遺伝的アルゴリズムを用いた自動並列化トランスレータの提案

戸松祐太[†1]　吉見真聡[†2]
廣安知之[†3]　三木光範[†2]

ソフトウェア実装に際して，アルゴリズムから抜き出した高い並列性を持つ領域を，GPU などの専用ハードウェアで処理する方法がよく行われるようになってきている．本研究報告では，困難を伴う並列化領域の抽出と判断に対応し，GPU を用いたプログラム実行の高速化を図る手法を提案する．本手法では，プログラム中で GPU に割り当てる処理を遺伝的アルゴリズムに基づいて最適化し，実行時間の短縮を図る．本研究報告では，様々なループ文で構成した自作テストプログラムとベンチマークプログラムを対象とした評価から，実装したトランスレータの定量的な検討を行う．

Proposal of the Automatic Parallelize Translator using Genetic Algorithm

YUTA TOMATSU,[†1] MASATO YOSHIMI,[†2]
TOMOYUKI HIROYASU[†3] and MITSUNORI MIKI [†2]

In developing software on GPU, it is important to find codes including parallelism, and execute these codes on GPU. We propose the translator which accelerates general C-code used GPU partially by ptimizing computing time circulating searching code regions which should be executed on GPU.We evaluated the translator by applying a test program which has variant loop-structures and a benchmark.From this result, This paper discusses tuning technique for the translator based on quantitative performance derived.

1. はじめに

汎用プロセッサは，動作周波数を高めた際の性能向上に消費電力量が見合わなくなった 2002 年頃から，プロセッサベンダは複数の演算コアによる並列処理で性能向上を図るようになった．IBM，東芝，SCE による Cell/B.E. が 9 コア，AMD の six-cores Opteron など，多種のマルチコアプロセッサが現れてきている．そのような状況にあって，従来メディア処理などの専用の目的で利用されてきたハードウェアを汎用計算に応用しようとする動きが活発になっている[1)2)3)]． PC において一般的なメディア処理専用デバイスとしては，GPU(Graphical Processing Unit) が挙げられる．GPU はリアルタイムな画像処理を対象に利用されるハードウェアであるが，膨大な演算の量に対して各演算自体は単純かつ並列性が高いため，近年では数百を超える演算コアを持つメニーコアプロセッサとしての性質を帯びるようになっている[4)5)6)]．

中国のスーパーコンピュータ Tianhe-1A は，Intel Xeon と NVIDIA Tesla による複合型の計算システムを用いて 2.507 PFlops を記録し，2010 年 11 月集計の高速計算機ランキングプロジェクト Top500[7)] では第 1 位に位置している．このシステムだけでなく，東京工業大学の TSUBAME や長崎大学の多体問題向け GPU クラスタなど，低コストな高性能アクセラレータとして GPU の利用が拡大している．

GPU を汎用の処理を行わせる技術は GPGPU (General Purpose computing on GPU) と呼ばれ，主要なベンダである NVIDIA や ATI は，GPGPU のための環境を整備し，提供するようになってきている．しかしその一方で， GPU を用いる場合のソフトウェア開発の困難さは依然として大きな問題の 1 つに挙げられる．

汎用マイクロプロセッサと GPU が協調動作する異種混合型の計算システムを用いて，その性能を大きく引き出すためには，各プロセッサの特徴に合わせた独自かつ高度なプログラミング技術が要求される．

近年では開発者の負担を軽減させる研究/開発が行われてきている．例えば，ストリーム処理可能なコードを明示することにより，自動的に処理を判断し，計算を実行するデバイス

†1 同志社大学大学院 工学研究科
Graduate School of Engineering, Doshisha University
†2 同志社大学 理工学部
Faculty of Science and Engineering, Doshisha University
†3 同志社大学 生命医科学部
Faculty of life and medical sciences, Doshisha University

を切り替えるフレームワーク[8] や，並列性が高い領域にディレクティブを挿入することでGPU 用のコードを自動生成するコンパイラ[9] などの開発が進められている．

本研究ではこれらの技術を踏まえて，C のプログラムを GPU 搭載型の並列計算環境向けに最適化する手法を提案する．提案手法では，対象プログラムの中から並列実行が望めるループ文の選択を組み合わせ最適化問題と設定し，CPU と GPU で実行するループ領域の配分を最適化する事により対象プログラムの実行時間の最小化を図る．本研究報告では，様々なループ文で構成した自作テストプログラムとベンチマークプログラムを対象とした評価から，実装したトランスレータの定量的な検討を行う．

2. GPU コンピューティングの概要

2.1 GPU の概要

GPU とは画像処理を専門的に受けもつ補助演算処理装置である．ここでは，GT200 シリーズである GTX280 を例に，GPU アーキテクチャの説明を行う．GT200 シリーズでは下記に示す主な要素で成り立っている．

- Texture Processor Cluster (TPC)
- Thread Execution Manager
- Device Memory
- L2 cache

TPC はいくつかのコアを搭載している演算ユニットの部分であり，GTX280 には，Texture Filtering Unit(TF) といくつかの Streaming Multiprocessor(SM) を内包している TPC が 10 個搭載されている．この各々の SM は下記に示す要素で構成されている．

