SnuCL and an MPI+OpenCL Implementation of HPL on Heterogeneous CPU/GPU Clusters*

Jaejin Lee
Center for Manycore Programming
Department of Computer Science and Engineering
Seoul National University
jlee@cse.snu.ac.kr
http://aces.snu.ac.kr

*This work was supported in part by grant 2009-0081569 (Creative ResearchInitiatives: Center for Manycore Programming) from the National Research Foundation of Korea. This work was also supported in part by the Ministry of Education, Science and Technology of Korea under the BK21 Project. ICT at Seoul National University provided research facilities for this study.
Goal

- To implement HPL using SnuCL
- This talk describes a preliminary step
  - Implementing HPL using MPI and OpenCL
    - Inter-node: MPI
    - Intra-node: OpenCL
  - Performance evaluation on the SnuCore heterogeneous cluster
OpenCL

- *Open Computing Language*

- A framework (parallel programming model) for *heterogeneous parallel computing*
  - A language, API, libraries, and a runtime system

- The specification of OpenCL 1.0 was released in late 2008
  - Now, OpenCL 1.2
OpenCL (contd.)

- From mobile devices to supercomputers
- Portable code across different architectures
  - CPUs, GPUs, Cell BE processors, etc.
  - Does not provide portable performance yet
- Based on ANSI/ISO C99 standard
- Supported by many vendors, such as Apple, AMD, ARM, IBM, Intel, NVIDIA, Samsung, etc.
OpenCL Limitations

- Current OpenCL implementations are targeting parallelism for multiple compute devices under a single OS instance
- Applications for a heterogeneous CPU/GPU cluster are typically written in MPI + OpenCL or MPI + CUDA
- Complicated, less portable, and hard to maintain
Illusion of a System Running a Single OS Instance

- If the programmer can write applications for heterogeneous CPU/GPU clusters using only OpenCL
- Easy to program and more portable
SnuCL

- An OpenCL framework
  - Freely available, open-source software developed at Seoul National University
  - http://aces.snu.ac.kr

- Platform layer + runtime + kernel compiler
SnuCL (contd.)

- Naturally extends the original OpenCL semantics to the heterogeneous cluster environment
- Provides an illusion of a single operating system instance
- With SnuCL, an OpenCL application written for a single heterogeneous system runs on a heterogeneous cluster without any modification
How to Achieve the Illusion?

- SnuCL runtime provides the illusion
- Source-to-source kernel restructuring techniques
  - OpenCL C to C (for CPUs)
  - OpenCL C to CUDA (for NVIDIA GPUs)
- Buffer management techniques
  - Efficient node to node data transfer
  - Consistency management
## The Effect of Using SnuCL

- Copy buffers between different nodes in the cluster environment (Buffer A → Buffer B)
- Using only OpenCL API functions

### Previous approach (Mixture of MPI and OpenCL)

- `MPI_Init(..);`
- `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`
- `cl_mem bufferA = clCreateBuffer(...);`
- `cl_mem bufferB = clCreateBuffer(...);`
- `void *temp = malloc(...);`
- `if (rank == SRC_DEV) {
   clEnqueueReadBuffer(cq, bufferA, ..., temp, ...);
   MPI_Send(temp, ..., DST_DEV, ...);
} else if (rank == DST_DEV) {
   MPI_Recv(temp, ..., SRC_DEV, ...);
   clEnqueueWriteBuffer(cq, bufferB, ..., temp, ...);
} ...`
- `MPI_Finalize();`

### SnuCL (OpenCL only)

- `cl_mem bufferA = clCreateBuffer(...);`
- `cl_mem bufferB = clCreateBuffer(...);`
- `...`
- `clEnqueueCopyBuffer(cq, bufferA, bufferB, ...);`
- `...`
SnuCL Extensions to OpenCL

- SnuCL has extensions to OpenCL for copying buffers
- Buffer-copy memory commands are often inefficient in the cluster environment depending on the access pattern
- Similar to MPI collective communication operations

<table>
<thead>
<tr>
<th>SnuCL</th>
<th>MPI Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>clEnqueueAlltoAllBuffer</td>
<td>MPI_Alltoall</td>
</tr>
<tr>
<td>clEnqueueBroadcastBuffer</td>
<td>MPI_Bcast</td>
</tr>
<tr>
<td>clEnqueueScatterBuffer</td>
<td>MPI_Scatter</td>
</tr>
<tr>
<td>clEnqueueGatherBuffer</td>
<td>MPI_Gather</td>
</tr>
<tr>
<td>clEnqueueAllGatherBuffer</td>
<td>MPI_Allgather</td>
</tr>
<tr>
<td>clEnqueueReduceBuffer</td>
<td>MPI_Reduce</td>
</tr>
<tr>
<td>clEnqueueAllReduceBuffer</td>
<td>MPI_Allreduce</td>
</tr>
<tr>
<td>clEnqueueReduceScatterBuffer</td>
<td>MPI_Reduce_scatter</td>
</tr>
<tr>
<td>clEnqueueScanBuffer</td>
<td>MPI_Scan</td>
</tr>
</tbody>
</table>
Matrix Multiplication Example

- Five different implementations
  - For a single GPU under a single OS instance
    - OpenCL
  - For multiple GPUs under a single OS instance
    - OpenCL
  - For multiple GPUs in the cluster using,
    - OpenCL + MPI
    - SnuCL
    - SnuCL with collective communication extensions

