# An OpenACC-Based Unified Programming Model for Multi-accelerator Systems

Jungwon Kim
Oak Ridge National Laboratory, USA
kimj@ornl.gov

Seyong Lee
Oak Ridge National Laboratory, USA
lees2@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory, USA
Georgia Institute of Technology, USA
vetter@computer.org

## Abstract

This paper proposes a novel SPMD programming model of OpenACC. Our model integrates the different granularities of parallelism from vector-level parallelism to node-level parallelism into a single, unified model based on OpenACC. It allows programmers to write programs for multiple accelerators using a uniform programming model whether they are in shared or distributed memory systems. We implement a prototype of our model and evaluate its performance with a GPU-based supercomputer using three benchmark applications.

***Categories and Subject Descriptors*** D.1.3 [*Programming Techniques*]: Concurrent Programming; D.3.4 [*Programming Languages*]: Processors – Code generation, Compilers, Optimization, Run-time environments

***Keywords*** OpenACC, Programming models, Heterogeneous computing, Accelerators

## 1. Introduction

Directive-based, high-level accelerator programming models such as OpenACC and OpenMP 4.0 have been gaining considerable attention as a powerful way to easily harness the computing power of the accelerators such as NVIDIA/AMD GPUs and Intel Xeon Phi. They allow programmers to use compiler directives to identify which parts of the program to be offloaded to an accelerator. Due to their code readability, maintainability, usability, and portability, directive-based programming models are increasingly being considered as an alternative to lower-level accelerator programming models such as CUDA and OpenCL.

Multi-accelerator systems are being increasingly used in high-performance computing to solve bigger problems within an acceptable time frame. The system can be a single-node system with multiple accelerators, or a cluster that contains multiple nodes with one or more accelerators per node. In order to scale accelerator applications across multiple accelerators, programmers usually have to resort to a combination of different programming models: OpenACC/OpenMP 4.0 for accelerators, OpenMP for multithreading, and MPI for inter-node communication. It adds the complexity of programming, and leads to lower productivity.

In this paper, we propose an OpenACC-based unified programming model for multiple-accelerator systems. It is a SPMD programming model of OpenACC. It expresses several levels of parallelism, from vector-level parallelism to node-level parallelism. The major contributions of this paper are the following:

- Our model extends the OpenACC execution model to the distributed heterogeneous systems. It provides a uniform way for programmers to program heterogeneous systems with multiple accelerators, in both single and multi-node configurations.
- Our runtime API provides a set of uniform synchronous/asynchronous communication routines to transfer data between host-to-host, host-to-accelerator, accelerator-to-host, and accelerator-to-accelerator either within a node or across nodes.
- The tight integration of message passing programming, multithreading programming and accelerator programming in our model allows various architecture-specific optimizations, transparent to users, such as 1) seamless streamlining of asynchronous intra-node/inter-node communication between accelerators, 2) data sharing and P2P direct transfers between program instances in the same node, and 3) dynamic mapping of logical tasks onto the hardware units.

## 2. Programming Model

Figure 1 illustrates the overview of our framework. The framework consists of two parts, a compiler and a runtime. Figure 2 shows the Jacobi code using our programming model. The programmer compiles the source codes using our compiler. The compiler is a source-to-source translator. It takes source codes and generates the host codes and the accelerator-specific kernel codes such as CUDA and OpenCL kernels. Then, the underlying backend compiler builds an output executable with CUDA/OpenCL, OpenMP, MPI, and our runtime libraries.

The executable runs in SPMD fashion. When a user launches the executable in a multi-accelerator system such as a heterogeneous cluster, a number of program instances, called tasks, execute in parallel on the cluster. Our runtime on each node creates the same number of lightweight user-level threads, called task threads, as the number of available accelerators in the local node. The runtime assigns a distinct accelerator to each task thread. Each of these task threads executes the same main function in the executable program using the assigned accelerator simultaneously in parallel. They follow different execution paths to work on different data using available language constructs.

