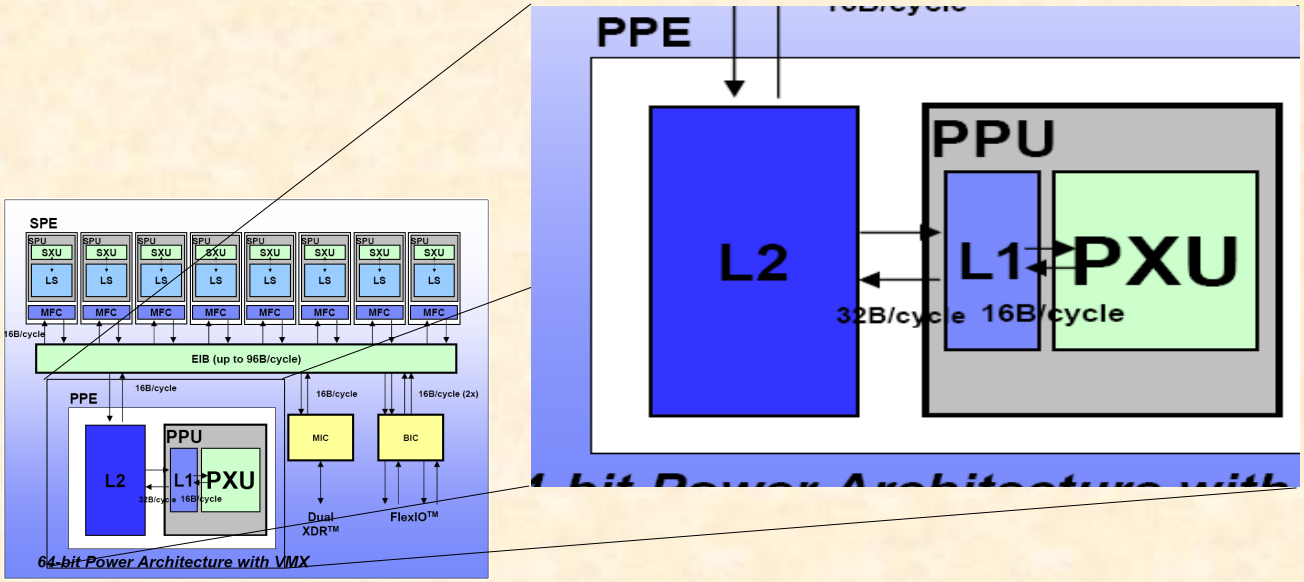
- **IBM Blade Center QS20**
  - 2 Cell BE processors
  - 1 GB RAM
  - $17K
- **PlayStation 3**
  - 1 Cell BE processor
  - 256 MB RAM
  - Only 6 of 8 SPU's available
  - $500
  - Running Linux (FC5)



Cell BE

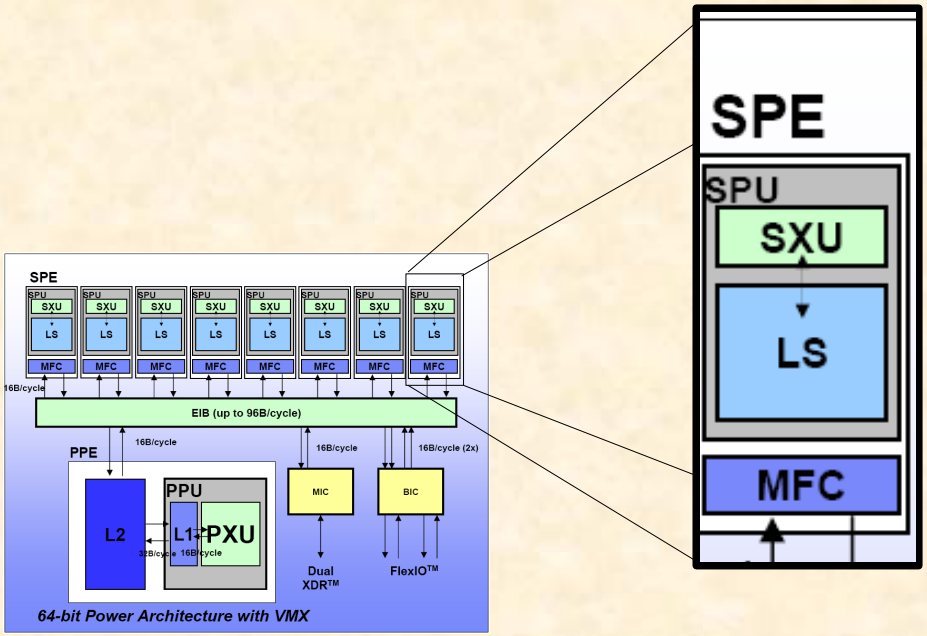
Cell BE Architecture – PPU

- **PPU runs independently from SPUs**
- **PPU typically used as central controller for SPUs**
  - **Spawns threads on each SPU, issues commands**



