Exploring Programming Multi-GPUs using OpenMP and OpenACC-based Hybrid Model

Rengan Xu, Sunita Chandrasekaran
and Barbara Chapman

Department of Computer Science
University of Houston

PLC Workshop 2013
Outline

1 Motivation

2 Overview of OpenMP and OpenACC

3 Porting Applications on Multi-GPU

4 Proposed Directives for Multiple Devices

5 Conclusion and Future Work
Motivation

- GPUs have high compute capability in HPC, but programming these devices is a challenge

- Low-level models: CUDA, OpenCL
  - Language extension
  - Time-consuming to write and error-prone

- High-level models: OpenACC, PGI, HMPP
  - Directive based
  - Hiding low-level details from the programmer
  - Reduce learning curve and development time

- Multi-GPU support:
  - One node: OpenMP + OpenACC
  - Multiple nodes: MPI + OpenACC
Overview of OpenMP and OpenACC

- **OpenMP**
  - Directive-based model for shared memory system
  - Contains directives, runtime routines and environment variables
  - Fork-join model
  - Threads communicate via shared variables

- **OpenACC**
  - Standard for directive-based accelerator programming
  - Contains directives, runtime routines and environment variables
  - Three levels parallelism: gang, worker and vector
  - Handle memory traffic between the host and device
Porting Applications on Multi-GPU

Parallelization strategy

Figure: Multi-GPU Solution using Hybrid OpenMP and OpenACC
Porting Applications on Multi-GPU

Testbed

- OpenACC compiler: HMPP (renamed as CAPS now)
- GCC 4.4.7 as host compiler, -O3 optimization used

Table: Specification of experiment machine

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Intel Xeon x86_64</td>
</tr>
<tr>
<td>Cores</td>
<td>16</td>
</tr>
<tr>
<td>CPU frequency</td>
<td>2.27GHz</td>
</tr>
<tr>
<td>Main memory</td>
<td>32GB</td>
</tr>
<tr>
<td>GPU Model</td>
<td>Tesla C2075</td>
</tr>
<tr>
<td>GPU cores</td>
<td>448</td>
</tr>
<tr>
<td>GPU clock rate</td>
<td>1.15GHz</td>
</tr>
<tr>
<td>GPU global &amp; constant memory</td>
<td>5375MB &amp; 64K</td>
</tr>
<tr>
<td>Shared memory per block</td>
<td>48KB</td>
</tr>
</tbody>
</table>
S3D is a solver that performs direct numerical simulation of turbulent combustion.

The thermodynamics kernel is chosen for experiment.

Two kernels are independent, same input, different output.

In single GPU, two kernels share the input.

In multi-GPU version:
  - The input are duplicate
  - Set the device number with runtime routine
  - Use OpenMP sections to distribute workload
Porting Applications on Multi-GPU

Matrix Multiplication

- Distribute one large kernel to multi-GPU
- Use explicit OpenMP static loop scheduling
- Each partitioned segment is executed on one GPU
- Set device number based on the thread number
- Only copy necessary data into each GPU
  - Partial copy in OpenACC
- Handle shared and private variables in OpenMP and OpenACC
Porting Applications on Multi-GPU

2D Heat Equation

- Formula:

\[
\frac{\partial T}{\partial t} = \alpha \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right)
\]
Porting Applications on Multi-GPU

2D Heat Equation

- Different kernels have dependence
- Host threads communicate and exchange data via shared data
- Atomic or critical regions used to prevent data race
- Barrier needed for synchronization

Figure: Multi-GPU Implementation Strategy for 2D Heat Equation
Porting Applications on Multi-GPU

S3D Thermodynamics Kernel

- Speedup of S3D, MM and Heat Equation compared to OpenMP (8 threads).
#pragma acc multi_device [clause [[,] clause]...] new-line
  structured-block where clause is one of the following:
devices(scalar-integer-expression)
if(condition)
async[(scalar-integer-expression)]
copy(list)
copyin(list)
copyout(list)
create(list)
Conclusion and Future Work

Conclusions:
- It is feasible to program multi-GPU with OpenMP and OpenACC.
- Significant speedup can be achieved by using multi-GPU.
- Proposed new directive to support multiple devices.

Future Work:
- Implement proposed directive in OpenUH compiler.
- Evaluate the implementation performance with PGI compiler.