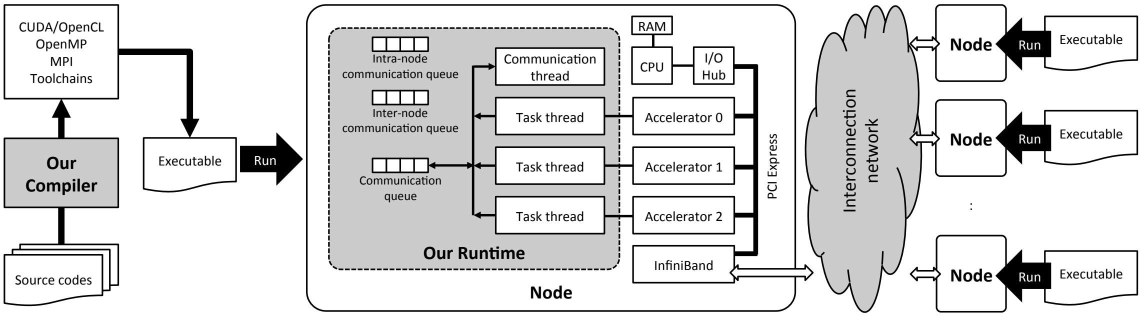

**Figure 1.** Our framework overview.

```
 1: double (*xlocal)[MAXN], (*xnew)[MAXN];
 2: double diffnorm, gdiffnorm;
 3: #pragma acc data localshared(temp)
 4: double temp[MAXN];
 5: ...
 6: void main() {
 7: int myid = acc_get_task_num();
 8: int ntasks = acc_get_num_tasks();
 9: size_t size = ((MAXN / ntasks) + 2) * MAXN * sizeof(double);
10: xlocal = (double(*)[MAXN]) malloc(size);
11: xnew = (double(*)[MAXN]) malloc(size);
12: ...
13: #pragma acc data create(xnew[0:(MAXN/ntasks)+2][0:MAXN]) \
14:                  copyin(xlocal[0:(MAXN/ntasks)+2][0:MAXN])
15: do {
16: if (myid < ntasks - 1)
17:   acc_mem_send(myid + 1, acc_deviceptr(xlocal[MAXN / ntasks]),
18:                MAXN * sizeof(double), 0);
19: if (myid > 0)
20:   acc_mem_recv(myid - 1, acc_deviceptr(xlocal[0]),
21:                MAXN * sizeof(double), 0);
22: ...
23: #pragma acc kernels loop independent reduction(+:diffnorm)
24: for (...) { ... }
25: acc_mem_allreduce(&diffnorm, &gdiffnorm, 1, acc_double, acc_sum);
26: } while (gdiffnorm > 1.0e-2)
27: }
```

**Figure 2.** Jacobi code using our programming model.

Each task has a unique ID assigned by the runtime throughout the whole system. The unique ID for each task and the total number of tasks in the whole system can be retrieved by `acc_get_task_num` and `acc_get_num_tasks` runtime API functions, respectively (line 7 and 8 in Figure 2).

Our model preserves the semantics of the standard OpenACC directives and runtime APIs, but it extends the OpenACC memory model by introducing logically distributed task address space; by default, all data (both global and local variables) in a program are private to each task. However, global variables annotated with `#pragma acc data localshared` directive (line 3) can be shared between tasks in the same node. This logically distributed task address space will be dynamically mapped to the underlying physical system by the runtime.

Our runtime API provides a set of uniform routines to transfer data between tasks in the system (e.g., `acc_mem_send` and `acc_mem_recv` in Figure 2), where data can reside in either host memory or device memory. The tasks associated with a data transfer can be in the same node or different nodes.

The communication thread in each node runtime manages all intra-node/inter-node communications related to the node. It seamlessly integrates the low-level accelerator APIs and inter-node communication APIs. This enables streamlining of asynchronous inter-node communication between accelerators, resulting in both high performance by full overlapping of computation and communication, and higher programming productivity. Also, it provides efficient data transfer such as P2P direct transfer between accelerators in the same node.

## 3. Preliminary Evaluation

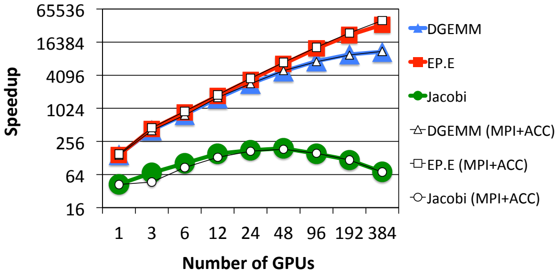

**Figure 3.** Speedup over a single CPU core.

We have built a prototype framework of our model using an open-source research compiler called OpenARC[1]. We have evaluated its performance with a GPU-based supercomputer using three benchmark applications. The cluster consists of 264 nodes. Each node has three NVIDIA M2090 GPUs and a Mellanox FDR InfiniBand interconnect.

Figure 3 shows the speedup of our framework over a single CPU core for each application. The y-axis has a logarithmic scale. We compare our applications with the MPI+OpenACC versions (MPI+ACC). Our applications show competitive performance with MPI+ACC versions. Especially, Jacobi shows better performance than Jacobi (MPI+ACC) when it runs with a small number of GPUs. This performance gain comes from the P2P direct transfer in NVIDIA GPUDirect between GPUs in the same node in our framework.

## Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes.

## References

[1] S. Lee and J. S. Vetter. OpenARC: Open Accelerator Research Compiler for Directive-based, Efficient Heterogeneous Computing. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 115–120, 2014.