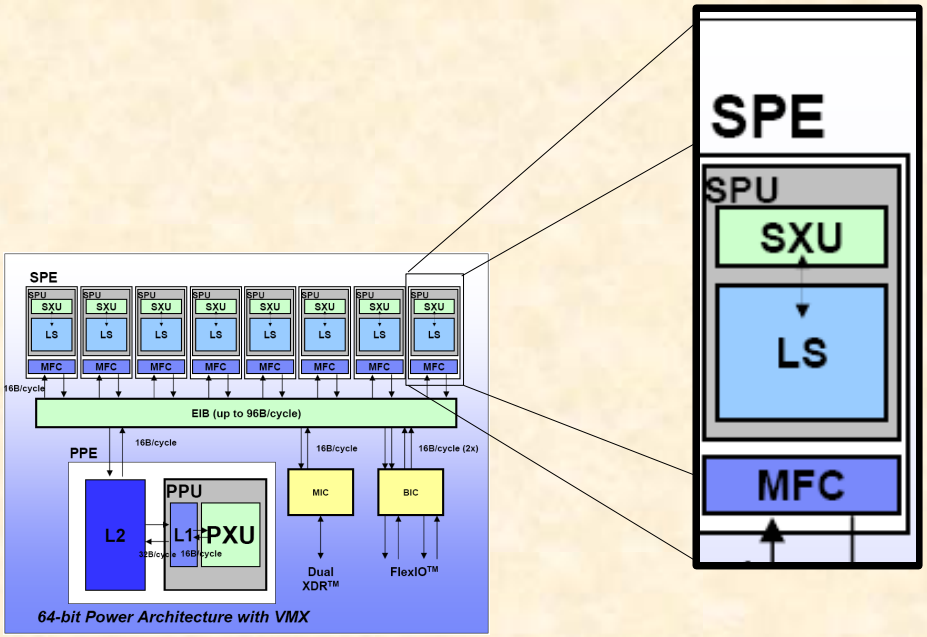
Cell BE Architecture – SPU (cont'd.)

- **SPUs are vector processors (SIMD)**
  - **can operate on 4 floats per cycle (or 4 ints, or 8 shorts)**
  - **> 25 Gflops per SPU → 150 Gflops total (max)**



