Programming

Cell & Larrabee

Using Accelerated Entry Methods

David Kunzman

Parallel Programming Lab

University of Illinois at Urbana-Champaign

11.30.2009

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB

DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS

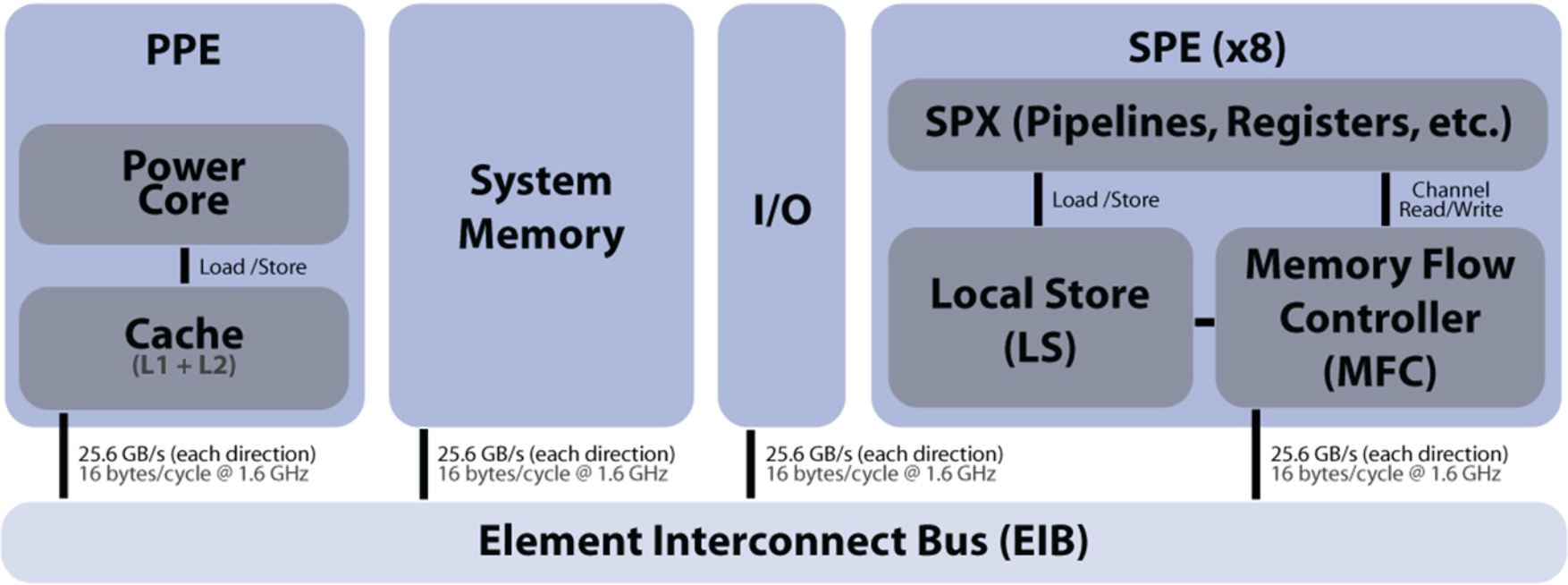
PPL UIUC

Motivation/Goals

- Allow Charm++ programs to use accelerators
  - Currently targeting Cell and Larrabee
  - Difficult to program
- Ease programmer burden
  - Single programming model for all processing cores
  - Portability
  - Modularity

