Productive Performance on Accelerator Based Systems Using OpenACC Compilers and Tools

Luiz DeRose
Programming Environments Director
Cray Inc.
Thanks to Partner AMD for Sponsorship!
The Cray XK6 Hybrid Architecture

Accelerating the Way to Better Science

- Announced in May 2011

- NVIDIA Fermi X2090 GPU
  - Upgradable to Kepler

- AMD Interlagos CPU

- Cray Gemini interconnect
  - high bandwidth/low latency scalability

- Unified X86/GPU programming environment

- Fully compatible with Cray XE6 product line

- Fully upgradeable from Cray XT/XE systems
Cray Vision for Accelerated Computing

**Most important hurdle** for widespread adoption of accelerated computing in HPC is **programming difficulty**

- Need a single programming model that is **portable across machine types**
  - **Portable** expression of heterogeneity and multi-level parallelism
  - Programming model and optimization should not be significantly different for “accelerated” nodes and multi-core x86 processors
  - **Allow users to maintain a single code base**

**Cray’s approach to Accelerator Programming** is to provide a tightly coupled high level programming environment with compilers, libraries, and tools that can **hide the complexity** of the system

**Ease of use** is possible with

- Compiler making it **feasible for users** to write applications in **Fortran, C, and C++**
- Tools to help users port and optimize for hybrid systems
- Auto-tuned scientific libraries
Programming for a Node with Accelerator

- Fortran, C, and C++ compilers
  - OpenACC directives to drive compiler optimization
  - Compiler does the “heavy lifting” to split off the work destined for the accelerator and perform the necessary data transfers
  - Compiler optimizations to take advantage of accelerator and multi-core X86 hardware appropriately
  - Advanced users can mix CUDA functions with compiler-generated accelerator code

- Cray Reveal, built upon an internal compiler representation of the application
  - Source code browsing tool that provides interface between the user, the compiler, and the performance tool
    - Scoping tool to help users port and optimize applications
    - Performance measurement and analysis information for porting and optimization

- Tools
  - Parallel Debugger support with DDT or TotalView
  - Working on Comparative Debugging with Monash University

- Scientific Libraries support
  - Auto-tuned libraries (using Cray Auto-Tuning Framework)
OpenACC Accelerator Programming Model

● Why a new model? There are already many ways to program:
  ● CUDA and OpenCL
    ● All are quite low-level and closely coupled to the GPU
  ● PGI CUDA Fortran
    ● Still CUDA just in a better base language
  ● PGI accelerator directives, CAPS HMPP
    ● First steps in the right direction – Needed standardization

● User needs to write specialized kernels:
  ● Hard to write and debug
  ● Hard to optimize for specific GPU
  ● Hard to update (porting/functionality)

● OpenACC Directives provide high-level approach
  ● Simple programming model for hybrid systems
  ● Easier to maintain/port/extend code
    ● The same source code can be compiled for multicore CPU
  ● Based on the work in the OpenMP Accelerator Subcommittee
  ● Possible performance sacrifice
    ● A small performance gap is acceptable (do you still hand-code in assembler?)
    ● Goal is to provide at least 80% of the performance obtained with hand coded CUDA

● Compilers support: all complete in 2012
  ● Cray CCE: complete in the 8.1 release (September 2012)
  ● PGI Accelerator version 12.6 onwards
  ● CAPS Full support in version 1.3
Cray XK6 Performance Tools

- **Scaling (running big jobs with a large number of GPUs)**
  - Results summarized and consolidated in one place

- **Statistics for the whole application**
  - Performance statistics mapped back to the user source by line number
  - Performance statistics grouped by accelerator directive
  - Single report can include statistics for both the host and the accelerator

- **Single tool for GPU and CPU performance analysis**
  - Performance statistics
    - Includes accelerator time, host time, and amount of data copied to/from the accelerator
  - Kernel level statistics
  - Accelerator hardware counters
Types of Statistics

- **Loop work estimates**
  - Provide information to identify important loops

- **Performance statistics**
  - Includes accelerator time, host time, and amount of data copied to/from the accelerator