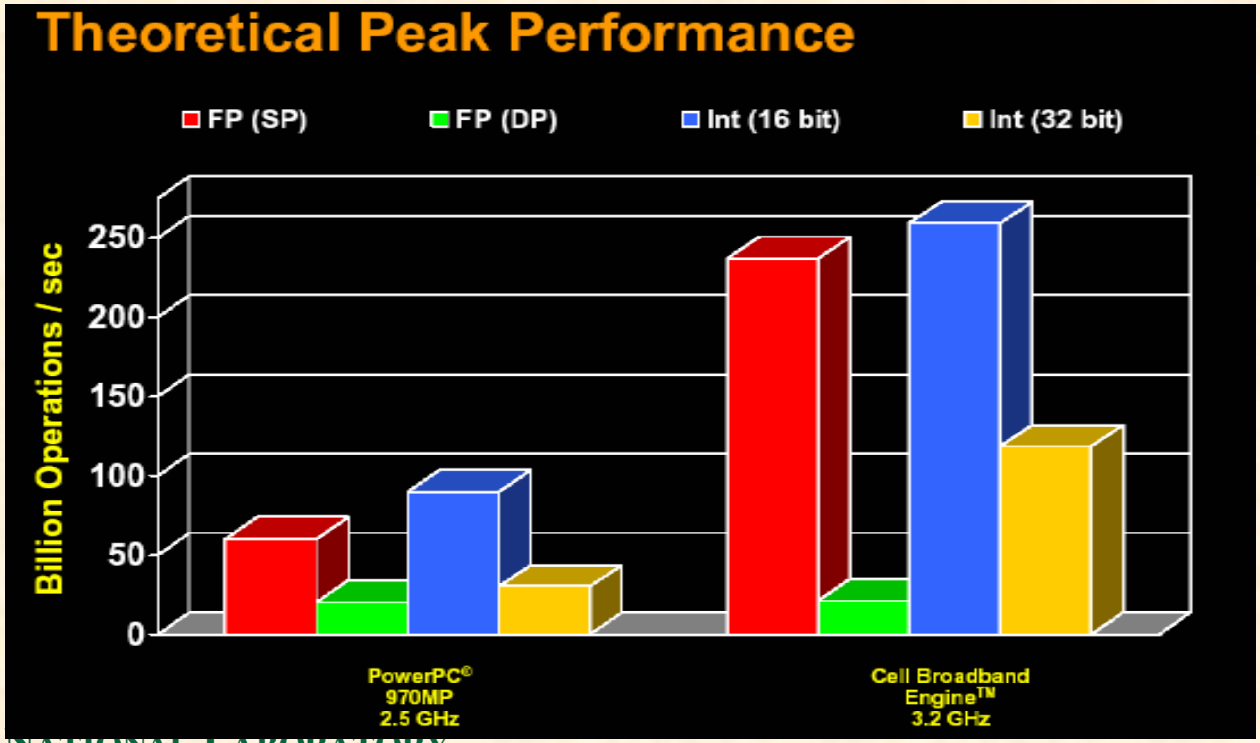
Cell BE Architecture – SPU

- **8 (6) SPUs run independent threads**
- **Each can only operate on data in Local Store (LS)**
  - **Only 256K to work with – big limitation (?)**



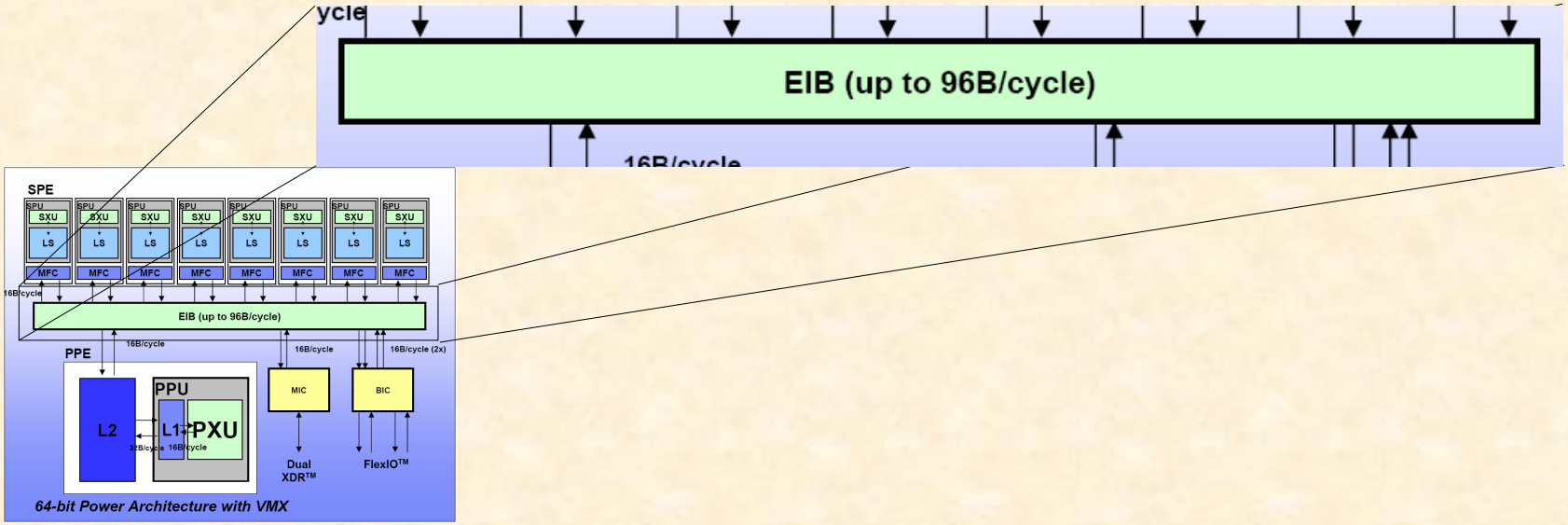
SIMD Performance

- **Cell SPUs are tailored to work very well with single-precision floats**
  - **Double-precision: not so much…**



Cell BE Architecture – EIB

- **Element Interconnect Bus (EIB) – used to transfer data between PPU and SPUs**
  - **25GB/s theoretical bandwidth**
  - **Can transfer data *concurrently* with code execution**



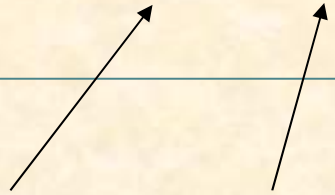
How do we write Cell code?

- **Step 1: Download Cell SDK 2.0**
  - **Specialized C/C++ compilers and utilities (gcc-like)**
    - `ppu-gcc`, `spu-gcc`, `spu-gdb`, **etc.**
  - **SDK libraries (useful functions for SPE computation)**
    - **e.g., matrix multiply, fast Fourier transform**
  - **C language extensions for exploiting SIMD capabilities**
  - **IBM Cell System Simulator (development environment)**

How do we write Cell code?