*The same code!*
SNU NPB Suite

- A set of the NAS Parallel Benchmarks (NPB 3.3) implemented in C, OpenMP C, and OpenCL [IISWC ’11]
  - NPB-SER-C: a serial C version of the NPB code
  - NPB-OMP-C: an OpenMP C version of the NPB code
  - NPB-OCL: an OpenCL version of the NPB code for a single device
  - NPB-OCL-MD: an OpenCL version of the NPB code for multiple OpenCL compute devices
- Source code is publicly available
  - [http://aces.snu.ac.kr](http://aces.snu.ac.kr)
High Performance LINPACK

- Solving a dense linear system of order \( n \)
  - \( n \times (n+1) \) coefficient matrix
- Using the blocked LU decomposition algorithm
  - The matrix is logically divided into blocks with a size of \( n_b \times n_b \)
  - Each iteration performs the following three steps
    - Factorization: the LU factorization is performed on \( L \)
    - Swap: Rows in \( U \) and \( A \) are swapped
    - Update: \( U \) and \( A \) are updated using \( L \)
HPL on Homogeneous Clusters

- Nodes are arranged as a two-dimensional grid
- The blocks of the matrix are cyclically distributed across the nodes
- Each iteration is performed on the multiple nodes
- Communication is required to
  - Find the pivot rows in the factorization phase
  - Broadcast L after the factorization phase
  - Swapping rows in U and A in the swap phase
HPL on Heterogeneous Clusters

- Most of the floating-point operations in HPL are from the update phase
  - Using DGEMM and DTRSM routines in BLAS
  - E.g., with n = 200,000 and nb = 2,000
    - Factorization: 0.5%, DTRSM: 0.5%, DGEMM: 99%
- Perform DGEMM and DTRSM in the update phase with GPUs
- Overlap the computation of GPUs with the computation/communication of CPUs
Exploiting Multiple GPUs

- Perform both DGEMM and DTRSM on multiple GPUs
  - In the update phase,
    - U is updated with X by solving the equation $L_1X = U$ using DTRSM
    - A is updated with the result of $A - L_2U$ using DGEMM
- Divide U, $L_2$, and A into submatrices and distribute them across GPUs
- Optimize the CPU side
  - The performance gap between CPUs and GPUs is big
  - CPUs become a performance bottleneck
Load Balancing Between GPUs

- Dynamically assigning the columns of blocks in U and A to the GPUs
  - U_i in DTRSM
  - A_{1i}, A_{2i}, A_{3i}, ..., A_{mi} in DGEMM
  - U_i can be reused in DGEMM
    - Minimizing the data transfer overhead between CPUs and GPUs
- If there is no column left to be scheduled, a GPU may steal some blocks from another GPU
Overlapping Computation/Communication on the CPU Side

- Overlapping the factorization phase (computation) in \((i+1)\text{th}\) iteration and the swap phase (communication) in the \(i\text{th}\) iteration.
Distributing the Factorization Phase

- If HPL is executed on a $P \times Q$ grid of the nodes, only $P$ nodes cooperate for the factorization phase
  - Make idle nodes to contribute to the factorization phase
    - Distributing some blocks of $L$ to the idle nodes
    - All nodes can cooperate for the factorization phase
Adaptive Execution

- If the CPUs become a performance bottleneck during execution, the number of GPUs assigned to the update phase is reduced
  - Reducing the number of threads assigned to the GPUs
  - Resulting in increased memory bandwidth
### SnuCore Specification

- **16 compute nodes**
- **Each node contains**
  - 2 × 12-core 2.1GHz AMD Opteron 6172
  - 3 × AMD Radeon HD 6990 graphics cards
    - 6 GPUs
    - Theoretical peak per GPU: 685 GFLOPS (double-precision)
  - Main memory: 128GB
  - Motherboard: Tyan s8232
  - 2 × 1.5 KW power supplies
  - Single-port Mellanox InfiniBand HCA
  - Mellanox InfiniBand QDR switch
  - Water-cooling system for GPUs
Performance of HPL in MPI+OpenCL

- Porting HPL to MPI+OpenCL
  - Followed the rules for Top500 and Green500
- 991 Gflops per node (total 15.9 Tflops)
  - The best per-node performance compared to those in Top500
- 871 Mflops/Watt
  - In between top 15 and 16 in Green500
- 1/12 ~ 1/5 in price/performance ratio compared to those in Top500
Conclusions and Future Work

- Software optimization techniques are very important to achieve good performance and low power consumption for heterogeneous clusters
  - Many GPUs (> 2) per node is fine with optimizations on the CPU side
  - Better performance with less nodes and less power consumption
- HPL will be ported to SnuCL for heterogeneous CPU/GPU clusters
  - Using OpenCL only
- SnuCL for OpenCL 1.2 will be soon publicly available
  - Fully support OpenCL version 1.2
  - 100% passing conformance test cases from the Khronos group
## References


**References (contd.)**


Contributors

Gangwon Jo
Jeongho Nah
Jungwon Kim
Sangmin Seo
Jun Lee
Jaejin Lee

SnuCL and SNU-NPB are publicly available at http://aces.snu.ac.kr

Thanks!