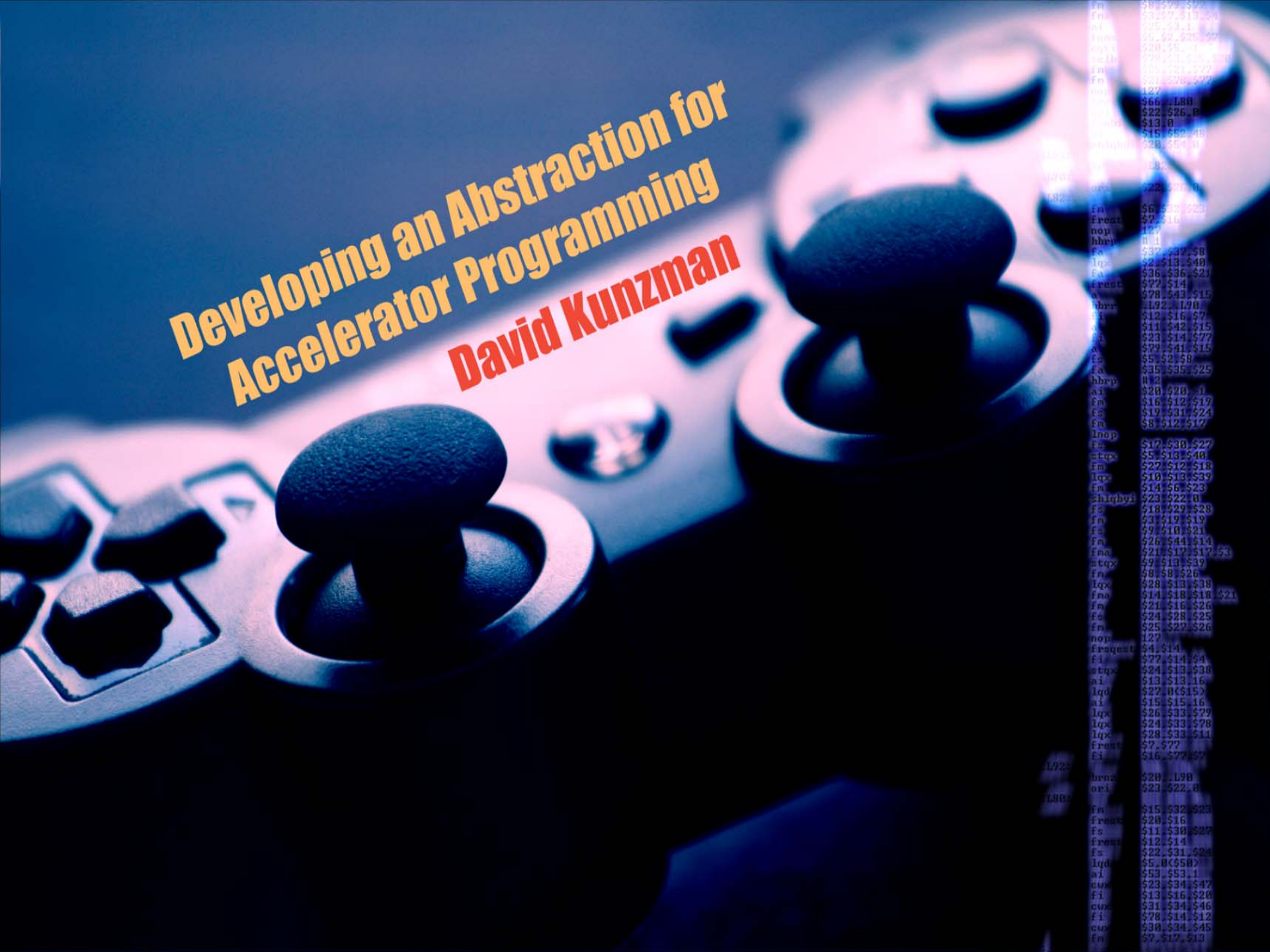

Developing an Abstraction for Accelerator Programming
David Kunzman

Intro

- This talk will focus on Cell processor
  - Cell Broadband Engine Architecture (CBEA)
    - Power Processing Element (PPE)
    - Synergistic Processing Element (SPE)
  - Current implementations
    - Sony Playstation 3 (1 chip with 6 SPEs)
    - IBM Blades (2 chips with 8 SPEs each)
    - Toshiba SpursEngine (1 chip with 4 SPES)
- Future work will try to include GPUs & Larrabee

Two Topics in One

- Accelerators (Accel)
  …this is going to hurt…
- Heterogeneous systems (Hetero)
  …kill me now…
- Goal of work… take away the pain and make code portable
- Code examples

Why Use Accelerators?