- **Step 2: Follow the general programming model**
- **example: PPU spawns 6 threads (1 per SPU), delivers application-specific info to each thread**

```
for(i=0; i<6; i++) {
        spu_create_thread(…, &fcn, &info[i], …);
}
```



function to run on SPE

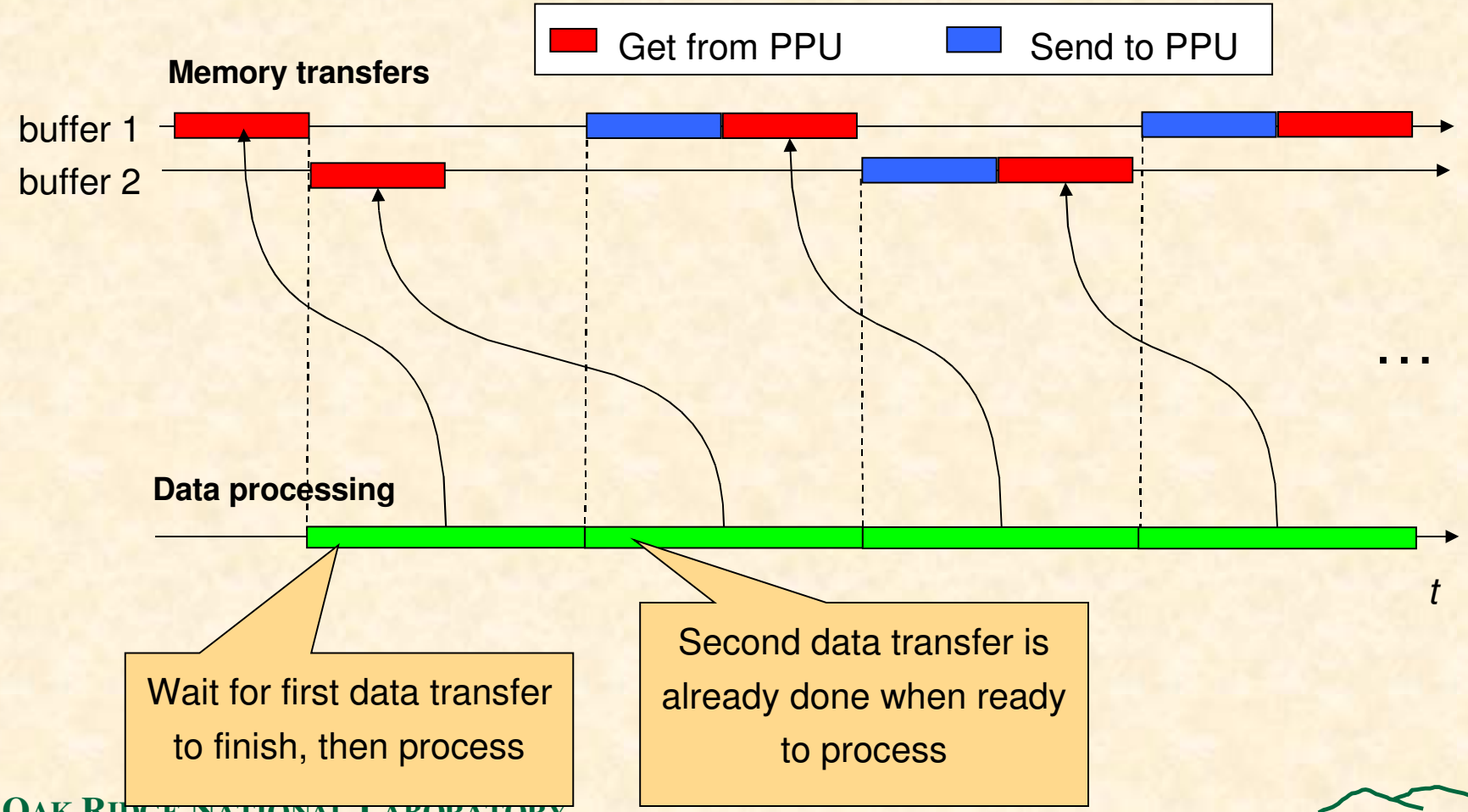
info to pass to SPE

How do we write Cell code?

- **Each SPE uses Memory Flow Controller (MFC) functions to transfer data from/to PPU**
- **Each MFC can queue up to 16 transfer requests**
- **Transfer-while-you-work**
  - **1. call transfer request**
  - **2. execute some code**
  - **3. wait for transfer to complete**
- **potential to completely "hide" memory transfers**

How do we write Cell code?

