GPU を用いた自動並列チューナの開発 gPot

戸松 祐太

1　はじめに

近年，画像処理用演算装置（Graphics Processing Unit,GPU）を用いて流体力学，画像分析，統計データ処理などの科学技術計算の並列計算を実現する研究が注目され，CPU に対して数十〜百倍の高速化が図られたという結果が報告されている．[1) 2) 3)]

科学技術計算だけでなく他のアプリケーションにおいても GPU で処理する事による高速化が望めるが，GPU 用アプリケーション開発者にとって大きな負担になっている．なぜなら，開発者は並列処理的な考え方や CUDA などの GPU 専用プログラム言語の取得が必須であり，時間や労働といったコストが高くなるからである．この為，開発者環境が整っているにも関わらず GPU を利用したアプリケーションは普及していない．また一方で，GPU を搭載した PC は普及しているが，その性能を利用したアプリケーションが少ない為，リソースの有効活用が求められている．

本研究目的は，開発者に対し GPU を意識させる事なくその性能を十分に活かした高速なアプリケーション開発の支援により GPU リソースを有効活用させる事である．その為に，逐次実行型のソースファイルを CPU と GPU で並列処理できるよう自動変換できるツールの開発を行う．

2　GPU による高速化

2.1　GPU の概要

GPU とは画像処理を専門的に受け持つ補助演算装置である．GPU には動作周波数が 1〜2GHz のストリーミング・プロセッサ（SP）と呼ばれる演算ユニットが多数実装されている．CPU と比べて安価であり，1 回の命令で複数の演算ユニットで同時に行う SIMD 演算機として用いられている．同じ処理を並列に処理できることから，単純なデータを一度に大量に処理できるという特徴を有する．

また，GPU はとても演算性能が高い演算装置である．理論値で，最近の CPU「Core i7」の性能が 4 コアの総合で約 100GFLOPS であるのに対し，NVIDIA の GPU「FeForce GTX285」では約 1TGLOPS の性能を持つ．

2.2　マルチスレッドによる処理

GPU の性能を十分に引き出す為には，計算を並列に処理させ SP を同時に稼働させる必要がある．GPU「GeForce GTX285」では SP 数は 240 個であるが，その SP 数を超える数千，数万スレッドの処理を効率的に並列処理を行うことにより約 1TGFLOPS の性能を引き出す事ができる．

スレッドを効率よく並列処理させる為には SP と Device Memory（GPU 内メモリ）や Host Memory（計算機のメインメモリ）の関係（Fig. 1）を良く理解しコーディングを行う必要がある．具体的には下記の事が考えられる．

1. Host Memory と Device Memory 間の通信を減らす
2. GPU 内のシェアードメモリの最適化
3. SP と Device Memory のデータ通信の最適化
4. GPU 側では分岐処理を多用しない

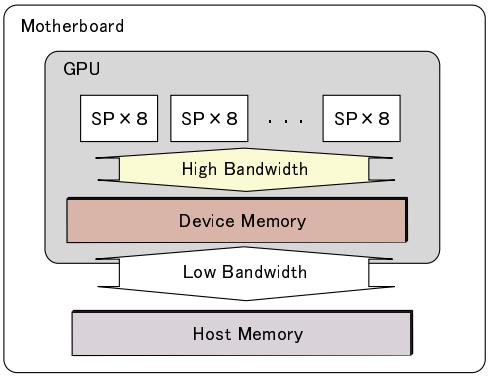

Fig.1　SP と Device Memory、Host Memory の関係

ここでは，上記の 1. について説明を行う．一般的な GPU は自身の中にもいくつかの専用のメモリを持っており，CPU 側から GPU 側へ処理を実行する際はデータをマザーボードを通して Device Memory にアクセスする．このマザーボード上のメインメモリと GPU 内メモリのデータ速度は約 1〜1.5GB/s ととても低速である為に，この間の通信を極力減らす事が重要となってくる．

上記のように、GPU により高速な処理をさせる為にはそのハードウェア構成によりコーディングが変わってくる。

3　自動並列化

2 章では GPU を用いて高速に処理させる為には並列処理を行う必要がある事を述べた．つまり，逐次型 CPU 用プログラムを GPU 用プログラムに変換する際には，逐次処理を並列処理に変換する自動並列化が必要となる．