- **Accelerator hardware counters**
  - Hardware counters on the accelerator itself.
    - On NVIDIA Fermi GPUs, there are about 50 available counters

- **Kernel level statistics**
  - Includes stats about grid size, block size, and occupancy
A Peek at Something New: Reveal

New code restructuring and analysis assistant…

Uses both the performance toolset and CCE’s program library functionality to provide static and runtime analysis information

Assists user with the code optimization phase by correlating source code with analysis to help identify which areas are key candidates for optimization

Key Features

Annotated source code with compiler optimization information

- Provides feedback on critical dependencies that prevent optimizations

Scoping analysis

- Identifies shared, private and ambiguous arrays
- Allows user to privatize ambiguous arrays
- Allows user to override dependency analysis

Source code navigation

- Uses performance data collected through CrayPat
What is Cray Libsci_acc?

- Selects best GPU kernel for the current task based on:
  - Problem, Problem Size, Data Size

- Selects best Kernel from:
  - Cray tuned kernels (ATF)
  - cuBlas, magmaBlas
  - Other available sources

- Provides two sets of interfaces to be used in difference scenarios with *minimized code modifications*
  - Basic Interface:
    - Data copy is automatic
    - GPU or CPU execution placement is automatic
    - Automatic Memcopy optimizations
      - Copy only necessary data (submatrix copy, basic interface)
  - Advanced Interface:
    - Data placement done by user
    - CCE Integration
Porting Himeno to the Cray XK6

- Several versions tested, with communication implemented in MPI and Fortran coarrays

- GPU version using OpenACC accelerator directives
  - Total number of accelerator directives: 27
    - plus 18 "end" directives

- Arrays reside permanently on the GPU memory

- Data transfers between host and GPU are:
  - Communication buffers for the halo exchange
  - Control value

- Cray XK6 timings compared to best Cray XE6 results (hybrid MPI/OpenMP)
Himeno performance

- XK6 GPU is about 1.6x faster than XE6
- OpenACC async implementation is ~ 8% faster than OpenACC blocking

All comparisons are for strong scaling on case XL
Computational chemistry package suite developed and maintained by the Gordon Group at Iowa State University

- http://www.msg.ameslab.gov/gamess/

Isolated computationally intensive kernel called CCSD(T)

- Method to calculate electronic correlation energy in water clusters
- ijk-tuples kernels contains iterations of:
  - Communication
  - Complex array transformations
  - Matrix-matrix multiplies
- Ideal for GPU execution
  - Data movement between host and device can be minimized
  - Kernel is compute intensive with many matrix multiplies
  - Data scrambling can be done on device

Other kernels (IJJ-tuples and IIJ-tuples) were also converted, but we do not have corresponding CUDA versions for comparisons
OpenACC vs. CUDA (vs. Fortran on CPU)

● **Source changes**
  ● The OpenACC conversion was accomplished with the addition of approximately 70 OpenACC directives to the original Fortran source
  ● The hand coded CUDA kernel contains approximately 1800 lines of code

● **Performance of ijk-tuples kernels on 16 XK Nodes**
  ● OpenACC – 138 seconds
  ● CUDA – 134 seconds

● **Performance of ijk-tuples kernels on 16 XE Nodes (256 ranks)**
  ● 311 Seconds
Summary

- Hybrid multicore has arrived and is here to stay
  - Fat nodes are getting fatter
  - GPUs have leapt into the Top500 and accelerated nodes

- Programming accelerators efficiently is hard
  - Need a high level programming environment
    - Cray Compilation Environment (CCE) focused on ease-of-use
      - OpenACC support
      - "Program Library" provides application specific repository for information for compiler and tools
    - Cray Reveal
      - Assists user in understanding their code and taking full advantage of SW and HW system
    - Cray Performance Analysis Toolkit
      - Single tool for GPU and CPU performance analysis with statistics for the whole application
    - Cray Auto-Tuning Libraries
      - Getting performance from the system … no assembly required