PPL UIUC

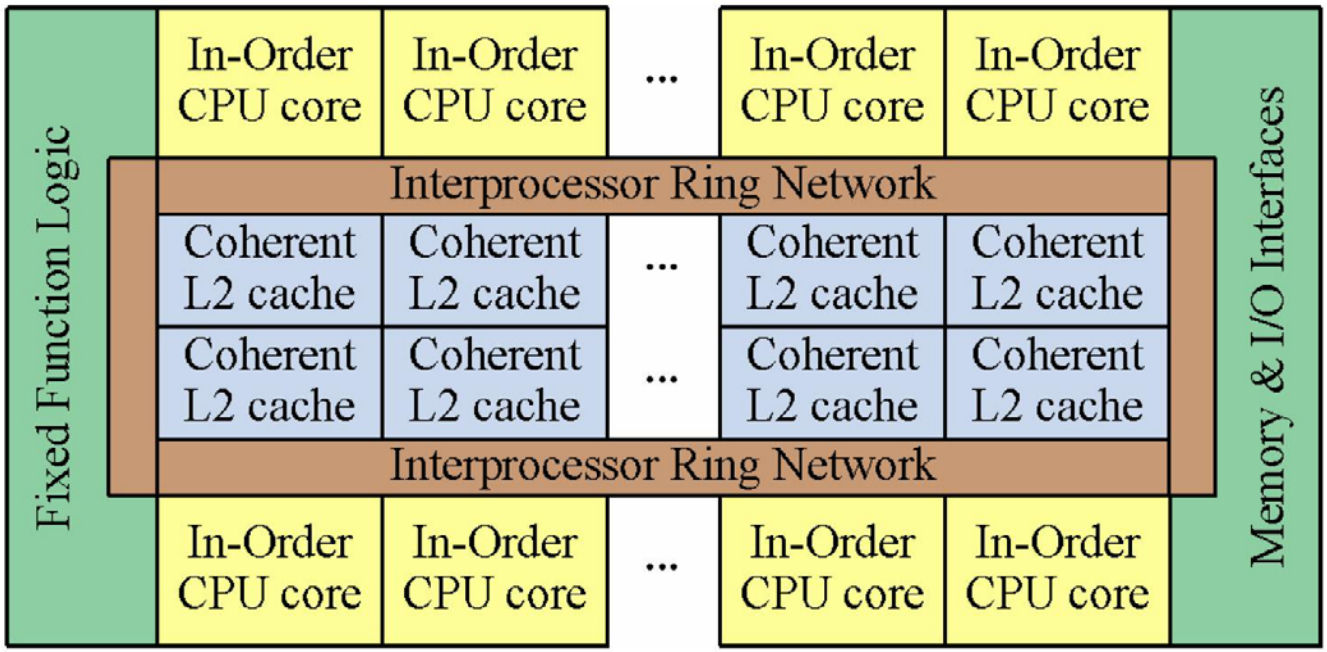
Cell Overview



- Single Power Processing Element (PPE)
- Multiple Synergistic Processing Elements (SPEs): Blade = 8, PS3 = 6, SpursEngine = 4
- SPEs are in different address spaces with limited local store (scratchpad) memory
- 4-way SIMD (vector) instructions

PPL UIUC

Larrabee Overview



- Designed as GPU (accelerator to a host core)
- Multiple cores
- Cores access a coherent cache-based memory hierarchy
- 16-way SIMD (vector) instructions
- *Image from Larrabee SigGraph 2008 paper*

PPL UIUC

Entry Method Structure

Interface File:

```
entry void entryName
        ( …passed parameters… );
```

Source File:

```
void ChareClass::entryName
        ( …passed parameters … )
        { … function body … }
```

Invocation:

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

PPL UIUC

Basic Entry Method

```
entry void accum(int inArrayLen, float inArray[inArrayLen]);


void ChareObj::accum(int inArrayLen, float* inArray) {

    if (inArrayLen != localArrayLen) return;

    for (int i = 0; i < inArrayLen; ++i)
        localArray[i] = localArray[i] + inArray[i];
}
```

To Invoke: myChareObj.accum(someFloatArray_len,
someFloatArray_ptr);

PPL UIUC

Extensions to Charm++

SIMD Instruction Abstraction

&

Accelerated Entry Methods

PPL
UIUC

SIMD Abstraction

- Abstract SIMD instructions supported by multiple architectures
  - Currently adding support for: SSE (x86), AltiVec/VMX (PowerPC; PPE), SIMD instructions on SPEs, and Larrabee
  - Generic C implementation when no direct architectural support is present
  - Types: vecf, veclf, veci, ...
  - Operations: vaddf, vmulf, vsqrtf, ...

