Landing GNU-Based OpenMP on CELL: Progress Report and Perspectives

Guang R. Gao
Computer Architecture and Parallel System Laboratory
Department of Electrical & Computer Engineering
University of Delaware
ggao@capsl.udel.edu
Outline

- Background
- Why GNU OpenMP on CELL?
- Project Status Report
- Preliminary Results
- Future Perspectives
CAPSL Research Layout
High End Computing Architecture & Programming Models

- Scientific Computation Kernels
- Other High End Applications
- High Performance Bio-computing Kernels

Advance Execution / Programming Models
- Percolation
- Location Consistency

Infrastructure & Tools
- Analytical Modeling
- System Tools
- Simulation/Emulation

Base Execution Model
- Fine Grained Multithreading (i.e. EARTH, CARE)
Outline

- Background
- Why GNU OpenMP on CELL?
- Project Status Report
- Preliminary Results
- Future Perspectives
CBE Architecture Overview

Local storage size per SPU: 256KB
Area: 221 mm²
Technology: 90nm SOI
Total number of transistors: 234M

Observed clock speed: a wide range of operating frequencies are supported to optimize for power and yield;
Peak performance (single precision): > 256 GFlops
Peak performance (double precision): > 26 GFlops
State on Parallel Languages

(based on a recent survey by G. Pfister, IBM)

- 200+ parallel language efforts in the past.
- At first glance: Most of them are not used!!!
- When talking about parallel languages, you usually hear MPI (90% of the time) and OpenMP (10%)
- Auto-parallelization has drifted from the general scene toward obscurity.
Why OpenMP?

- OpenMP is an industrial standard for writing parallel programs on shared memory architecture.
- OpenMP is available.
- OpenMP is being productively used.
- OpenMP is …
OpenMP
Major Issues and Challenges

- For Compiler Writers, not Users
- Pragma / Directive Based
- Default is set to make it easy to write fast (but not necessarily correct) programs.
- OpenMP does not support sequential consistency
- Data layout and locality management
- **Lack of support for OpenMP by the GCC compilers for the CBE.**
- It only has 10% of parallel programming user community.

ACK: this list comes from private communication with a number of people: William Gropp, John, Mellor-Crummey, Rick Stevens, Thomas Sterling, Ross Towle, Kathy Yelick, etc.
"I think it's a waste of time to focus on trying to force these old broken poor parallel processing languages/protocols into the new approach."

However OpenMP is widely available today

- This is evident from its inclusion in the GNU Compiler Collection in Release 4.2.0
GOMP Status

- OpenMP support for C, C++, and Fortran 95
- Will support 2.5 and 3.0 soon
- Released in May 5, 2007 as part of the official release of GCC 4.2
Outline

- Background
- Why GNU OpenMP on CELL?
- Project Description
- Preliminary Results
- Future Perspectives
GNU Based OpenMP on CELL

Objectives

- A working OpenMP-CELL platform
- Has the following features
  - Single source compilation
  - Code partition and overlay
  - Software caching
  - A simple runtime system
- Should finish in 1-yr, and pass a set of (non-toy) benchmarks and publish papers
- Optimization is NOT an objective, but
- Should propose a wish list of research topics for the next phase
- Try to leverage knowledge/experience from the Cyclops-64 project
**Single Source Compilation**

Progress Report

- **Source Code**
  - Modified compiler, assembler, and linker

- **SPU-cc**
  - Partition creation by clustering
  - Addition of assembly directives
  - Insertion of library calls
  - Outlining of parallel functions

- **PPU-cc**
  - Insertion of library calls
  - Outlining of sequential code

- **SPU exec**
  - SPU binary plus partition manager and software cache libraries

- **PPU exec**
  - PPU binary plus GOMP & SPE libraries

- **Embedder**
  - Final Executable with all the necessary (static) libraries

- **Final Executable**
  - Final Executable with all the necessary (static) libraries
The Code Overlay Problem

Large Applications must be partitioned and stored in Main Memory

A Heterogeneous Architecture

Main Processor

Interconnection Network

I/O Interface

Memory Interface

Local Memory

Code Overlay Region

Data Buffers / Software Cache

Resident Code / Micro Kernel

Local memory is small
Our Code Overlay Manager

Features
- Semi-static sub-division of buffer
- Replacement policies and buffer behaviors
  - LRU vs. other replacement Policies
  - Lazy Reuse [cache-like] Buffer Behavior

Modified Toolchain
- User aided and automatic code partitioning
- Command line options

Remarks
- compiler does no need to break object code into multiple files, and explicitly put the names of the files into a linker script,
- simply link the partition manager library and use the default GNU linker script
Software Cache

Why software caching?

