
Takashi Shimokawabe (Tokyo Institute of Technology)

The weather prediction code demands large computational performance to achieve fast and high-resolution simulations. Skillful programming techniques are required for obtaining good parallel efficiency on GPU supercomputers. Our framework-based weather prediction code ASUCA has achieved good scalability with hiding complicated implementation and optimizations required for distributed GPUs, contributing to increasing the maintainability; ASUCA is a next-generation high-resolution meso-scale atmospheric model being developed by the Japan Meteorological Agency. Our framework automatically translates user-written stencil functions that update grid points and generates both GPU and CPU codes. User-written codes are paralleled by MPI with intra-node GPU peer-to-peer direct access. These codes can easily utilize optimizations such as overlapping technique to hide communication overhead by computation.

1 Introduction

Numerical weather prediction is one of the major applications in high-performance computing and is accelerated on GPU supercomputers. Obtaining high-performance using thousands of GPUs often needs skillful programming. The Japan Meteorological Agency is developing a next-generation high-resolution meso-scale weather prediction code ASUCA. We are implementing it on a multi-GPU platform by using a high-productivity framework. This poster presents our proposed framework and its performance evaluation.

2 Overview of Framework

- The proposed framework is designed for stencil applications with explicit time integration running on regular structured grids.
- The framework is intended to execute user programs on NVIDIA's GPUs and CPU.
- The framework is written in the C++ language and CUDA and can be used in the user code developed in the C++ language.
- Improving portability of both framework and user code and cooperation with the existing codes.
- The framework allows us to write multi-GPU code without considering handling multiple GPUs on a single process.
- To perform stencil computations on grids, the programmer only defines C++ functions that update a grid point, which is applied to entire grids by the framework.

3 Framework Implementation

Structure of Framework

- The framework supports multiple GPU computing.
- Optimized parallelization: Intra-node parallelization: mpi library
- Inter-node parallelization: OpenMP
- DomainGroup domain_group(rank, &manager);
- User-written main code, including memory allocation and time integration loop.

Stencil Computation on Grids

In order to execute stencil computation on grids, the programmer must describe functions that update a grid point. The framework provides C++ classes that apply user-written functions to grids. The user-written functions are executed on grids sequentially for CPU and in parallel for GPU using CUDA's global kernel functions.

GPU-GPU communication

- 2 sockets of the Intel CPU Xeon X5670 (Westmere-EP) 2.93 GHz 6-core
- 24 sockets of the NVIDIA Tesla K20X GPUs
- 128 GB of DDR memory
- PCI Express bus 2.0 × 16 (8 GB/s)
- Total 4224 NVIDIA Tesla K20X GPUs
- 24 nodes of the Intel CPU Xeon X5670 (Westmere-EP) 2.93 GHz 6-core
- 2 sockets of the NVIDIA Tesla K20X GPUs
- 128 GB of DDR memory
- PCI Express bus 2.0 × 16 (8 GB/s)
- Total 24 NVIDIA Tesla K20X GPUs

4 Programming Model

Parallelizing User Code

- User-written main code is executed in all OpenMP threads created by an OpenMP parallel directive.

5 Performance results

<table>
<thead>
<tr>
<th>TSUBAME 2.5 supercomputer at the Tokyo Institute of Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total 4224 NVIDIA Tesla K20X GPUs</td>
</tr>
<tr>
<td>Each node of TSUBAME 2.5</td>
</tr>
<tr>
<td>3 Tesla K20X GPUs attached to the PCI Express bus 2.0 × 16 (8 GB/s)</td>
</tr>
<tr>
<td>2 sockets of the Intel CPU Xeon X5670 (Westmere-EP) 2.93 GHz 6-core</td>
</tr>
<tr>
<td>2 QDR InfiniBand</td>
</tr>
</tbody>
</table>

Reference