- **Example: double buffering**

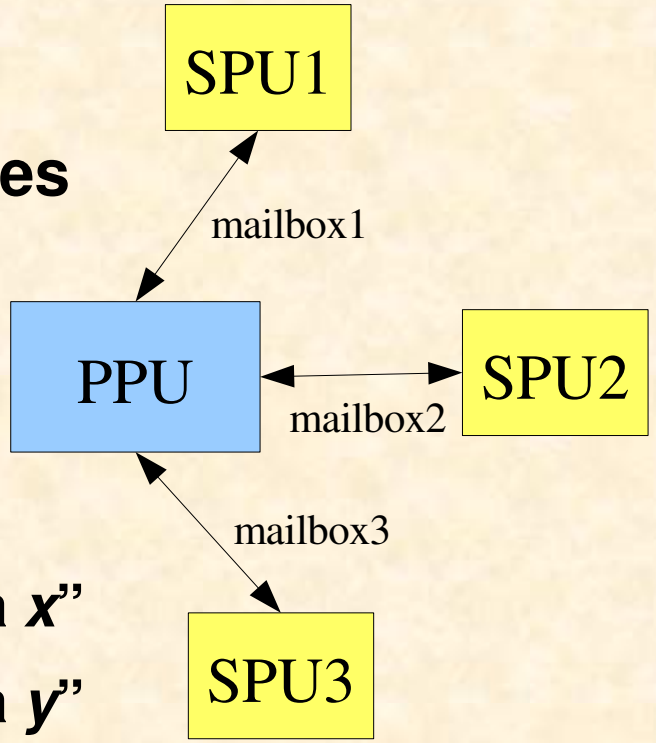


How do we write Cell code?

- **Memory transfers have to be aligned to 16-byte boundaries**
- **Best performance if aligned to 128-byte boundaries**

How do we write Cell code?

- **PPU/SPE communication via mailboxes**
  - **32-bits per message**
  - **Each SPU has own mailbox with PPU**
- **Example:**
  - **PPU: "hey SPU1, go do task #1 on data *x*"**
  - **PPU: "hey SPU2, go do task #1 on data *y*"**
  - **SPU1: "okay, I'm done with task #1"**
  - **PPU: "fine. go do task #5 on data *z*"**
  - **SPU2: "okay, I'm done with task #1"**
  - **PPU: "wonderful. go do task #6 on data *x*"**

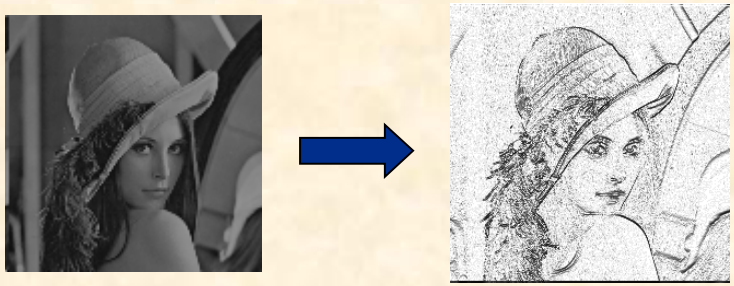


How do we write Cell code?

- **SDK functions provided by IBM**
- **Tailored for SIMD architecture (SPUs)**
  - **audio resampling library**
  - **curves and surfaces library (e.g., bezier curves)**
  - **FFT library**
  - **“game math” library (fast approximations)**
  - **image library**
  - **large matrix library**
  - **others**

Example: edge detection in images

- **Implemented a simple image edge detector on PS3 using the SDK image library**
  - **Involves image filtering operations (convolution)**



