Porting Charm++ to the Cell Processor

David Kunzman, Laxmikant V. Kale
Cell Workshop
Motivation

- Parallel Programming Lab
  - Abstractions
  - Guided by real applications
Motivation

- **We want to use the Cell processor for science and engineering applications**
  - **Pros**
    - Provides great computational power (256 gigaflop/s peak)
    - Allows for both fine and coarse-grained parallelism
  - **Cons**
    - Hard to program
      - Want programmer to focus on application code, not Cell specific code
    - Even harder to debug
    - Portability
      - Cell specific code mixed in with application code
Motivation

- We believe the Charm++ paradigm fits well with Cell
- I will describe
  - What is Charm++
  - Why it is a good fit
  - Offload API
  - First application: NAMD (biomolecular simulation)
Charm++

- Object-Oriented, Message-Driven Parallel Programming Paradigm
  - Application broken up into objects called *chares*
  - Chares communicate using asynchronous messages
  - Chares have special member functions called *entry methods* that receive messages
- User doesn’t worry about processors when programming
Why Charm++ & Cell?

- **Data Encapsulation / Locality**
  - Each message associated with…
    - Code: Entry Method
    - Data: Message & Chare Data

- **Virtualization (many chares per processor)**
  - Provides opportunity to overlap SPE computation with DMA transactions
  - Helps ensure there is always useful work to do

- **Message Queue Peek-Ahead / Predictability**
  - Peek-ahead in message queue to determine future work
  - Fetch code and data before execution of entry method
Charm++

- Charm++ Runtime System
  - Chare to Processor Mapping
  - Routing Messages
  - Scheduling / Executing the Entry Methods
  - Dynamic Load-Balancing Framework
  - Dynamic Communication Optimization Library
  - Other Libraries: POSE (PDES), ParFUM (Mesh), …
  - Fault-Tolerance
  - …

- Adaptive MPI (AMPI) : MPI implementation that uses the Charm++ Runtime System
Charm++ Applications

- Molecular Dynamics
- Computational Cosmology
- Quantum Chemistry (QM/MM)
- Protein Folding
- Crack Propagation
- Rocket Simulation
- Dendritic Growth
- Space-time meshes
Charm++ on Cell

- How are we going to use the Cell?
- “Offload” the execution of entry methods onto the SPEs
Offload API

- **Goal:** Create an easy-to-use API that allows PPE code to efficiently use the SPEs
  - Allow user to focus on application code
  - Allow Offload API to take care of architecture specific details (DMAs, double-buffering, etc.)

- **Independent of Charm++**
  - Can be used by an C/C++ program
  - Designed with Charm++ in mind
Offload API

- Basic Idea
  - Encapsulate portions of the overall computation into chunks of computation called Work Requests
  - Each Work Request can have multiple input and output buffers (read/write, read-only, write-only)
  - Execute each Work Request on one of the SPEs
  - Concurrent Work Requests are self-contained
    - No data dependencies between them
    - Order-of-execution does not matter
Offload API

- **PPE Code**: “Offload API”
  - Queues the Work Requests
  - Checks for Work Request completion

- **SPE Code**: “SPE Runtime”
  - Coordinates all other Work Request activities
    - Set-aside LS memory
    - Issue DMA-Get for input data
    - Execute Work Request
    - Issue DMA-Put for output data
[1] : PPE Sends Work Request
[3] : DMA-Get used to retrieve input data from system memory
[4] : Work Request is executed
[5] : DMA-Put used to place output data into system memory
[6] : SPE notifies PPE of Work Request Completion

(NOTE : Not to Scale)
Offload API

- **Interface**: Functions to...
  - Init/Close the Interface
  - Send Work Requests (WR)
    - Standard: 1 read/write, 1 read-only, 1 write-only
    - Scatter/Gather: DMA List (many buffers of each type)
  - Work Request Groups (WRGroup)
  - Check for Completions (next slide)
  - Make Progress
Offload API

- Notification of Completion
  - Callback
    - Each WR/WRGroup can specify a callback function
    - Default callback can be set
  - If no callback specified, user must explicitly check for completion using a Handle
    - Blocking
    - Polling/Non-Blocking
Status

- Charm++ runs on the Cell Blade using the Offload API
  - Requires some code modification
- Preliminary implementation of NAMD
NAMD: A Production MD Program

- Fully featured program
- NIH-funded development
- Distributed free of charge (~17,000 downloads so far)
- Binaries and source code
- Installed at NSF centers
- User training and support
- Large published simulations (e.g., aquaporin simulation featured in keynote)
NAMD

- NAnoscale Molecular Dynamics
- Molecular Dynamics
  - Classical Physics
  - Bio-Molecules
- Developed and Maintained by
  - Theoretical and Computational BioPhysics Group (Beckman Institute, University of Illinois at Urbana-Champaign)
  - Parallel Programming Lab (Computer Science Department, University of Illinois at Urbana-Champaign)
NAMD

**Decomposition**

- 3D space broken up into a grid of 3D cubes called “Patches”
- Compute objects calculate forces between atoms for each neighboring pair of Patches