PPL UIUC

SIMD Instruction Abstraction

```
entry void accum(int inArrayLen, align(sizeof(vecf)) float inArray[inArrayLen]);

void ChareObj::accum(int inArrayLen, float* inArray) {

    if (inArrayLen != localArrayLen) return;

     vecf* inArrayVec = (vecf*)inArray;
     vecf* localArrayVec = (vecf*)localArray;
     int arrayVecLen = inArrayLen / vecf_numElems;
     for (int i = 0; i < arrayVecLen; ++i)
          localArrayVec[i] = vaddf(localArrayVec[i], inArrayVec[i]);

     for (int i = arrayVecLen * vecf_numElems; i < inArrayLen; ++i)
          localArray[i] = localArray[i] + inArray[i];
}
```

To Invoke: myChareObj.accum(someFloatArray_len, someFloatArray_ptr);

PPL UIUC

Accelerated Entry Methods

- Targets computationally intensive code
  - Execute on accelerator if one or more supported accelerators are present
  - Otherwise, execute on host core
- Structure based on standard entry methods
  - Data dependencies expressed via messages (passed parameters)
  - Code is self-contained (accesses message and chare object data)
- The runtime system manages…
  - Data movement: DMAs automatically overlapped with work on the accelerator(s)
  - Scheduling: based on data dependencies (messages and chare objects)
- Multiple independently written portions of code share the same accelerator(s)

Accel Entry Method Structure

Basic

Interface File:

```
entry void entryName
  ( …passed parameters… );
```

Source File:

```
void ChareClass::entryName
  ( …passed parameters … )
  { … function body … }
```

vs.

Accelerated

Interface File:

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

Invocation (both): chareObj.entryName(… passed parameters …)

PPL UIUC

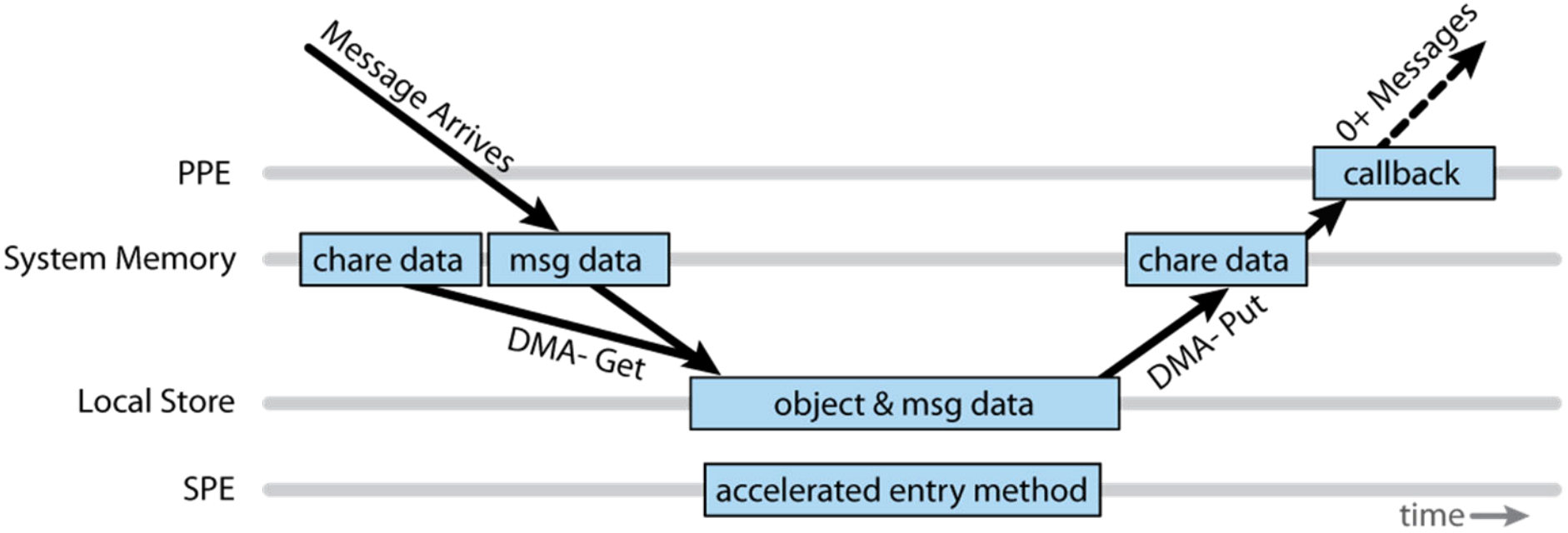
Same Entry Method Accelerated

```
entry [accel] void accum(int inArrayLen, align(sizeof(vecf)) float inArray[inArrayLen])
                [  readOnly : int localArrayLen <impl_obj->localArrayLen>,
                  readWrite : float localArray[localArrayLen] <impl_obj->localArray> ] {

    if (inArrayLen != localArrayLen) return;
     vecf* inArrayVec = (vecf*)inArray;
     vecf* localArrayVec = (vecf*)localArray;
     int arrayVecLen = inArrayLen / vecf_numElems;

     for (int i = 0; i < arrayVecLen; ++i)
          localArrayVec[i] = vaddf(localArrayVec[i], inArrayVec[i]);

     for (int i = arrayVecLen * vecf_numElems; i < inArrayLen; ++i)
          localArray[i] = localArray[i] + inArray[i];

} accum_callback;
```

