Third Party Tools for Titan

Richard Graham, Oscar Hernandez, Christos Kartsaklis, Joshua Ladd, Jens Domke (Oak Ridge National Laboratory)

Jean-Charles Vasnier, Stephane Bihan, and Georges-Emmanuel Moulard (CAPS Enterprise)

Presented by: Joshua S. Ladd
Goals

• Provide OLCF-3 users with high productivity programming languages & compilers.

• Provide a programming environment with tools to support the porting of codes.

• Work with vendors to provide compiler, performance, and debugger capabilities needed to port applications with GPUs:
  – CAPS enterprise (OpenHMPP Directives)
  – The Portland Group (Accelerator Directives)
  – Cray (OpenMP for Accelerators)
  – NVIDIA
  – TU-Dresden (Vampir)
  – Allinea (DDT)

• Joint Standardization Efforts: OpenACC, OpenMP ARB
Improve Productivity and Portability

• The Directive based approach provides:
  – Incremental porting/development
  – Fast Prototyping
    • The programmer can quickly produce code that runs in the accelerator
  – Increases Productivity
    • Few code modifications to produce accelerated code
  – Retargetable to different architectures (CPU, GPUs, FPGAs)
  – Tools can assist the user generate the directives, debug them, and do performance analysis

• Leading technologies with accelerator directives:
  – OpenACC directives (Cray, PGI, CAPS, NVIDIA)
  – CAPS OpenHMPP directives
  – PGI accelerator directives
## Compilers Available for OLCF-3

<table>
<thead>
<tr>
<th>Compiler Vendor</th>
<th>C/C++</th>
<th>Fortran</th>
<th>CUDA C / OpenCL</th>
<th>CUDA Fortran</th>
<th>OpenHMPP</th>
<th>PGI Acc Dir</th>
<th>OpenACC</th>
<th>OpenMP CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PGI</td>
<td>X</td>
<td>X</td>
<td>P</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CAPS HMPP</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVIDIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pathscale</td>
<td>X</td>
<td>X</td>
<td></td>
<td>P</td>
<td>P</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>GNU</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

X = Supported  
P = In Progress

- Cray, CAPS, and NVIDIA are directly involved with the OLCF-3 Effort
Current Work

- We are currently working with Vendors to provide a set of tools that target the application needs for OLCF-3
- We are also building a tool environment to support the applications:
Programming Models

• System supports: C/C++/Fortran, OpenMP, MPI, SHMEM
• GPU accelerator directives: OpenACC and OpenHMPP directives
• We identified features needed for programmability, performance improvements or bugs.
  – C++ support [Feature]
  – Fortran module support [Feature]
  – Inlining support [Feature]
  – Need to allocate data directly in the GPU [Performance]
  – 3D scheduling support for thread blocks [Performance]
  – Support for libraries [Feature]
  – Codelet functions [Feature]
  – Worksharing between devices in nodes. [Feature]
HMPP Overview

- C/Fortran OpenHMPP and OpenACC directive-based programming model (API-based for C++)
  - Offload functions and regions onto accelerators
  - Manage CPU-GPU data movements
  - Finely tune loop optimizations and use of hardware specific features
  - Complementary to OpenMP and MPI

- A source-to-source HMPP compiler
  - Generates CUDA/OpenCL/MiC kernels
  - Works with standard compilers

- A runtime library
  - Dispatch computations on available GPUs
  - Scale to multi-GPU systems
HMPP Enhanced Features: Driven by OLCF Application Needs

• MADNESS:
  – API for C++ applications

• LAMMPS:
  – Multi-GPU and CPU data distribution

• LSMS:
  – Seamless integration of accelerated libraries
HMPP for C++ Applications

• Runtime API close to OpenCL with one directive to let HMPP generate the CUDA version of a function

```cpp
template<typename T>
void add_vector(int n, T * y, T * a, T * b)
{
    for(int i = 0 ; i < n ; i++)
        y[i] = a[i] + b[i];
}
```

// Instantiate the add_vector<float> template and // associate it to the codelet "add_vector_float"
#pragma hmppcg entrypoint as add_vector_float, target=CUDA
template void add_vector<float>(int n, float *y,
                             float *a, float *b);

• Templates not supported by OpenACC
• MADNESS – heavily dependent on templates
HMPP Data Distribution

- Directives for C and Fortran
  - Define a multi-GPU data distribution scheme and let HMPP execute over multiple devices

```c
float vin1[NB][SIZE][SIZE], vin2[NB][SIZE][SIZE], vout[NB][SIZE][SIZE];
...
#pragma hmpp parallel, device="i%2"
for( i=0;i<NB;i++) {
  //Allocate the mirrors for vin1, vin2 and vout
  #pragma hmpp allocate, data["vin1[i]", ...], size={size,size}
  //Transfer data to the GPU from the mirrors
  #pragma hmpp advancedload, data["vin1[i]","vin2[i]","vout[i]"
  ...
  #pragma hmpp sgemm callsite
  sgemm( vin1[i], vin2[i], vout[i] );
  ...
  //Get back the result
  #pragma hmpp delegatedstore, data["vout[i]"
}
```
HMPP Accelerated Library Integration

- Use accelerated libraries as an alternative to the original library
  - Keep one single source code with CPU and GPU libraries
  - Uses hmppalt directives with a proxy mechanism
- Compatible with CUBLAS, MAGMA, CULA libraries
- Access of data allocated by library with directives
LSMS case study

