Portable and Productive Performance with OpenACC Compilers and Tools

Luiz DeRose
Sr. Principal Engineer
Programming Environments Director
Cray Inc.
We provide a complete turn key solution to help solve challenges in science and engineering that require supercomputing.
The New Generation of Supercomputers

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

- Programming accelerators efficiently is challenging
  - Three levels of parallelism required
    - MPI between nodes or sockets
    - Shared memory programming on the node
    - Vectorization for low level looping structures
  - Need a hybrid programming model to support these new systems
  - Need a high level programming environment
    - Compilers, tools, & libraries
The Cray XK7 Hybrid Architecture

- NVIDIA Kepler GPUs
- 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
Major Hybrid Multi Petaflop Systems

Blue Waters: Sustained Petascale Performance
- Production Science at Full Scale
- 244 Cray XE Cabinets + 32 Cray XK Cabinets
- 11.5 Petaflops
- 1.5 Petabytes of total memory
- 25 Petabytes Storage
- Cray’s scalable Linux Environment
- HPC-focused GPU/CPU Programming Environment

Titan: A “Jaguar-Size” System with GPUs
- 200 cabinets
- 18,688 compute nodes
- 25x32x24 3D torus (22.5 TB/s global BW)
- 128 I/O blades (512 PCIe-2 @ 16 GB/s bidir)
- 1,278 TB of memory
- 4,352 sq. ft.

“Piz Daint”
Cray has signed a contract with the Swiss National Supercomputing Centre (CSCS) to expand its Cray XC30 supercomputer with NVIDIA Kepler GPUs. When the upgrade is completed, the Centre’s Cray XC system, will be the first petascale supercomputer in Switzerland.
Cray’s 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 difference 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 an **ease of use** 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**
  - **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
- **Debugger support** with DDT and TotalView
- **Cray Reveal**, built upon an internal compiler database containing a representation of the application
  - Source code browsing tool that provides interface between the user, the compiler, and the performance analysis tool
    - **Scoping tool** to help users port and optimize applications
    - **Performance measurement and analysis** information for identification of main loops of the code to focus refactoring
- **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
    - 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
    - Non-executable statements (comments, pragmas)
    - The same source code can be compiled for multicore CPU
  - Based on the work in the OpenMP Accelerator Subcommittee
  - PGI accelerator directives, CAPS HMPP
    - First steps in the right direction – Needed standardization
  - **Possible performance sacrifice**
    - A small performance gap is acceptable (do you still hand-code in assembly?)
    - Goal is to provide at least 80% of the performance obtained with hand coded CUDA
- **Compiler support: all complete in 2012**
  - Cray CCE: complete in the 8.1 release
  - PGI Accelerator version 12.6 onwards
  - CAPS Full support in version 1.3
The Cray Compilation Environment

- Cray technology focused on scientific applications
  - Takes advantage of **automatic vectorization**
  - Takes advantage of **automatic shared memory parallelization**

- **OpenACC 1.0 compliant (working on OpenACC 2.0)**
  - CCE Identifies parallel loops within code regions
  - Splits the code into accelerator and host portions
  - Workshares loops running on accelerator
    - Make use of MIMD and SIMD style parallelism
  - Data movement
    - allocates/frees GPU memory at start/end of region
    - moves data to/from GPU
  - Compiles to PTX not CUDA
  - Single object file
  - Debuggers see original program not CUDA intermediate
Steps to Create a Hybrid Code

1. **Identify possible accelerator kernels**
   - Determine where to add additional levels of parallelism
   - Assumes MPI application is functioning correctly on X86
   - Find top serial work-intensive loops (perftools + CCE loop work estimates)

2. **Perform parallel analysis, scoping and vectorization**
   - Split loop work among threads
     - Do parallel analysis and restructuring on targeted high level loops
     - Use CCE loopmark feedback, Reveal loopmark and source browsing

3. **Move to OpenMP and then to OpenACC**
   - Add parallel directives and acceleration extensions
     - Insert OpenMP directives (Reveal scoping assistance)
     - Run on X86 to verify application and check for performance improvements
     - Convert desired OpenMP directives to OpenACC

4. **Analyze performance from optimizations**
Moving to a Hybrid or Many-core Environment

New analysis and code restructuring 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
Reveal with Loop Work Estimates

Try "Getting Started" in the "Help" Menu
Visualize Loopmark with Performance Information

- Performance feedback
- Loopmark and optimization annotations
- Compiler feedback
Compiler Messages Explanations

Integrated message ‘explain support’
View Pseudo Code for Inlined Functions

Inlined call sites marked

Expand to see pseudo code
Scoping Assistance – Review Scoping Results

Loops with scoping information are highlighted – red needs user assistance.

Parallelization inhibitor messages are provided to assist user with analysis.

User addresses parallelization issues for unresolved variables.
Scoping Assistance – User Resolves Issues

Use Reveal’s OpenMP parallelization tips

Click on variable to view all occurrences in loop
Scoping Assistance – Generate Directive

Reveal generates example OpenMP directive
OpenACC Debugging

- On device debugging with Allinea DDT
  - Variables – arrays, pointers, full F90 and C support
  - Set breakpoints and step warps and blocks

- Requires Cray compiler for on-device debugging
  - Other compilers to follow

- Identical to CUDA
  - Full warp/block/kernel controls

Step host & device
View variables
Set breakpoints

Compatibility with Cray CCE 8.x
OpenACC
Cray 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
Cray Apprentice2 Overview with GPU Data
Accelerator Statistics View from Apprentice2