- Performance

Why Not Use Accelerators?

- Hard to program
  - Many architecturally specific details
    - Different ISAs between core types
    - Explicit DMA transactions to transfer data to/from the SPEs' local stores
    - Scheduling of work and communication
  - Code is not trivially portable
    - Structure of code on an accelerator often does not match that of a commodity architecture
    - Simple re-compile not sufficient

Extensions Charm++

- Added extensions
  - Accelerated entry methods
  - Accelerated blocks
  - SIMD instruction abstraction
- Extensions should be portable between architectures

Accelerated Entry Methods

- Executed on accelerator if present
- Targets computationally intensive code
- Structure based on standard entry methods
  - Data dependencies expressed via messages
  - Code is self-contained
- Managed by the runtime system
  - DMAs automatically overlapped with work on the SPEs
  - Scheduled (based on data dependencies: messages, objects)
  - Multiple independently written portions of code share the same SPE (link to multiple accelerated libraries)

Accel Entry Method Structure

```
entry [accel] void entryName
        ( …passed parameters… )
        [ …local parameters… ]
        { … function body … }
        callback_member_funcion;

objProxy.entryName( … passed parameters …)
```

Accelerated Blocks

- Additional code that is accessible to accelerated entry methods
  - #include directives
  - Functions called by accelerated entry methods

SIMD Abstraction

- Abstract SIMD instructions supported by multiple architectures
  - Currently adding support for: SSE (x86), AltiVec (PowerPC; PPE), SIMD instructions on SPEs
  - Generic C implementation when no direct architectural support is present
  - Types: vec4f, vec2lf, vec4i, etc.
  - Operations: vadd4f, vmul4f, vsqrt4f, etc.

“HelloWorld” Code

hello.ci

```
mainmodule hello {
  ...
  accelblock {
    void sayMessage(char* msg,
                    int thisIndex,
                    int fromIndex) {
      printf("%d told %d to say ₩"%s₩"₩n",
             fromIndex, thisIndex, msg);
    }
  };

  array [1D] Hello {
    entry Hello(void);
    entry [accel] void saySomething(
        int msgLen,
        char msg[msgLen],
        int fromIndex )[
        readonly : int thisIndex <impl_obj->thisIndex>
      ] {
        sayMessage(msg, thisIndex, fromIndex);
    } saySomething_callback;
  };
};
```

Hello.C

```
class Main : public CBase_Main {
  Main(CkArgMsg* m) {
    CkPrintf("Running Hello on %d processors for %d elements₩n",
             CkNumPes(), nElements);
    char *msg = "Hello from Main";
    arr[0].saySomething(strlen(msg) + 1, msg, -1);
  };
  void done(void) { CkPrintf("All done₩n"); CkExit(); };
};

class Hello : public CBase_Hello {
  void saySomething_callback() {
    if (thisIndex < nElements - 1) {
      char msgBuf[128];
      int msgLen = sprintf(msgBuf, "Hello from %d", thisIndex) + 1;
      thisProxy[thisIndex+1].saySomething(msgLen, msgBuf,
                                          thisIndex);
    } else {
      mainProxy.done();
    }
  }
 };
```

“HelloWorld” Output

X86

```
---------------------------------
Running Hello on 1 processors for 5 elements
-1 told 0 to say "Hello from Main"
0 told 1 to say "Hello from 0"
1 told 2 to say "Hello from 1"
2 told 3 to say "Hello from 2"
3 told 4 to say "Hello from 3"
All done
```

Blade

```
---------------------------------
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
SPE reported _end = 0x00006930
Running Hello on 1 processors for 5 elements
-1 told 0 to say "Hello from Main"
0 told 1 to say "Hello from 0"
1 told 2 to say "Hello from 1"
2 told 3 to say "Hello from 2"
3 told 4 to say "Hello from 3"
All done
```

