B-027

CPU と GPU の協調による組合せ最適化アルゴリズムの並列実装に関する研究
A Study on Parallel Impremention by Cooperation between CPU and GPU for Combinatrial Optimization Algorithms

渡部 晴人† 加藤 聡†
Haruto Watanabe Satoru Katoru

1. 研究の背景と目的

組合せ最適化は離散的な集合の中から最適解を探索する最適化手法の一つであり，集合の全ての組合せの数が入力の数に合わせて指数関数的に増えていく NP 困難である問題を解くために用いられる．組合せ最適化の歴史は長く，とくに巡回セールスマン問題では 2004 年にスウェーデン 24978 都市問題[1]の最適解が求められている．

これらの問題を解く際にはアルゴリズムの最適化によって不要な計算を省いても膨大な計算量が必要になるために大規模な並列計算機が用いられる．CPU の性能は 18～24 か月ごとに 2 倍になるというムーアの法則があり，より大きな問題を解く際には計算量の増加が CPU の性能向上ペースを大きく上回る．そのため現実的な計算時間で解くにはより大規模な並列計算機を構成する必要があるが，コストが高くなり現実的でない．

CPU に対し，グラフィックス処理ユニットである GPU の性能向上が近年目覚ましく，一般の CPU の 1/2～1/3 のクロック周波数でありながら数十倍の浮動小数点演算性能を持つようになっている．そのためコストパフォーマンスが良く，近年の HPC マシンにも搭載されるようになっている．しかし，拡張ボードのため CPU と比べて入出力のメモリ転送速度が遅く，データ並列処理に最適化された SIMD 型アーキテクチャのため，タスク並列処理が重要となる問題には適用できない． さらに，近年の HPC マシンでは CPU に加えて高速な GPU が多数搭載される例が増えており，ヘテロジニアスコンピューティングによるプログラムの最適化が必要になる．

そこで本研究では OpenCL[2]を用い，CPU と GPU 双方の長所を活かし短所を補う形でプログラムの最適化を行い組合せ最適化問題の高速化を試みる．

2. OpenCL

OpenCL はハードウェア，ソフトウェア技術で有力なコンピュータ企業が多数加盟する標準化団体 Khronos Group によって策定され， CPU による並列処理システムに DSP, GPU などのアクセラレータを加えたヘテロジニアスな並列コンピューティングを行うために設計されたオープンなフレームワークである[3].

OpenCL にはホストとデバイスの概念があり，プログラムはホストである制御プロセッサからデバイスである演算用プロセッサで動作するカーネルコードを実行する形となっている．CPU のみでカーネルコードを実行する場合など，制御プロセッサと演算用プロセッサは同一でも問題はない．カーネルコードは OpenCL C 言語で共通化されており，ハードウェアによる動作と拡張命令の対応度に違いはあるものの，OpenCL C 言語に沿ったカーネルコードはどのプラットフォームでも実行できるようになっている．そのため，一度 OpenCL で実装すると他のプラットフォームへの移植の際に，カーネルコードのハードウェアに依存した実装とホストコードからカーネルコードを呼び出す部分を修正するだけで済み，実装の最適化に集中できる．

OpenCL のカーネルコードで実行される処理単位をワークアイテムと呼ぶ．同じグループのワークアイテムでは同一のプログラムが実行されるが，ワークアイテム ID によってそれぞれ異なるデータを処理することが出来る．一つのプログラムによって複数のデータが処理されることから，このような並列処理を SPMD と呼ぶ．

3. 巡回セールスマン問題

都市の集合とそれぞれの都市間の移動距離の集合が与えられたとき，全ての都市を 1 度ずつ通り距離が最少となる巡回路を求めるのが巡回セールスマン問題である．巡回セールスマン問題は前述した NP 困難に属する問題で，巡回路の数は(都市数-1)の階乗となる．そのため都市数が増えると組合せ爆発が起こり，数十都市ですら全ての問題を総当たりで行うことは現実的に不可能である．ちなみに，対称性のある問題では順方向と逆方向で巡回路の総距離は同じとなるので，巡回路の数は半分とみなすことが出来る．

巡回セールスマン問題のように NP 困難な組合せ問題を解くため，組合せ最適化アルゴリズムでは，問題の計算量を間引きして高速化が行われる．これには問題の唯一の最適解を見つけ出す厳密解法と，最適解を保証しない代わりに厳密解法より計算量が小さい近似解法の二つがある．

厳密解法に属するアルゴリズムとして分枝限定法があり，各種の最適化問題に適用できる汎用的な手法である．分枝限定法は分枝操作と限定操作の 2 つからなる．ある集合 S を与えられたとき，探索木を構成する部分問題の集合 S'を再帰的に生成することを分枝操作と呼ぶ．分枝操作を行う中で， 部分問題 S'_i の最小値の上限値と下限値を計算し，もし下限値が全体の暫定的な最小値を上回っていた場合，その部分問題から分枝操作で生成される以降の部分問題を生成する必要は無くなる．この工程を限定操作と呼び，探索木から部分問題を切り捨てることから刈り込みとも呼ばれる．