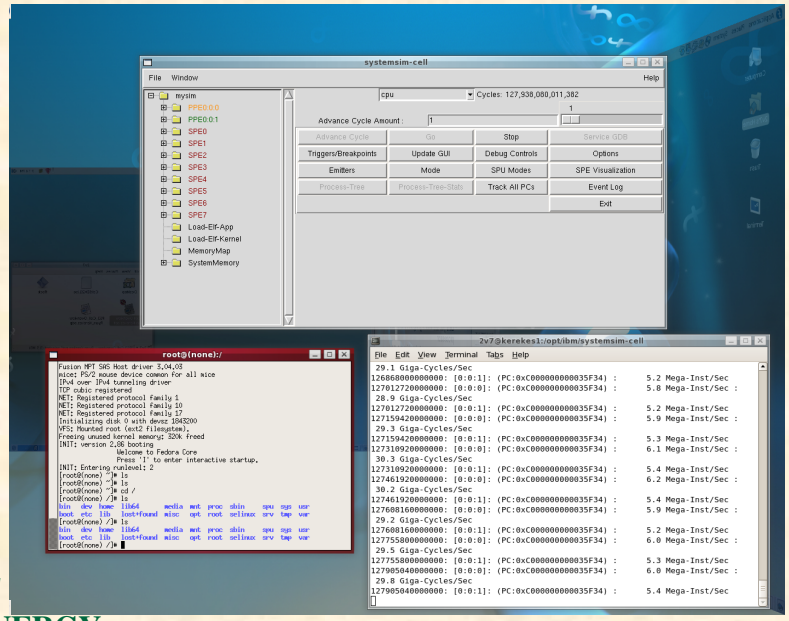
- **Filtering speed comparison:**

Single CPU	Single SPU	6 SPUs
4,693 conv/sec	10,240 conv/sec	63,325 conv/sec

- **Cell BE demonstrated 13.5x speedup over CPU**

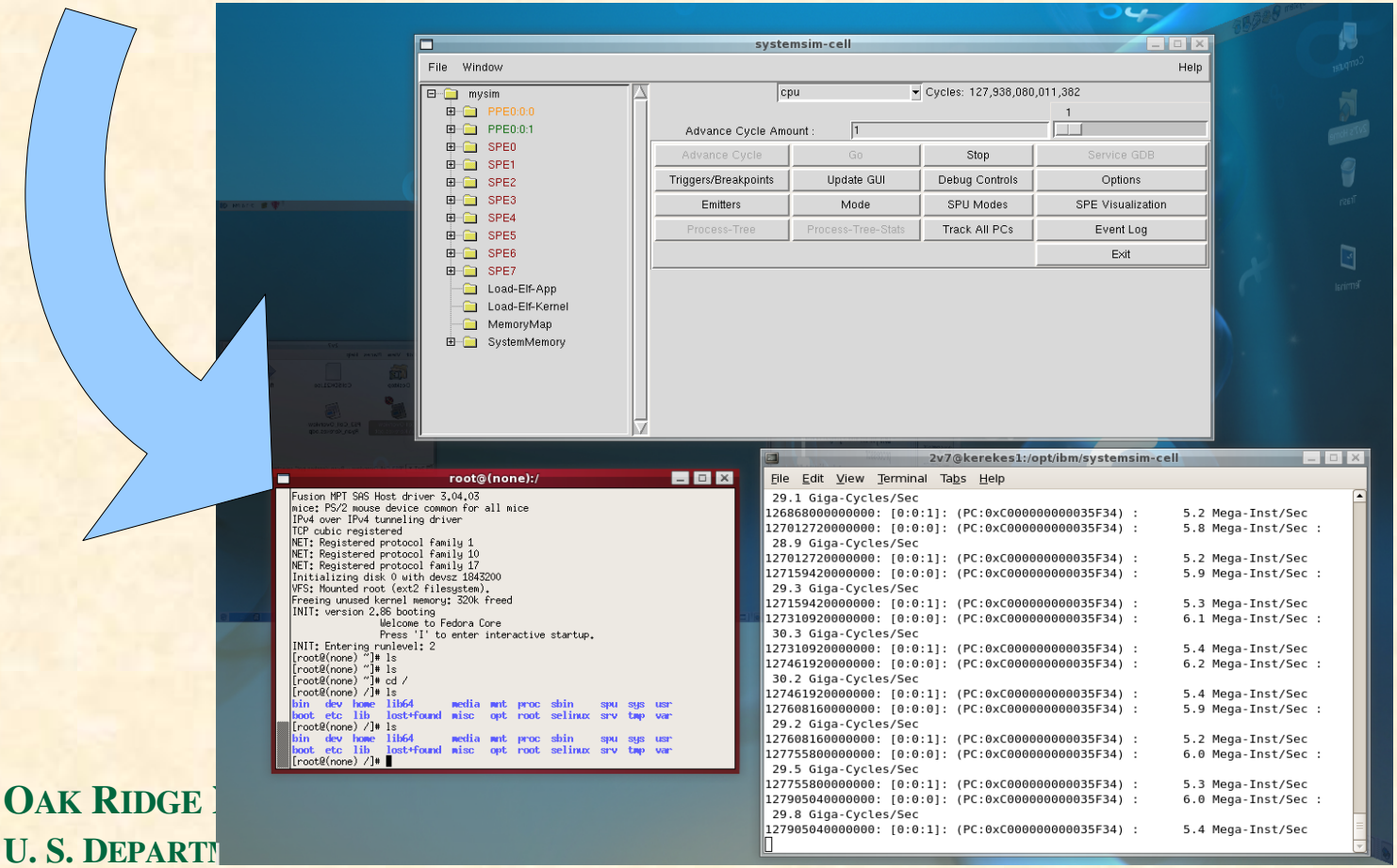
IBM Cell System Simulator

- **IBM Cell System Simulator: a development tool**
  - **Lets you see what's going on inside the Cell processor**
  - **can be run on a non-Cell machine (e.g., my desktop)**
  - **good for testing/debugging Cell software**



