Background and Goal

- Accelerated parallel computers (e.g., GPU clusters) emerged.
- MPI+CUDA style programming is difficult for programmers.
- Direct communication between accelerators would be important to improve strong scaling.
  - NVIDIA GPU Direct
  - NVIDIA GPUs

- MPI+CUDA style programming is difficult for programmers.
  - NVIDIA GPU Direct
  - NVIDIA GPUs

Direct communication between accelerators would be important to improve strong scaling.

Our Approach: New language for accelerated parallel computers by combining XcalableMP and OpenACC

Another directive-based language extension for heterogeneous CPU/ACC systems.

- Offloading programs from a host CPU to an attached accelerator device (ACC)
- Portability across operating systems and various types of CPUs and ACCs.

XcalableMP = XcalableMP + OpenACC + Novel Extensions

Basic Concept

- XcalableMP for distributed-memory parallelism
- OpenACC for offloading works to ACC
- XACC extensions for:
  - Multiple ACCs
  - Direct communication between ACCs

Target Architecture

- Homogeneous nodes composed of one CPU and one or more ACCs
- Two kinds of interconnect: between CPUs and between ACCs

XACC Extensions

- Data and work mapping onto multiple ACCs (two-level distribution)
- 1. Distribution among nodes by XMP directives
- 2. Distribution among ACCs within a node by the novel layout and on_device clauses.
- Direct communication between ACCs
  - Communication directives from XMP accept data on the ACC memory.

Array/Work

- Array directives declare an XACC device that may be a set of ACCs.
- on_device clause specifies the target ACC of OpenACC directives.
- acc clause specifies which instance of the data (on CPU or ACCs) is to be communicated.
- layout clause specifies data/work mapping onto an XACC device.
- shadow clause specifies the stencil area of a distribute array.

XcalableACC = XcalableMP + OpenACC + Novel Extensions

Omni XcalableACC

- being developed as an additional function of the Omni XcalableMP compiler
- Translating an XACC program to an MPI+OpenACC program.
- primary target: HA-PACS/TCA at CCS, U. Tsukuba

Preliminary Evaluation

Environment

- HA-PACS/TCA, based on the TCA architecture
- the "PEACH2 (PCI-Express Adaptive Communication Hub ver.2)" chip enables direct communication between accelerators
- Omni as the backend OpenACC compiler

Evaluation

- The Himeno benchmark (a very typical stencil code) parallelized with XACC
  - size = 128x128x256
  - Up to 2.7 times faster than MPI+OpenACC

Conclusion

- XACC could support both high performance and high productivity.

Configuration of an HA-PACS/TCA node

- Intel Xeon-E5 2680v2 2.8 GHz x 2 Socket
- GDDR5 6GB/GPU
- Omni XcalableACC
- Translator
- Frontend
- Compiler
- Runtime
- MPI library
- Configurator
- Frontend: CPU
- Backend: GPU
- HA-PACS/TCA node
- 0
- 1
- 2
- 4
- 8
- 16
- Number of Nodes
- Performance (GFLOPS)
- 0
- 80
- 160
- 240
- 320
- 400
- 224 GFlops/CPU
- 1.81 GFlops/ACC
- 17.20 GFlops/ACC (TCA)
- 12.20 GFlops/ACC (TCA)