4. 予備実験

4.1 概要

OpenCL における CPU と GPU のパフォーマンスを比較するため，巡回セールスマン問題を列挙法で解くテストプログラムを実装した．列挙法では部分問題の計算は一切必要ないため，はじめに全ての 2 都市間の距離を計算してテーブル matrix を構築した後，全ての巡回路テーブル route を生成し，図 1 に示すカーネル関数 calc_route によって巡回路の距離を計算した．

† 松江工業高等専門学校 Matsue College of Technology

```
int i = get_global_id(0);
int j = 0;
output[i]=0;
for(j=0; j<n-1; j++){
  output[i] +=
    matrix[route[i*n+j]*n+route[i*n+j+1]
   ];
}
output[i] += matrix[ route[i*n+j] ];
//matrix  ：2都市間の距離テーブル
//route   ：都市番号を格納した巡回路
//output  ：計算された巡回路の総距離
```

図 1　関数 calc_route

この関数ではワークアイテムを用いた SPMD のデータ並列処理を使用しており，変数 i には get_global_id(0)により 0 から総巡回路数までの値が入る．ワークアイテムには今回の場合，i を用いた部分が i=0 から総巡回路数の値が入った処理に展開される．

このプログラムはデバイスで実行される際，例えば GPU のようにデータ並列処理をサポートするデバイスでは i の値が違う複数のカーネルコードが多数のストリームプロセッサで並列処理される．また，x86CPU のようにデータ並列処理がサポートされていないデバイスでは OpenCL のスケジューラにより i=0 から総巡回路数の値までの処理が逐次実行される．

4.2 検証

CPU に Phenom X3 8450(2.1GHz)，GPU に Radeon HD 4850 を用いて 6 都市から 8 都市のデータを入力し calc_route を CPU と GPU それぞれで 5 回ずつ実行した．ATI Stream Profiler 1.2 により計測した実行時間と，ホスト-デバイス間のデータの転送時間の平均値はそれぞれ表 1，表 2 のようになった．

表 1　calc_route の実行時間(単位:ms)

	CPU	GPU
6 都市	0.101102	0.197738
7 都市	0.597422	0.298146
8 都市	3.649262	0.7966

表 2　データの転送時間(単位:ms)

	CPU	GPU
6 都市	0.006848	0.197738
7 都市	0.015252	4.951466
8 都市	0.075976	5.371116

7 都市の時点で calc_route の実行時間は GPU が CPU を下回り，8 都市の場合は CPU と比べ約 5 倍高速に実行している．これは，上記のように calc_route が SIMD によるデータ並列処理を用いているため，多数のストリームプロセッサを搭載した GPU で並列実行されたからだと考えられる．しかし，データの転送時間では入力数に関わらず CPU が GPU と比べ高速である．

また，Stream Profiler で計測した GPU の ALU 使用率は表 3 のようになった．

表 3　ALU 使用率

	ALU 使用率
6 都市	15.26
7 都市	26.13
8 都市	28.96

いずれの場合も 30%を切っており，今回の実装では GPU の性能が最大限発揮されていないことが伺える．これは Radeon HD 4850 が 5-way VLIW を採用しているため，それを一切考慮していない今回のコードでは遊休状態のストリームプロセッサが存在してしまうことが原因として考えられる．

5. まとめ，および今後の課題

表 1 から，カーネルコードの実行時間について，データ並列処理が可能な GPU は CPU と比べ問題数が増えるほど有利になることが分かる．しかし，表 2 が示すようにデータの転送に関してはハードウェアの関係上，GPU は CPU に比べ遥かに遅いことが分かる．また，GPU のように並列処理を前提としたプロセッサの性能を発揮するためにはアーキテクチャを考慮したプログラムの最適化が必要なことが表 3 から分かる．

組合せ最適化アルゴリズムを高速化するためには，まず不要な部分問題を省いて計算量を低減することと，部分問題の下限値をすみやかに求めることが大切である．そのため組合せ最適化アルゴリズムを用いて NP 困難の問題を解いた実例では必ずと言っていいほど並列計算機が用いられており，その中には SIMD 型の並列計算機を用いたものも存在する[4]．しかし，GPU は上記のようにデータの転送が遅く，CPU ほど高度な分岐予測機構を搭載していないため，単独で分枝限定法のような組合せ最適化アルゴリズムを実行することは困難である．また GPU による計算はデータの転送コストが非常に大きいため，部分問題の計算で逐次呼び出すことは賢明ではない．一方，CPU では GPU ほど高速なデータ並列処理が行えないため，大規模な問題において性能が低下してしまう．こうしたそれぞれの欠点を CPU と GPU の分業を行うことで補い，処理の高速化を図る必要があると考えられる．

今後の研究ではこれらを踏まえ，CPU と GPU によるヘテロジニアスコンピューティング環境を構築し，分枝限定法の並列プログラムを実装する予定である．

参考文献

[1]Georgia Tech,"Optimal Tour of Sweden", http://www.tsp.gatech.edu/sweden/, (2004)
[2]Khronos Group, "OpenCL Overview", http://www.khronos.org/opencl/, (2008)
[3]土山 了士, 中村 孝史, 飯塚 拓郎, 浅原 明広, 三木 聡, "OpenCL 入門", インプレスジャパン(2010)
[4] 宮本 隆宏, 岩崎 一彦, "超並列計算機 MP-1 を用いたナップサック問題解法の試み", 電子情報通信学会技術研究報告, Vol93, No123(1993)