Cray Programmierumgebung für GPUs

ZKI-Arbeitskreis Supercomputing
19.-20.5.2011
DESY, Zeuthen

May 19, 2011
<table>
<thead>
<tr>
<th>Node Characteristics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cores</td>
<td>24 (Magny Cours)</td>
</tr>
<tr>
<td>Peak Performance MC-12 (2.2)</td>
<td>211 Gflops/sec</td>
</tr>
<tr>
<td>Peak Performance MC-8 (2.4)</td>
<td>153 Gflops/sec</td>
</tr>
<tr>
<td>Memory Size</td>
<td>32 GB per node</td>
</tr>
<tr>
<td></td>
<td>64 GB per node</td>
</tr>
<tr>
<td>Memory Bandwidth (Peak)</td>
<td>83.5 GB/sec</td>
</tr>
</tbody>
</table>

10 12X Gemini Channels (Each Gemini acts like two nodes on the 3D Torus)

High Radix YARC Router with adaptive Routing
168 GB/sec capacity
Cray Gemini Network

- System on a Chip design
  - 2 HyperTransport NICs
  - Embedded high performance router
- 3D Torus network
  - XT5/XT6 systems field upgradable
- Advanced features
  - Global memory access
  - Atomic memory operations
  - Hardware support for PGAS languages
- Key performance metrics
  - MPI latency as little as 1.4 μsec
  - MPI message rate of 9M/sec
  - Injection bandwidths in excess of 6 GB/sec
  - Cray SHMEM put rate of 25M/sec
  - Scattered/indexed put rates of 60-90M/sec
Why Accelerated Computing?

- The great technology inflection point of mid last decade
  - Power has become a huge cost and the major design constraint
- Multi-core x86 was a great first response...
- ...but it won’t get us where we need to go
  - Standard cores are designed to run threads *fast*, not power-efficiently
- Conclusion
  - Need lots of very simple cores optimized for power on parallel code
  - But also need some fast cores for serial code
  ⇒ heterogeneous processing (a.k.a. accelerators)
- GPUs are looking very promising for HPC
  - They now have good Double Precision FP and error protection
  - They ship O(100M) units a year
  - It looks like they can credibly support both masters
# Cray Glacier Node

<table>
<thead>
<tr>
<th>Glacier Compute Node Characteristics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Processor</td>
<td>AMD Series 6100 (Interlagos)</td>
</tr>
<tr>
<td>Host Processor</td>
<td>90-147 Gflops</td>
</tr>
<tr>
<td>Tesla X2090 Cores</td>
<td>448</td>
</tr>
<tr>
<td>Tesla X2090 Perf.</td>
<td>600+ Gflops</td>
</tr>
<tr>
<td>Host Memory</td>
<td>16 or 32GB 1300 MHz DDR3</td>
</tr>
<tr>
<td>Tesla X090 Memory</td>
<td>6GB GDDR5 capacity 170 GB/sec</td>
</tr>
<tr>
<td>Gemini High Speed Interconnect</td>
<td>Upgradeable to Tesla follow on</td>
</tr>
</tbody>
</table>

### XE6 Node Characteristics
- Number of Cores: 24 (Magny Cours)
- Peak Performance: 128 (2.2) 211 Gflops/sec
- Peak Performance: 80 (2.4) 153 Gflops/sec
- Memory Size: 32 GB per node
- Memory Bandwidth (Peak): 83.5 GB/sec

### Glacier Compute Node Characteristics
- Host Processor: AMD Series 6100 (Interlagos)
- Host Processor: 90-147 Gflops
- Tesla X2090 Cores: 448
- Tesla X2090 Perf.: 600+ Gflops
- Host Memory: 16 or 32GB 1300 MHz DDR3
- Tesla X090 Memory: 6GB GDDR5 capacity 170 GB/sec
- Gemini High Speed Interconnect: Upgradeable to Tesla follow on

### 10 12X Gemini Channels
- (Each Gemini acts like two nodes on the 3D Torus)

### High Radix YARC Router with adaptive Routing
- 168 GB/sec capacity

May 19, 2011
Cray Proprietary
Glacier Accelerator Blade

- Four nodes per blade
- Gemini Mezzanine
- Plug compatible with Cray XE6 blade
- Configurable processor, memory and SXM GPU card
- Prototype modules are powered up and running OS & initial software
● Provide a **tightly coupled** programming environment with compilers, libraries, and tools that will **hide the complexity** of the system
  ● Focus on integration and differentiation
  ● Target **ease of use** with extended **functionality** and increased automation