- **Non-Bonded**
  - Basically $O(N^2)$: Each atom in one patch interacts with every atom in the other patch
  - Approximately 80% of work performed each timestep

- **Bonded**
NAMD on the Cell

- Offloading Non-Bonded Force
  - Input: Atom lists, simulation parameters
  - Output: Force arrays
  - Directly using Offload API (currently)
Conclusions

- Charm++ fits well with Cell Architecture
- Current Status: Charm++ applications can utilize SPEs (with modification)
- Offload API
  - Easy-to-use
  - Simplifies Code
  - Independent of Charm++
Future Work

- **Offload API**
  - Continue to add features
    - Load-Balancing (between SPEs)
    - Moving SPE code with Work Request
    - SPE Affinity (code and data)
  - Performance Analysis

- **Charm++ Runtime System**
  - Modify charmxi (Charm++ translator) to auto generate Offload API code when compiling on Cell-based systems
  - Applications: NAMD, Cosmology
Acknowledgments

- National Institutes of Health
  - Grant: NIH PHS 5 P41 RR05969-04

- IBM
  - Specifically Hema Reddy
  - Cell Group in General
More Information...

- Charm++ / Offload API
  - http://charm.cs.uiuc.edu
- NAMD
  - http://www.ks.uiuc.edu/Research/namd
Questions?
Offload API Code Example

```c
#include <stdio.h>
#include <string.h>
#include <spert_ppu.h> // Offload API Header
#include "hello_shared.h"
#define NUM_WORK_REQUESTS 10

int main(int argc, char* argv[]) {
    WRHandle wrHandle[NUM_WORK_REQUESTS];
    char msg[] __attribute__((aligned(128))) = { "Hello" };  
    int msgLen = ROUNDUP_16(strlen(msg));
    InitOffloadAPI();

    // Send some work requests
    for (int i = 0; i < NUM_WORK_REQUESTS; i++)
        wrHandle[i] = sendWorkRequest(FUNC_SAYHI, NULL, 0, msg, msgLen, NULL, 0);

    // Wait for the work requests to finish
    for (int i = 0; i < NUM_WORK_REQUESTS; i++)
        waitForWRHandle(wrHandle[i]);

    CloseOffloadAPI();
    return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include "spert.h" // SPE Runtime Header
#include "hello_shared.h"

inline void sayHi(char* msg) {
    printf(""%s" from SPE %d...\n", msg, (int)getSPEID());
}

#ifdef __cplusplus
extern "C"
#endif
void funcLookup(int funcIndex, void* readWritePtr, int readWriteLen,
                 void* readOnlyPtr, int readOnlyLen,
                 void* writeOnlyPtr, int writeOnlyLen,
                 DMAListEntry* dmaList) {
    switch (funcIndex) {
    case SPE_FUNC_INDEX_INIT: break;
    case SPE_FUNC_INDEX_CLOSE: break;
    case FUNC_SAYHI:
        sayHi((char*)readOnlyPtr);
        break;
    default: // should never occur
        printf("ERROR :: Invalid funcIndex (%d)\n", funcIndex);
        funcIndex);
    }
}
```

```c
#include <stdio.h>
#include "spert.h" // SPE Runtime Header
#include "hello_shared.h"

inline void sayHi(char* msg) {
    printf(""%s" from SPE %d...\n", msg, (int)getSPEID());
}

int main(int argc, char* argv[]) {
    WRHandle wrHandle[NUM_WORK_REQUESTS];
    char msg[] __attribute__((aligned(128))) = { "Hello" };  
    int msgLen = ROUNDUP_16(strlen(msg));
    InitOffloadAPI();

    // Send some work requests
    for (int i = 0; i < NUM_WORK_REQUESTS; i++)
        wrHandle[i] = sendWorkRequest(FUNC_SAYHI, NULL, 0, msg, msgLen, NULL, 0);

    // Wait for the work requests to finish
    for (int i = 0; i < NUM_WORK_REQUESTS; i++)
        waitForWRHandle(wrHandle[i]);

    CloseOffloadAPI();
    return EXIT_SUCCESS;
}
```

Output

```
"Hello" from SPE 0...
"Hello" from SPE 1...
"Hello" from SPE 2...
"Hello" from SPE 3...
"Hello" from SPE 4...
"Hello" from SPE 5...
"Hello" from SPE 6...
"Hello" from SPE 7...
```

---

**Illiinois**

**University of Illinois at Urbana-Champaign**

**PARALLEL Programming Lab**
Charm++ & Offload API

Development Effort

- Phase 1: (completed in a couple of hours)
  - Goal: Allow Charm++ applications to execute on PPE
  - Port Charm++ Runtime System to PPE

- Phase 2: (well-underway, already useful)
  - Goal: Allow C/C++ applications and Charm++ applications to effectively utilize the SPEs in a clear and straightforward manner
  - Create Offload API
  - Add hooks into Charm++ Runtime System for Offload API

- Phase 3: (will be started in the near future)
  - Goal: Allow Charm++ applications to be portable between Cell-based and non-Cell-based platforms with little or no modification to application code
  - Modify Charm++ translator (charmx) to automatically generate Offload API code from the application code
Charm++ & Offload API