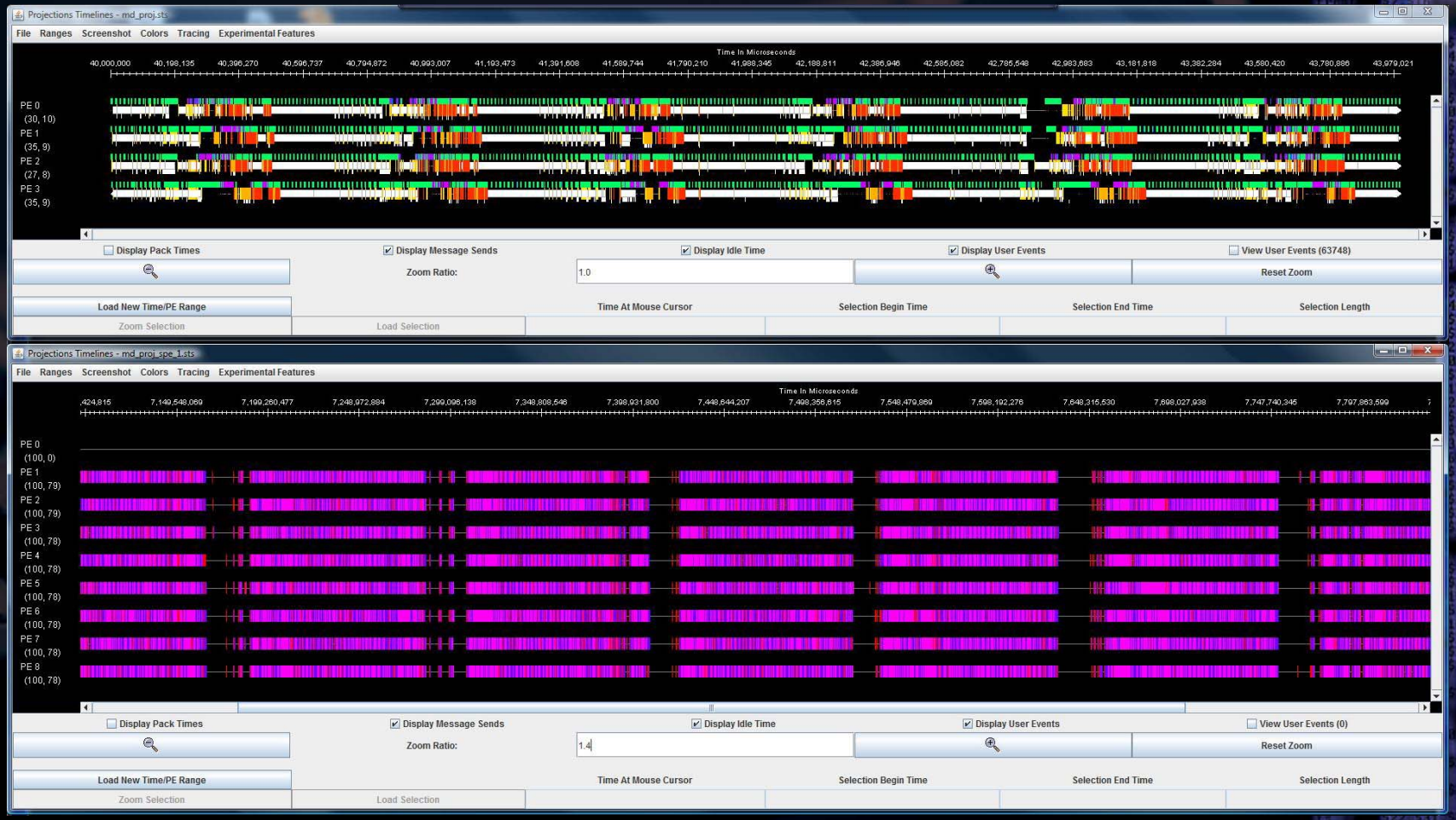
MD Example Code

- List of particles evenly divided into equal sized *patches*
  - Compute objects calculate forces
    - Coulomb's Law
    - Single precision floating-point
  - Patches sum forces and update particle data
  - All particles interact with all other particles each timestep
- ~92K particles (similar to ApoA1 benchmark)
- Uses SIMD abstraction for all versions

MD Example Code

- Speedups (vs. 1 x86 core using SSE)
  - 6 x86 cores: 5.89
  - 1 QS20 chip (8 SPEs): 5.74
- GFlops/sec for 1 QS20 chip
  - 50.1 GFlops/sec observed (24.4% peak)
  - Nature of code (single inner-loop iteration)
    - Inner-loop: 124 Flops using 54 instructions in 56 cycles
    - Sequential code executing continuously can achieve, at most, 56.7 GFlops/sec (27.7% peak)
    - We observe 88.4% of the *ideal GFlops/sec for this code*
  - 178.2 GFlops/sec using 4 QS20s (net-linux layer)