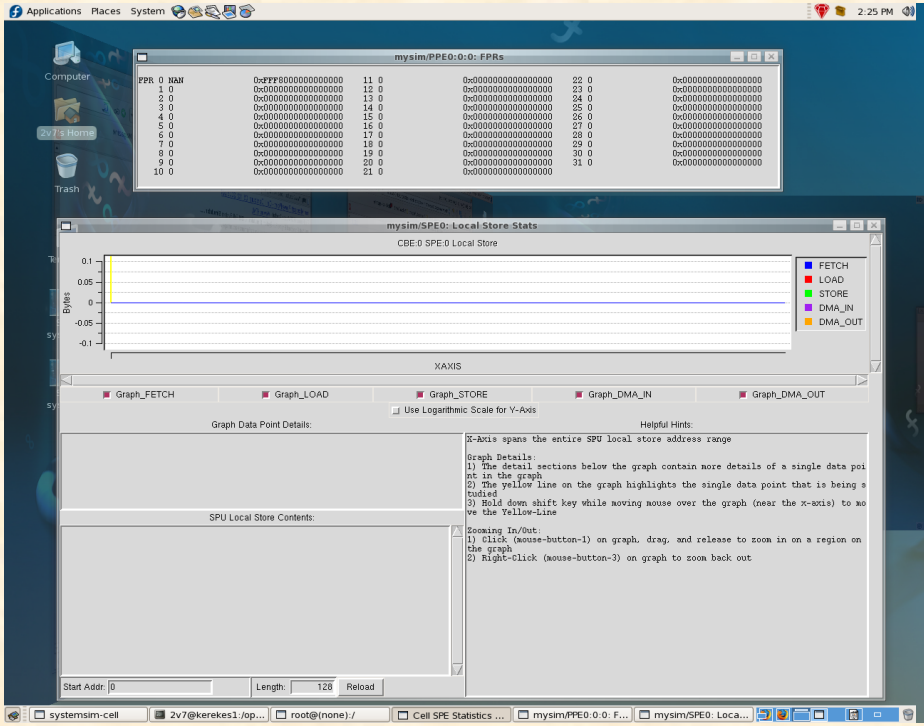
IBM Cell System Simulator

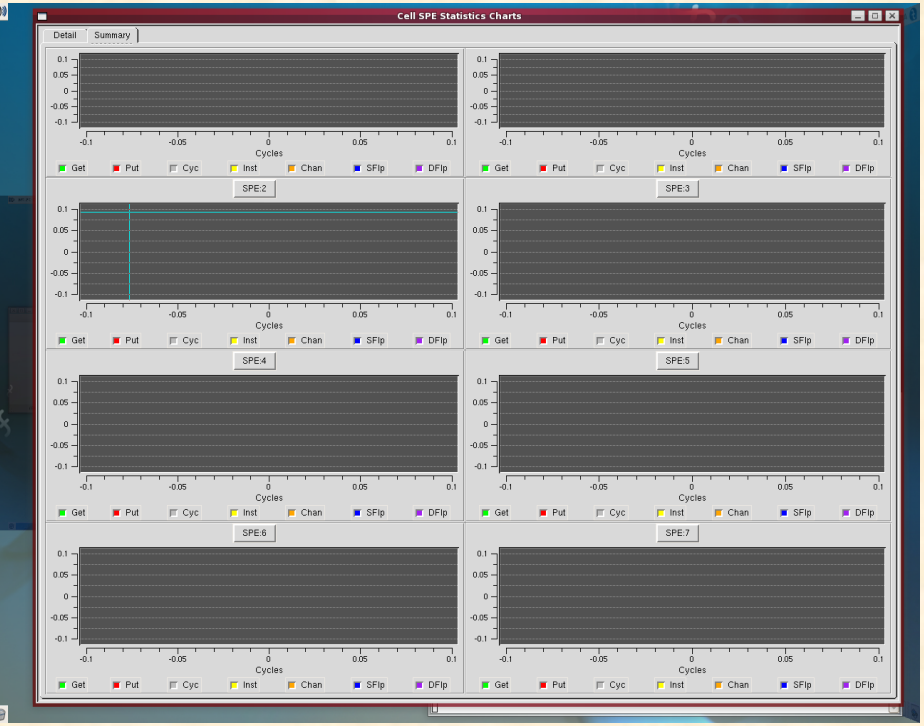
- **Creates a virtual Cell machine and boots a virtual Linux OS on top of it**



