発表概要

GPGPUフレームワークMESI-CUDAのマルチGPU環境への対応

山本 怜[1,a)]　大野 和彦[1]

2015年8月5日発表

GPGPU の分野において，複数の GPU を搭載したマルチ GPU 環境を用いてより高い計算性能を実現する試みがなされている．現在主流の開発環境である CUDA はマルチ GPU に対応しているが，個々の GPU を明示的に操作する必要があり，プログラムの記述が煩雑になる．さらに，1 台のホスト上に搭載できる GPU の個数が限られているため，より多くの GPU を利用する大規模な環境は分散型マルチ GPU 環境となる．この場合，同一ホスト上の GPU か否かで通信オーバヘッドを考慮するなど，プログラムの記述やチューニングはさらに難易度が高くなる．我々は CUDA よりプログラム記述が容易なフレームワーク MESI-CUDA を開発している．MESI-CUDA は CPU・GPU コアが単一の仮想共有メモリにアクセスするプログラミングモデルを採用している．処理系はホストメモリ・デバイスメモリの確保・解放やデータ転送などのコードを自動生成することで，このモデルで記述されたプログラムを CUDA コードに変換する．本提案では，このモデルをそのままマルチ GPU 環境に拡張することで，低レベルな各 GPU への操作の記述を不要にする．また，論理的なスレッド生成方式を導入し，ユーザが生成を指示したスレッド群は実行時スケジューラにより適切な GPU へ自動的に割り当てる．コンパイラは各スレッドのデータアクセス範囲などを静的解析し，実行時スケジューラはデータ転送量の最小化などの自動最適化を実現する．

A GPGPU Framework MESI-CUDA for Multi-GPU Environment

Rei Yamamoto[1,a)]　Kazuhiko Ohno[1]

Presented: August 5, 2015

Recently, GPGPU is used for high performance computing. Although multi-GPU is expected as the platform for higher performance, current standard programming environment CUDA requires explicit operation on the individual GPUs. Furthermore, hand-tuning is necessary to use all GPUs efficiently. Because only a few GPUs can be physically installed on a single host, a large-scale multi-GPU environment will be a cluster of hosts connected by the network. On such a environment, the user must specify inter/intra-host communication considering the difference of the overhead. Thus the programming and tuning will be more difficult. We are developing a new programming framework named MESI-CUDA which enables easier GPU programming than CUDA. In this paper, we propose an extension of MESI-CUDA to support multi-GPU environments. Current MESI-CUDA provides a simple programming model that every CPU/GPU cores accesses a single virtual shared memory. The compiler translates a MESI-CUDA program to CUDA program automatically generating memory management and data transfer code. We extend this model to support multi-GPU environments, hiding the individual GPUs from the user and eliminating low-level specifications. We introduce a new logical thread creation scheme; the user creates GPU threads without specifying the target GPU and the runtime thread scheduler automatically invokes physical threads on the available GPUs. The MESI-CUDA compiler makes static analysis to obtain the data access range of each thread. Using the analysis result, the runtime scheduler performs automatic optimization such as minimizing data transfer.

[1] 三重大学大学院工学研究科
Graduate School of Engineering, Mie University
[a)] yamamoto@cs.info.mie-u.ac.jp