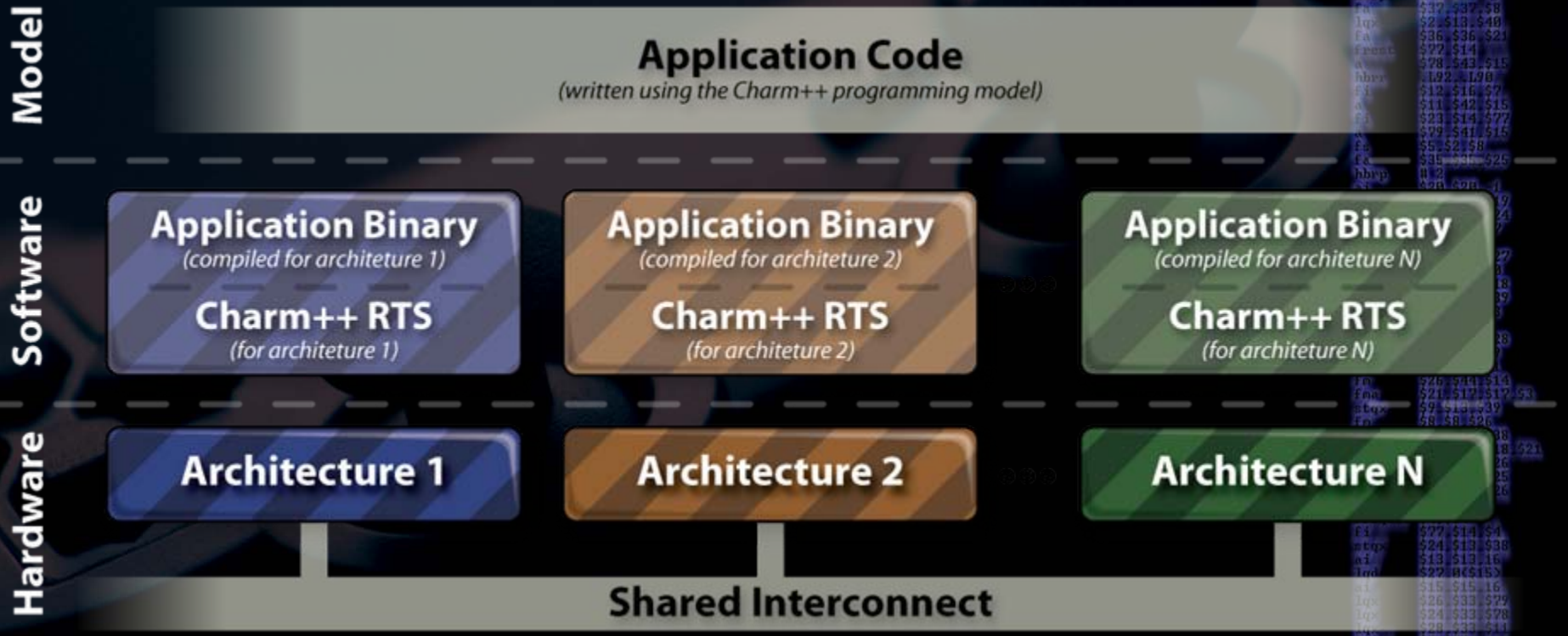
Projections



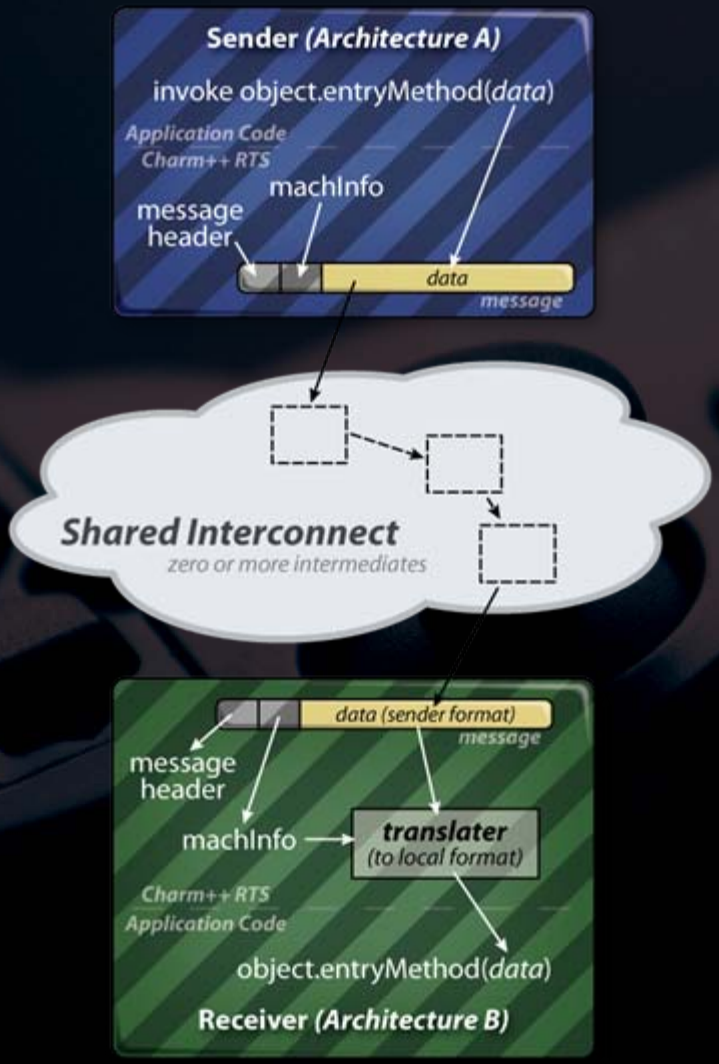
Why Heterogeneous?

- Trend towards specialized accelerator cores mixed with general cores
  - #1 supercomputer on Top500 list, Roadrunner at LANL (Cell & x86)
  - Lincoln Cluster at NCSA (x86 & GPUs)
- Aging workstations that are loosely clustered

Hetero System View



Messages Across Architectures



- Makes use of Pack-UnPack (PUP) routines
  - Object migration and parameter marshaled entry method are the same as before
  - Custom pack/unpack routines for messages can use PUP framework
- Supported machine-layers:
  - net-linux
  - net-linux-cell

Making Hetero Runs

- Launch using *charmrun*
  - Compile separate binary for each architecture
  - Modified nodelist files to specify correct binary based on architecture

Hetero “Hello World” Example

Nodelist

```
----------------------------
group main ++shell "ssh -X"
host   kaleblade  ++pathfix  __arch_dir__         net-linux
host      blade_1  ++pathfix  __arch_dir__  net-linux-cell
host        ps3_1  ++pathfix  __arch_dir__  net-linux-cell
```

Accelblock change in hello.ci (just for demonstration)

```
----------------------------
 accelblock {
   void sayMessage(char* msg,
                   int thisIndex,
                   int fromIndex) {
     #if CMK_CELL_SPE != 0
       char *coreType = "SPE";
     #elif CMK_CELL != 0
       char *coreType = "PPE";