IBM Cell System Simulator

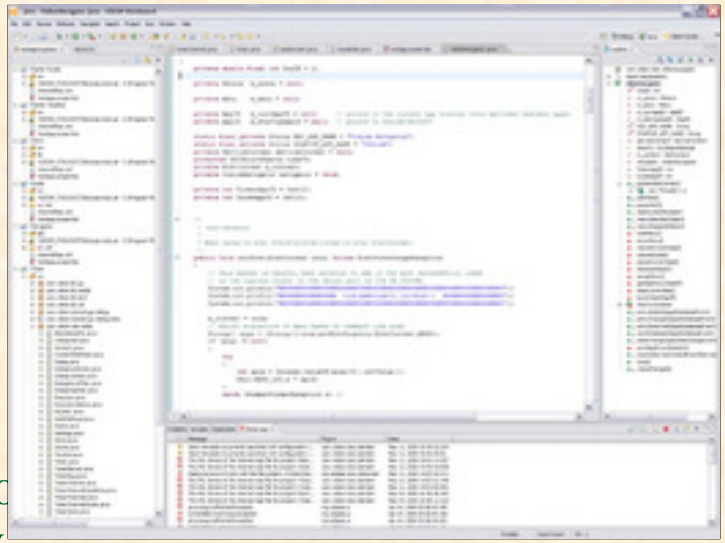
- **Many different views for performance analysis/debugging**





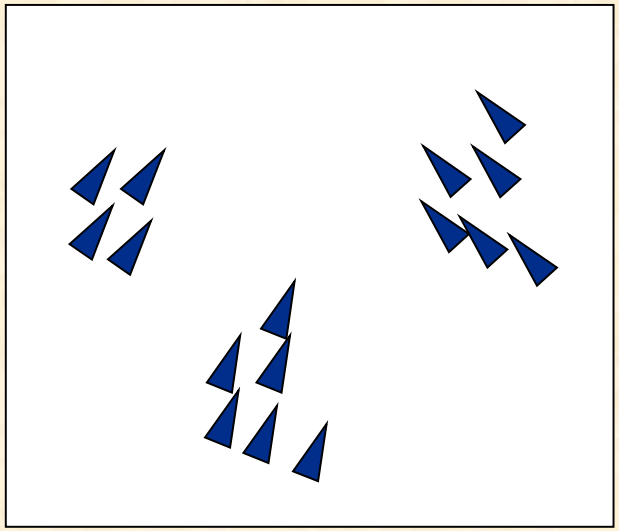
IDE for Cell

- **IBM Integrated Development Environment (IDE) for Cell SDK**
  - **built on Eclipse**
  - **requires Linux (Fedora Core 6)**
  - **integrates Cell System Simulator into Eclipse**
  - **visual interface for SPU-GDB**



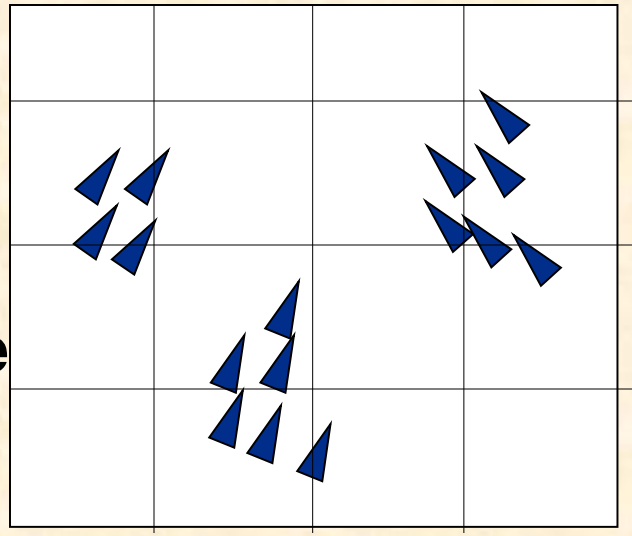
Flocking on PS3

- **Implemented a document flocking algorithm on the PS3**
  - **(no documents yet)**
- **Parallelized by letting each SPU update 1/6 of the "birds" during each generation**
  - **determine neighbors**
  - **update velocities/positions**



Flocking on PS3

- **Problems with this implementation:**
  - **partitioned by bird index**
  - **each SPU needs to have all birds stored**
  - **theoretical limit of ~7,000 birds (256K local store)**
  - **need clever implementation for more documents**
- **Can we partition by space?**
- **How do we scale this to be able to use clusters of PS3s?**
  - **are PS3s a good choice?**



Conclusion: Possible uses for PS3

- **Need to hide communication**
  - **Main memory access: need 24 flops/value**
  - **Network (MPI cluster): need ~5000 flops/value**
- **Application must be *data-parallel* as well as computationally parallel**
- ***Image analysis* – computing large numbers of features per image**
- ***Parallel document clustering?***