• Already existing GPU version using cuBLAS and CULA libraries

```fortran
!$hmppalt myproxy declare, name="zgemm", fallback=true
SUBROUTINE lsmszgemm(error,transa,...)
  ...
END SUBROUTINE lsmszgemm

!$hmppalt lsms declare, name="zgetrf", fallback=true
SUBROUTINE lsmszgetrf(error, m, n, a, lda, ipiv, info)
  ...
END SUBROUTINE lsmszgetrf

do iblk=nblk,2,-1
  ...
call cula_device_zgetrf(...) 
call cula_device_zgetrs(...) 
call cublas_zgemm(...) 
call cublas_zgemm(...) 
enddo

call cublas_zgemm(...) 
call cublas_get_matrix(...) 
```

Declare proxy interfaces

Call HMPP alternative library while keep original call in place
Vampir Toolset

• VampirTrace
  – Application instrumentation
  – Pre-run phase
  – Via compiler wrapper, library wrapper and/or third-party software
  – Measurement
  – Event collection (functions calls, MPI, OpenMP, performance counter, memory usage, I/O, GPU)
  – Timer synchronization
  – Filtering and grouping of events

• Vampir (Client and Server)
  – Trace visualization software
  – Alternative and supplement to automatic analysis
  – Show dynamic run-time behavior graphically
  – Provide statistics and performance metrics
  – Interactive browsing, zooming, selecting capabilities
Vampir Performance Analysis Tools

• Custom improvements for the OLCF-3 system
• Focused on three main areas
  – Scaling the Vampir tools to higher processor counts
  – Integrating GPU support for a comprehensive analysis of heterogeneous systems
  – Additional usability enhancements
Vampir toolset challenges

• Support for GPU tracing
• Scaling up to a Titan-sized HPC-system
• Overcome I/O challenges related to the huge amount of data generated by traces as well as the number of tracing streams
• General enhancements in terms of usability and analysis possibilities
Vampir GPU support for Titan

• Successive CUDA support based on latest CUDA releases

• How CUDA fits into Vampir’s design
  – accelerator threads treated like OMP threads
  – CUDA memcpy treated like MPI communication

• Tracing of multiple CUDA kernels in one executable

• Tracing of asynchronous GPU events

• Tracing of performance counters on GPUs
Vampir Scalability Improvements

• Parallelization of VampirTrace tools: otfmerge, vtfilter, otfprofile
  – enable the processing of traces with >10k streams!!

• Enhanced output of otfprofile
  – Global message statistics
  – Time spent in MPI
  – Process clustering information

• Improved MPI behavior of VampirServer
  – Analyze traces with >10k analysis processes
  – Allows to analyze traces on the entire Titan system

• New displays
  – Performance radar (performance counter for all processes over time)
  – Highlight performance bottlenecks (related to counters)
I/O Challenge

- Online trace file compression (reduce i/o load)
  - Find pattern and record only one (aka rewind)
  - Find pattern irregularities: use marker to highlight pattern irregularities

- Visualization of those compressed traces via multiplexing the recorded pattern (only in the display, not in memory) to show the trace as a whole

- Use IOFSL to aggregate large number of streams in a small number of files (instead of saving each stream in one file) to reduce the load for the Lustre meta data server
S3D on Jaguar (200k+ cores)
Vampir Usability/Analysis Enhancements

- Vampir clients for all major OS platforms (Linux, Mac, Win)
- Automatic generation of filter files (based on previous runs) to trace only performance relevant parts
- Performance Radar
  - Derived counters
  - Highlighting of function anomalies based on user defined thresholds
- Enhanced filtering capabilities
  - functions, functions groups, processes inside of Vampir
- Compare View
  - Compare multiple traces directly, while showing them in one display
Vampir Usability/Analysis Enhancements

Performance Radar

Compare View
Allinea DDT Debugger

• Work with Allinea to improve the scalability of the DDT debugger

• Data Analysis
  – Parallel Watchpoints
  – Addition of “sparklines”
  – Scalable data analysis
  – Scalable breakpoints, stepping and program stack queries

• Major GPU enhancements
DDT Integration with Cray PE

• Support for Abnormal Process Termination (APT), allows to attach DDT to aborted process and review stack

```
Application 1110443 is crashing. ATP analysis proceeding...

Stack walkback for Rank 23 starting:
  _start@start.S:113
  __libc_start_main@libc-start.c:220
  main@atploop.c:48
  __kill@0x4b5be7
Stack walkback for Rank 23 done
Process died with signal 11: 'Segmentation fault'
View application merged backtrace tree file 'atpMergedBT.dot' with 'statview'
You may need to 'module load stat'.

atpFrontend: Waiting 5 minutes for debugger to attach...
```

• Multiple core file support using xt_setup_core_handler()

• Open MPI (Cray XK/alps) version support
DDT GPU Support

• Cray compiler support
  – Stepping into OpenACC regions
  – Setting breakpoints within OpenACC regions

• Support for HMPP directives
  – Stepping into HMPP codelets

• Multiwarp stepping in CUDA codes

• CUDA memory debugging capabilities
  – Identify memory leaks on the device
  – Visual display of memory allocated on host and device
DDT Debugging with HMPP directives
DDT and OpenACC Directives

```fortran
input(i) = i*4.0

result = -5

!$omp acc_region_loop private(t) firstprivate(a.var_int, var_double, var_complex)
DO i = 1, n
  t = input(i) * 4.0
  result(i) = t + a - b ! Net consequence, result(i) = t - 2
END DO

IF (var_bool) THEN
  var_double = var_int + var_real;
  var_complex = CMPLX(a,t)
END IF

! Characters/strings not supported on accelerators
! var string = var string // var char
```
DDT and VisIT

- Integration with VisIT (Available with VisIT 2.5 summer 2012)
  - Coupling advanced application generated data visualization capabilities with scalable debugging
Thank you for your attention!

• Questions and comments are most welcome