自動並列化とはループとプログラム構造を解析し，プログラム中の並列実行可能な領域を見つける高度なプログラム解析に基づいた並列処理である．インテルが開発している自動並列コンパイラ [4)] の機能の 1 つに，予めテンプレートを用意し，それに従って並列処理用に変換すると言うものがある．

本研究で用いる自動並列化とは，ループ構造を解析しテンプレートとマッチした部分を変換する部分に注目する．下記では，そのテンプレートについて述べる．

3.1 独立な計算を行うループ構造

ループ内の変数に依存関係がない Fig. 2 のようなループ構造は Fig. 3 のように自動変換を行うことができる．これは，Fig. 2 の処理を 2 スレッドに分けることによりループの反復回数を半分にする自動並列処理である．CPU において，この並列化を行う事により最大 2 倍の高速化が望める．

```
for (i=1;i¡100;i++)
{
  a[i]=a[i]+b[i]*c[i];
}
```

Fig.2　ループ構造のコード

```
//Thread 1 for (i=1;i¡50;i++)
{
  a[i]=a[i]+b[i]*c[i];
}
//Thread 2
for (i=50;i¡100;i++)
{
  a[i]=a[i]+b[i]*c[i];
}
```

Fig.3　Fig. 2 を 2 スレッドで並列化した際のコード

3.2 総和計算を行うループ構造

上記のループ構造は，各ステップにおいて独立な計算を行うものであるが，Fig. 4 のように一つの変数に総和計算を行うループ構造も存在する．このような場合は，Fig. 5 のように処理を行う変数を複数の配列に分割し，それぞれの配列毎に各スレッドで処理を行うようにすることで並列処理が可能である．

```
for (i=1;i¡100;i++)
{
  add+=a[i];
}
```

Fig.4　総和計算を行うループ構造

4 提案ツール：gPot(GPU Parallel Optimization Too)

4.1 gPot について

C プログラムを GPU を利用して高速化を行う為には，そのプログラムの並列処理可能部分を見つけ出し，GPU で処理できる形に変換する必要がある．本提案ツール gPot は 3 章で説明した自動並列化の考えを用いて，C または C++ で書かれたソースファイルから並列処理できる部

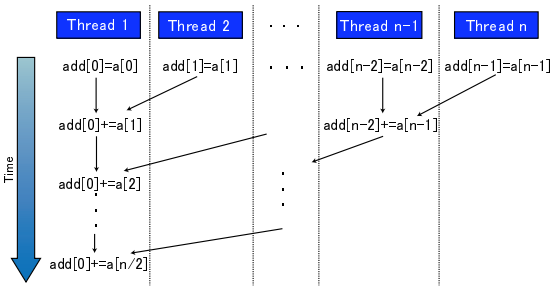

Fig.5　Fig. 4 を並列化した際の流れ

分を見つけ出し，CPU と GPU の処理を割り振りを段階的に最適化していくツールを目指す．

この gPot を用いることにより，開発者は GPU のプログラム技術を知らなくとも GPU を利用したアプリケーションを容易に作成する事ができ，アプリケーションの高速化の恩恵を受ける事ができる．

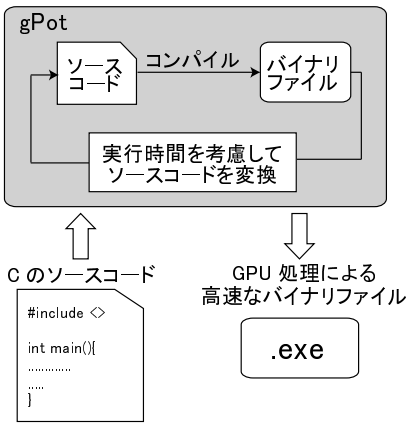

Fig.6　gPot の概要図

gPot のアルゴリズムは下記の通りであり，実行時間が最小になるようソースファイルの最適化を行う．

【gPot のアルゴリズム】

Step1　ソースファイルをコンパイルし，実行時間を計測する

Step2　C ソースファイルから並列可能部分を見つける

Step3　Step2 で見つけた部分を GPU 用プログラムに変換する

Step4　Step3 で変換を行ったソースファイルをコンパイルし，実行時間を計測する

Step4　Step1 の実行時間と Step4 の実行時間を比較し，早くなっていた際のソースファイルを受諾し Step2 に戻る

4.2 C プログラムから GPU 用プログラムへの変換

C プログラムを GPU 用プログラムへ変換する際には，NVIDIA 社が提供している CUDA プログラミングを用いて行う．この CUDA は C ベースで GPU に処理を投げることが出来る．