● Ease of use will be possible with
  ● Compiler making it feasible for users to write applications in Fortran, C, C++
  ● Runtime to allow for nested parallel structures and calling of routines on the accelerator
  ● Tools to help users port and optimize for accelerators
  ● Auto-tuned scientific libraries
OpenMP for Accelerators

- OpenMP provides for manual shared-memory parallelism support across X86 and accelerator cores
- Cray is extending the OpenMP 3.0 standard for accelerator support
  - Cray co-chairs the OpenMP accelerator subcommittee
  - Pushing for official accelerator adoption in OpenMP 4.0
- The compiler will take care of
  - Determining the suitability of the parallel region for the visible code
    - Deep call chains may require user input
  - Splitting the code into accelerator and host portions
  - Handling any necessary details of moving data to and from the accelerator, with minimal assistance from the user
  - Setting up the actual execution of the parallel region on the accelerator
Programming for a Node with Accelerator

- Fortran, C, and C++ compilers
  - Compiler support with identification of parallel loops (through vectorization and autothreading) to take advantage of both the accelerator and multi-core X86
  - Workload distribution for splitting off work destined for the accelerator
  - OpenMP-based directives to drive compiler optimization
  - Advanced users can mix CUDA functions with compiler-generated accelerator code
  - Runtime library to allow for nested parallel regions to run on the accelerator

- Cray Optimization Explorer (COE) built upon a database containing a representation of the application
  - Scoping tool to help users port and optimize applications
  - Performance measurement and analysis tools for porting and optimization
    - Loop statistics
    - Advanced statistics gathering facility for CPU and GPU
  - Application tuning tool based on auto-tuning technology to help users optimize application kernels

- Scientific Libraries support
  - Auto-tuned libraries (using ATF infrastructure)
  - Exploit mixed precision in the algorithms (using IRT background)
Co-chairs
- James Beyer (Cray) and Eric Stotzer (TI)

The committee’s goal is to come up with a way to allow programmers to target accelerators (GPUs) with OpenMP type directives

OpenMP is a high-level programming model

CUDA and OpenCL are low-level programming models

Initial proposal submitted by Cray

Membership
- AMD*
- ANL
- BSC*
- CAPS*
- Cray*
- EPCC*
- IBM*
- Intel*
- PGI*
- Oracle
- Texas Instruments*
- TACC*

*active members
Whole program analysis scoping tool
- Investigate parallelizability of high level looping structures
  - The compiler performs an initial parallelization analysis to identify obvious inhibitors to parallelization
  - The user instructs the compiler to ignore various inhibitors if possible
  - The compiler performs an initial scoping analysis and presents the User with concerns with array usage
  - The user works with the environment to trace variables through the high level looping structure, works with the compiler to scope the variables in question

Navigation
- Ranked list of top timing consuming functions or loops
- Call tree with functions and/or loops
- Collapsed call tree including both functions and loops
- User can navigate from function to loop to line within source code
- Source code is annotated with line numbers, compiler optimization markers, loop iteration counters, performance statistics
- User can dive into function from call site
- User can select variable and find all references through program
Scientific Libraries

- Focus on investigation of use of Auto-Tuning Framework for Acceleration of Scientific Libraries
- LSMS Tuning for Application Readiness Review
  - ZBLOCK_LU has an internal tunable parameter
  - Changing the internal block size changes the performance notably and increases the time spent in ZGEMM

![Graph showing performance comparison of Auto-tuned ZGEMM, CUBLAS, and GOTO variants.](image-url)
In addition to the Cray Differentiated Programming Environment for GPUs, the following third party components are being considered to be integrated:

- **Compilers**
  - NVIDIA C and C++
  - PGI Fortran, C, and C++

- **Libraries**
  - CUDA Runtime support libraries
  - NVIDIA Thread Storage libraries
  - NVIDIA GPU-accelerated BLAS
  - NVIDIA GPU-accelerated FFT
  - NVIDIA GPU-accelerated LAPACK
  - MAGMA
  - PETSc
  - Trilinos

- **Tools**
  - Environment setup
    - Modules
  - Debuggers
    - NVIDIA debugger
    - TotalView
    - DDT
  - Performance Tools
    - CUDA Visual Profiler
    - OpenCL Visual Profiler

As new libraries routines are needed by important applications, those will be considered and prioritized.