Table 4: Time and Bytes Transferred for Accelerator Regions

<table>
<thead>
<tr>
<th>Host</th>
<th>Host</th>
<th>Acc</th>
<th>Acc Copy</th>
<th>Acc Copy</th>
<th>Events</th>
<th>Calltree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>Time</td>
<td>Time</td>
<td>In</td>
<td>Out</td>
<td>PE=IDE</td>
<td>(MBytes)</td>
</tr>
<tr>
<td>100.0%</td>
<td>24,777</td>
<td>24,244</td>
<td>2555</td>
<td>2560</td>
<td>38164</td>
<td>Total</td>
</tr>
<tr>
<td>90.0%</td>
<td>24,777</td>
<td>24,244</td>
<td>2555</td>
<td>2560</td>
<td>38164</td>
<td>Total</td>
</tr>
<tr>
<td>100.0%</td>
<td>24,773</td>
<td>24,187</td>
<td>2555</td>
<td>2560</td>
<td>38152</td>
<td>Total</td>
</tr>
<tr>
<td>4%</td>
<td>15.6%</td>
<td>3.863</td>
<td>1.115</td>
<td>--</td>
<td>2555</td>
<td>3809</td>
</tr>
</tbody>
</table>

Wallclock time: 26.309761s

himeno_mpi2 (1,600 events in 0.444s)
Call Tree with GPU regions
What is Cray Libsci_acc?

- **Provide basic scientific libraries optimized for hybrid systems**
  - Incorporate the existing GPU libraries into Cray libsci

- **Independent to, but fully compatible with OpenACC**

- **Multiple use case support**
  - Get the base use of accelerators with no code change
  - Get extreme performance of GPU with or without code change

- **Provide additional performance and usability**

- **Two interfaces**
  - **Simple interface**
    - **Auto-adaptation**
    - Base performance of GPU with minimal (or no) code change
    - Target for anybody: non-GPU users and non-GPU expert

  - **Expert interface**
    - Advanced performance of the GPU with controls for data movement
    - Target for CUDA, OpenACC, and GPU experts
    - **Does not imply that the expert interfaces are always needed to get great performance**

Cray Inc. Proprietary © 2013
Adaptation in the Simple Interface

- You can pass either host pointers or device pointers with the simple interface.

- A is in host memory
  
  \[
  \text{dgetrf}(M, N, A, \text{lda}, \text{ipiv}, \&\text{info})
  \]
  
  - Performs hybrid operation on GPU
  - If problem is too small, performs host operation.

- Pass Device memory
  
  \[
  \text{dgetrf}(M, N, \text{d}_A, \text{lda}, \text{ipiv}, \&\text{info})
  \]
  
  - Performs hybrid operation on GPU

- BLAS 1 and 2 performs computation local to the data location
  
  - CPU-GPU data transfer is too expensive to exploit hybrid execution
Libsci_acc: Simple Interface for BLAS3 and LAPACK

User Application

dgemm_();

where is the data?

Libsci_acc DGEMM_ACC

Large enough?

Libsci_acc Hybrid DGEMM

Libsci DGEMM

Cray Inc. Proprietary © 2013
Case Study: the Himeno Benchmark

- Parallel 3D Poisson equation solver
  - Iterative loop evaluating 19-point stencil
  - Memory intensive, memory bandwidth bound

- Fortran, C, MPI and OpenMP implementations available from http://accc.riken.jp/HPC_e/himenobmt_e.html

- Strong scaling benchmark
  - XL configuration: 1024 x 512 x 512 global volume
  - Expect halo exchanges to become significant
  - Use asynchronous GPU data transfers and kernel launches to help avoid this
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

![Himeno Benchmark - XL configuration graph](image)
CloverLeaf

● 2D hydro code, with several stencil-type operations

● Developed by AWE
  ● Using to explore programming models
  ● to be released as Open Source to the Mantevo project hosted by Sandia (miniapps)

● Current performance for 87 steps

<table>
<thead>
<tr>
<th>Mesh</th>
<th>CUDA</th>
<th>OpenACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>960x960</td>
<td>2.44</td>
<td>2.03</td>
</tr>
<tr>
<td>3840x3840</td>
<td>37.42</td>
<td>31.77</td>
</tr>
</tbody>
</table>
**GAMESS**

- Computational chemistry package suite developed and maintained by the Gordon Group at Iowa State University
  - [http://www.msg.ameslab.gov/gamess/](http://www.msg.ameslab.gov/gamess/)

**ijk-tuples kernel - Source changes**
- CUDA - **1800 lines of hand-coded** CUDA
- OpenACC – approximately **75 directives added** to the original source

**Performance of ijk-tuples on 16 XK6 Nodes with Fermi**
- CPU Only (16 ranks per node) 311 Seconds
- CUDA – 134 seconds
- OpenACC – 138 seconds
- **CUDA was only ~3% faster than OpenACC**

**Performance of ijk-tuples on 16 XK6 Nodes with Kepler**
- CPU Only (16 ranks per node) 311 Seconds
- CUDA – 76.6 seconds
- OpenACC – 68.1 seconds
- **OpenACC was ~12.5% faster than CUDA !!**
Summary

- Cray provides a high level programming environment for accelerated Computing
  - Fortran, C, and C++ compilers
    - **OpenACC directives to drive compiler optimization**
    - Compiler optimizations to take advantage of accelerator and multi-core X86 hardware appropriately

- Cray **Reveal**
  - **Scoping analysis** tool to assist user in understanding their code and taking full advantage of SW and HW system

- **Cray Performance Measurement and Analysis toolkit**
  - Single tool for GPU and CPU performance analysis with statistics for the whole application

- **Parallel Debugger support** with **allinea** [www.allinea.com](http://www.allinea.com) DDT or TotalView

- Auto-tuned Scientific Libraries support
  - Getting performance from the system … **no assembly required**