CUDA を用いて C プログラムから変換を行う際には，予め変換用のテンプレートを用意し，そのテンプレートにマッチするループ構造を抽出/変換を行うようにする．変換を施すループ構造は 3 章で説明した 2 つの構造に限る事にした．例えば，Fig. 3 を CUDA 用に変換する場合は Fig. 7 になる．

```
__global__ void gpot1(){
  share_memory[idx]+= "a[idx]" ;
  for(temp=2;temp¿THREAD_NUMBER;temp*=2){
   if(idx%temp==0)
     share_memory[idx]+=share_memory[idx+temp/2];
  }
}
```

Fig.7　Fig. 3 の変換後

4.3 gPot の最適化手法

2 章で GPU による高速化は GPU の特性を考えて行う必要があると述べた．つまり，ループ構造を全て GPU で処理を行っても高速化が望める訳ではないので，どの部分のループ構造を GPU で処理させるかを選択することはとても重要となる．

そこで上記の問題を，どのループ構造を選択するかを設計変数空間とし，変更を行ったプログログラムの実行時間を評価値と捉え組合わせ最適化として解く事を考える．用いる最適化手法は遺伝的アルゴリズムを考えている．

5 検証

ここでは，4.2 節の C プログラムから CUDA プログラムへの変換についての検証を述べる．簡単なπ計算を行うプログラム (Fig. 8) を gPot のアルゴリズム Step:3 に適応させ変換を行った．本検証を行う為に使用したマシンのスペックを Table 1 に示し，結果を Fig. 9 に示す．Fig. 9 の横軸は GPU で処理させるスレッド数で，縦軸は実行時間である．

```
#define num_steps 100000 main(){
 int i; float x, pi, step, sum=0.0;
 for( i=1;i¡=num_steps;i++ ){
  x = (i-0.5)*step;
  sum = sum+4.0/(1.0+x*x);
 }
 pi = step*sum;
}
```

Fig.8　π計算のプログラム

Table1　検証マシンのスペック

CPU	AMD Optern (1.8GHz × 2)
GPU	GeForce 8400GS(SP × 8 1.25GHz)
Main Memory	DDR2-SDRAM (2.5G)

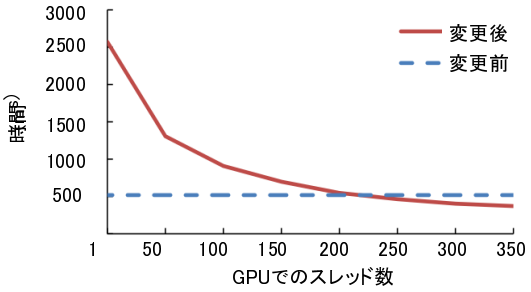

Fig.9　実行結果

Fig. 9 を見ると，スレッド数 200 を超えると GPU で処理させる方が実行時間が早くなっている事が分かる．しかし，それ以降では，実行時間が停滞気味である．今回の変換では，2.2 節の「Host Memory と Device Memory 間の通信を減らす」しか考慮していない．他の部分を考慮した変換を行う事により，Fig. 9 の実行時間よりも低くなると考えられる．

参考文献

1) 青木尊之．フル gpu による cfd アプリケーション．情報処理, Vol. 50, No. 2, pp. 107–115, 2009.
2) NVIDIA.　Cuda programming guide 2.3, 2009. http://www.nvidia.com/object/cuda_develop.html.
3) 藤本 典幸筒井 茂義．進化計算の並列化への課題：マルチスレッドプログラミングから超多スレッドプログラミング．進化計算シンポジウム, pp. 229–235, 12 2009.
4) インテル　コンパイラー　自動並列化ガイドデュアルコア/マルチコア対応アプリケーション開発 4. http://jp.xlsoft.com/documents/intel/compiler/528J-001.pdf.