To Invoke: myChareObj.accum(someFloatArray_len, someFloatArray_ptr);

PPL UIUC

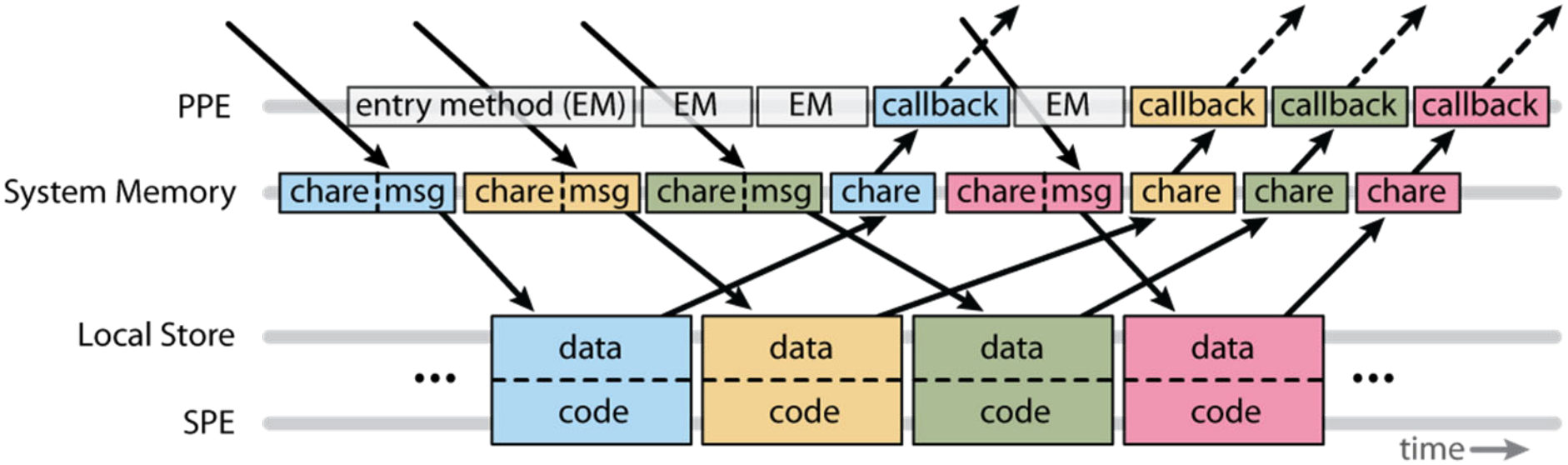
Timeline of Events



- Runtime system…
  - Directs data movement (messages & DMAs)
  - Schedules accelerated entry methods and callbacks

PPL UIUC

Communication Overlap



- Data movement automatically overlapped with accelerated entry method execution on SPEs and entry method execution on PPE

A Step Further: Heterogeneity

- Code already portable between systems with and without accelerators
  - Have runtime system account for host core differences by automatically modifying application data (message data)
  - Then able to use all cores (host and accelerators)

PPL UIUC

Summary

- Common SIMD instructions abstracted via SIMD instruction abstraction
- Accelerators available to Charm++ programs via accelerated entry methods
  - Currently targeting Cell and Larrabee
  - Ease programmer burden
- Support for heterogeneous systems
- *Disclaimer: Still under development*

PPL UIUC