講義「並列プログラミング」の紹介

伊野 文彦
大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻

1. はじめに

2000 年初頭までの CPU（Central Processing Unit）の処理速度（性能）を回顧したい。その時期は、新しい CPU が開発されるたびに動作周波数が向上していたため、CPU を更新すれば、プログラムはより短い時間で実行を終えることができた。

しかし、2004 年に Intel Pentium 4 プロセッサの動作周波数が 3.8 GHz に達して以来、電力消費の制約が原因で、動作周波数の向上は頭打ちである。動作周波数を向上させる替わりに、CPU は複数のコアを搭載するマルチコア化により性能を高めている。したがって、プログラムの性能を向上するためには、そのアルゴリズムの並列化が必須である[1]。

また、並列プログラミングが必要とされる領域は、この 10 年で飛躍的に拡大している。例えば、グラフィクス処理のためのハードウェア GPU（Graphics Processing Unit）は、チップ内に 2688 個ものコアを持ち、TOP500 リストにおいて世界最速（2012 年 11 月現在）のスーパーコンピュータ Titan に汎用アクセラレータとして採用されている。もはや携帯電話や自動車向けの CPU でさえマルチコアである。

このような状況に応えるために、大阪大学大学院情報科学研究科では大学院生を対象として，講義「並列プログラミング」および「並列アルゴリズム」を隔年で開講している。本稿では、前者の目標、内容、およびその演習環境について紹介する。

2. 講義「並列プログラミング」

並列プログラミングの醍醐味は、複数のプロセッサを用いて 1 つの計算を高速化することである。高速化を実現するために、数多の並列計算機やプログラミング言語が開発されてきた。これらのうち、本講義では GPU クラスタを対象として（図 1）、OpenMP[2]および MPI（Message Passing Interface）[3]を用いてそれぞれノード内およびノード間並列処理を実現し、さらに CUDA（Compute Unified Device Architecture）[4]を用いてノード内処理を加速できることを目指している。

並列プログラムを単に記述できるだけでなく、逐次プログラムに対して高い速度向上率を達成できることが重要である。したがって、逐次プログラムのチューニング技術、その性能ボトルネックの分析、ならびに並列アーキテクチャについても解説している。以降では、各々の内容について概略を述べる。

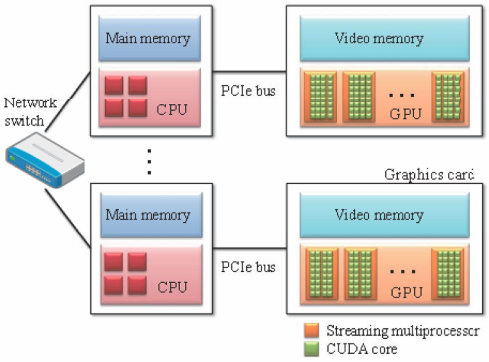

図 1 ：GPU クラスタの概要

2.1 並列計算機の概論

並列プログラムでは、対象の並列アーキテクチャを意識した記述が求められる。例えば、メモリアーキテクチャ（分散メモリや共有メモリ）は、並列プログラムを記述する際の大前提である。また、高い性能を達成するためには、コア間の接続形態やその性能特性について理解しておく必要がある。これらを考慮しない並列プログラムが逐次プログラムよりも低速になることは往々にしてある。メモリ一貫性、キャッシュ一貫性、ネットワークのトポロジ、2 分割帯域幅などについて解説している。

2.2 並列アルゴリズムの入門

OpenMP であれ CUDA であれ、プログラムの記述

スタイルは違えど、並列化の本質的な概念は共通である。まず、プログラム内に隠れている性能ボトルネックを特定し、その並列性を見出す。例えば、データ並列性、タスク並列性およびパイプライン並列性などが知られている。この際、アムダールの法則に従い、利用できるコア数に対して速度向上率の上限が十分に大きいことを確認すべきである。

次に、その並列性を効率よく使うために、各コアに対する処理割当やデータ配置を決める必要がある。処理割当はデータ配置と互いに強く依存しているため、この部分がもっとも難しい。並列化の阻害要因であるデータ依存を除去するためのデータ複製や owner computes ルールなどについて学ぶ。

計算量の観点から並列アルゴリズムを分析することも大切である。最適な逐次アルゴリズムに対して、余分なコアや時間を消費しているか否かを調べるためのコスト・ワーク最適の概念やアルゴリズムカスケーディング技術について学ぶ。

2.3 OpenMP と MPI による並列プログラミング

アルゴリズムの並列化が終われば、あとはそれを並列プログラムとして記述すればよい。本講義では、共有メモリ型並列計算機および分散メモリ型並列計算機向けにそれぞれ標準化されている OpenMP および MPI について解説している。

OpenMP では、コンパイラへの指示文を逐次プログラムに挿入するだけで fork-join 実行モデルに基づく並列プログラムを記述できる。メモリ空間を共有することに起因するクリティカルセクションやアトミック操作の概念を学ぶ。

一方、MPI ではメッセージ交換モデルに基づく並列プログラムを記述できる。特に、単一ノード内に格納できない大規模データを扱うためのデータ分割の概念を学ぶ。また、ノンブロッキング通信による通信と計算のオーバラップなど、通信に固有のチューニング技術についても解説している。

2.4 CUDA による GPU 計算

GPU を汎用アクセラレータとして使うための統合開発環境 CUDA について解説している。階層化されたアーキテクチャ、それを最大限に活用できるスケーラブルなプログラミングモデル、および高い実効メモリ帯域幅を引き出せるメモリ参照の最適化技術などについて学ぶ。また、本来の用途であるグラフィクス処理についても簡単に述べている。

2.5 演習環境について

並列プログラムの実行環境として GPU クラスタを提供している。受講生は SSH 接続できる PC のみを用意すればよい。GPU クラスタは 16 ノードで構成されていて、これらは GbE スイッチおよび Infiniband スイッチで接続されている。各ノードは 4 コアの Xeon プロセッサ、24GB の主記憶、GPU として Tesla C1060 を装備する。OS として Linux ディストリビューションの 1 つである ROCKS を採用していて、クラスタへのジョブはバッチ実行される。

3. おわりに

本稿では、大阪大学大学院情報科学研究科の講義「並列プログラミング」について紹介した。将来の Exa FLOPS（Floating-point Operations Per Second）時代に向けて並列処理のコミュニティが解決すべき課題は多い。例えば、OpenMP、MPI および CUDA を組み合わせたプログラム記述は大きな労力を必要とする。特に、新しい並列アーキテクチャが開発されるたびにプログラムの書き直しが必要となる現状を打破できる抽象化が望まれる。

参考文献

(1) H. Sutter, Dr. Dobb's J., **30**, (2005).

(2) OpenMP Architecture Review Board, OpenMP Application Programming Interface, Version 3.1, (2011).

(3) Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Version 3.0, (2012).

(4) NVIDIA Corporation, CUDA C Programming Guide, PG-02829-001_v5.0, (2012).