Features:
- Cache-Coherence enforced at synchronization points (e.g. barrier, lock, etc.)
- Handle false-sharing at byte level

Other cache design decisions
- Cache parameters (32-bit address, block size: 128B, 128 blocks (16k))
- Cache organization (set-associative, current: 4W)
- Write back vs. write through
- Replacement policy: LRU

Remark: Only used as a backup solution
## Software Cache

**An Overview**

- Smooth the heterogeneity among different memory modules;
- The SPEs can simultaneously source/sink 8 bytes per processor cycles (25.6+25.6GB/s at 3.2GHz);
- 6 cycle load latency to 256KB local storage (LS) on SPE;
- Bytewise dirty bits but is adaptive;
- Cache line fill/flush are performed via DMA transfer;

### Memory Hierarchy

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPU</td>
<td>Processor Peripheral Unit</td>
</tr>
<tr>
<td>SPU</td>
<td>Stream Processing Unit</td>
</tr>
<tr>
<td>LS</td>
<td>Local Storage</td>
</tr>
<tr>
<td>Main Mem</td>
<td>Main Memory</td>
</tr>
</tbody>
</table>

### Hardware Components

- **Element Interconnect Bus**
- **0-16 bytes**
- **4 bytes**
- **128 bytes**

### Memory Accesses

- **dirty bit vector**
- **tag & status**
- **data**
A Simple Runtime System

Why a simple runtime system?

Features of our simple runtime system

- Shadow (PPU) threads and worker (SPU) threads

Mainly used for testing the compiler and tool-chain
A Simple Runtime System
An Overview

Thr 0 serves as the Master Thread and creates all other threads.
A Simple Runtime System

Threads and Communication

- POSIX Thread
- SPU Thread

- Send Task
- PPU Task Function
- Request Handler

- Receive Task
- SPU Task Function
  - Computational Work
  - GOMP Runtime Function
  - Computational Work
  - Task Signal Completion Send

- Initial Signal
- Command Buffer reply
- Command Buffer request
- Completion signal
- Incoming signal
- Outgoing signal
Status Summary

- Code partition between SPU and PPU
  - Single source compilation
  - Outline parallel sections for SPU
- Explicit data movement between main memory and SPU
  - Software cache
  - Double buffering
- Code overlay to support large programs
  - Code partition support by the tool-chain
  - Object code format changes
  - Partition manager: decide when to load a new partition
- OpenMP runtime
  - PPU and SPU work together
Outline

- Background
- Why GNU OpenMP on CELL?
- Project Description
- Preliminary Results
- Future Perspectives
Experimental Framework

- Tool-chain
  - CELL SDK v1.1

- Modified components
  - spu-ld v2.16.1
  - spu-as v2.16.1
  - spu-gcc v4.2.0

- Extra libraries
  - Software cache
  - Partition Manager

Yellow Dog Linux v5.0

PS3* Hardware

*PS3 is a trademark of Sony corporation
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff, huff2</td>
<td>Huffman decoding from MPEG2</td>
</tr>
<tr>
<td>idct, idct_2</td>
<td>IDCT and IQquantization from MPEG2</td>
</tr>
<tr>
<td>resize, reside_2</td>
<td>YUV file resizing algorithm</td>
</tr>
<tr>
<td>alphablend</td>
<td>A process of combining a translucent foreground color with a background (stream) file</td>
</tr>
<tr>
<td>convert</td>
<td>YUV2RGB - convert yuv file to raw stream file</td>
</tr>
<tr>
<td>prgb2gm</td>
<td>Convert RBB file into BMP file</td>
</tr>
<tr>
<td>gzip</td>
<td>SPEC compression utility</td>
</tr>
<tr>
<td>OpenMP Validation Suite V1.0</td>
<td>OpenMP test cases from University of Houston</td>
</tr>
</tbody>
</table>
Preliminary Experimental Results

- Pass preliminary tests for all benchmarks
- The automatic code overlay works
  - provides important performance gains for different applications
  - Modulus is better when the code / partitions have no re-use
  - LRU is better when the code / partitions have re-use
  - Degradation ➔ 8 % in the worst case
Outline

- Background
- Project and Problem Formulation
- Status Report
- Results
- Related Work
- Future Perspectives
Related Work

- Manzano, et. all, IWOMP 2007
- O'Brien, et. all, IWOMP 2007
- Software model for Cyclops-64 chips/system
- Many others (including work presented in this workshop)
Summary and Future Work

A preliminary GNU OpenMP on CELL is in good progress.

Release plan

- Alpha release is being planned
- Beta release partner(s) – please contact us

Compiler optimization and infrastructure
Acknowledgement

- The CELL team at UD and ETI
- Other Members of CAPSL
- IBM CELL team (especially, Peter Hofstee, Michael Gschwind, Kevin O’Brian, etc.)
- Other collaborators
- Our hosts