     #else
       char *coreType = "GEN";
     #endif
     printf("[%s] :: %d told %d to say \"%s\"\n",
            coreType, fromIndex, thisIndex, msg);
   }
 };
```

Launch Command:

```
----------------------------
./charmrun ++nodelist ./nodelist_hetero +p3
       ~/charm/__arch_dir__/examples/charm++/cell/hello/hello 10
```

Output

```
----------------------------
Running Hello on 3 processors for 10 elements
[GEN] :: -1 told 0 to say "Hello from Main"
[SPE] :: 0 told 1 to say "Hello from 0"
[SPE] :: 1 told 2 to say "Hello from 1"
[GEN] :: 2 told 3 to say "Hello from 2"
[SPE] :: 3 told 4 to say "Hello from 3"
[SPE] :: 4 told 5 to say "Hello from 4"
[GEN] :: 5 told 6 to say "Hello from 5"
[SPE] :: 6 told 7 to say "Hello from 6"
[SPE] :: 7 told 8 to say "Hello from 7"
[GEN] :: 8 told 9 to say "Hello from 8"
All done
```

Summary

- Development still in progress (both)
- Addition of accelerator extensions
  - Example codes in Charm++ distribution (the nightly build)
  - Achieve good performance
- Heterogeneous system support
  - Simple example codes running
  - Not in public Charm++ distribution yet

Credits

- Work partially supported by NIH grant PHS 5 P41 RR05969-04: Biophysics / Molecular Dynamics
- Cell hardware supplied by IBM SUR grant awarded to University of Illinois
- Background Playstation controller image originally taken by “wlodi” on Flickr and modified by David Kunzman