- 8 Streaming Processors(SPs)
- 2 Special Function Units(SFUs):
- shared memory
- An instruction and data L1 cache

2.2 CUDA プログラミング

現在，GPU で汎用計算をさせるためのプログラム環境は，ATI や NVIDIA 社から提供されている．その中で NVIDIA 社が提供している CUDA について記述する．

CUDA で作成したアプリケーションはホストとデバイスで動作する．このホストとデバイスはそれぞれ CPU と GPU に対応する．CUDA プログラミングでは，デバイスでの処理をホスト側で関数（カーネル）として呼び出す必要があり，この関数の呼び出しの際にデバイス側で動作させるスレッド数を CUDA で定義されているグリッド数，ブロック数を用いて指定することができる（図 1）．また，カーネルを通してデータを送信する際には，デバイスメモリの確保とデータ送信を明示しなければならない．

```
1 /*function processing on GPU*/
2 __global__ void vecAdd(float *a,float *b){
3   int tid=threadIdx.x; /*Getting thread ID*/
4   a[tid]=a[tid]+10;
5 }
6
7 /*main function on GPU*/
8 int main(void){
9    ...
10   vecAdd <<< 1,256 >>> (dA,dB);
11   ...
12}
```

図 1 CUDA プログラミングの例
Fig. 1 Example of CUDA's program

2.3 PGI アクセラレータ

PGI アクセラレータは，CPU-GPU 間のデータ移動や GPU 演算の呼び出しなど，CUDA 独自の知識が無くとも GPU の利用を可能とするコンパイラ[9] である．

PGI では，プログラム構造とデータを自動的に分析することにより，指定されたループ領域を GPU で処理するバイナリファイルを自動で生成する．

並列化が望めるループ領域に対して，PGI ディレクティブを差し込んだ例を図 2 に示す．1 行目の「#pragma acc region」が PGI ディレクティブである．上記のソースコードを PGI コンパイラでコンパイルする事により，GPU を考慮した実行ファイルを生成する．

2.4 GPU 用ソフトウェア開発の現状

開発環境 CUDA や PGI アクセラレータなど GPU を利用しやすい状況が整ってきている一方で，従来のプログラムコードを GPU で実行するための手法は大きな進展が見られていない．その為，開発者には対象アプリケーション毎における GPU 専用のコードの実装や，GPU で処理するコード領域を明示する作業が要求される．また，実行プロファイルの検討を通して，高速化に寄与するコード領域の調査は可能であるものの，CPU と GPU が協調動作するシステム開発に要する人的，時間的コストは依然大きい．

```
1 #pragma acc region
2 for( i=0;i<N;i++ ){
3   for( j=0;j<N;j++ ){
4      c[i*N+j]=a[i*N+j]+b[j*N+i];
5 }}
```

図 2 PGI ディレクティブの例
Fig. 2 Example of PGI's directive

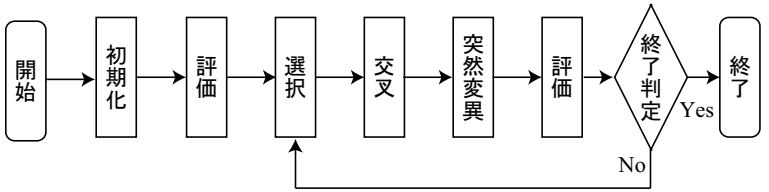

図 3 GA のフローチャート
Fig. 3 Flowchart of GA

3. 自動並列化トランスレータの提案

3.1 提 案 手 法

2.4 節で述べた開発コストの問題を低減するために，逐次型プログラムの一部を GPU に処理させる自動コード変換手法を提案する．GPU は一般的に，データ依存性の無い繰り返し文のブロックを，並列実行する事で高性能を実現する．その一方で，ループ回数が少なかったり，またはブロック処理が軽量であったりする場合は，メインメモリと GPU のメモリ間でのデータ通信のオーバヘッドが大きくなり，性能が上がらない場合もある．その為に，各ループ分の処理を適切な実行デバイスへの割り振り方を決定することはとても重要である．この割り振り方を組み合わせ最適化と見なす事ができる為，プログラムを GPU-CPU 協調コードに変換する際に遺伝的アルゴリズムを用いる．CPU コードから GPU コードへの変換には PGI ディレクティブ（以下，ディレクティブ）を用いる．

3.2 遺伝的アルゴリズム

遺伝的アルゴリズム (GA:Genetic Algorithm) は生物の進化過程を工学的に模倣した確率的な最適化手法である．GA ではある世代を形成している個体の集団を母集団と呼ぶ．また，この母集団の中から自然界の進化過程と同様に，環境の適合度の高い個体が高い確率で選択される．そして，その個体に対して交叉や突然変異がある確率で発生する事により次世代の母集団が形成される．これらの遺伝的操作を繰り返し，最後に得られた母集団の中で最も適合度の高い個体を最適解とする．

GA のフローチャートを図 3 に示し，GA のアルゴリズムを下記に示す．

(1) 【初期化】予め設定された数の個体を生成する．この個体数を母集団サイズと呼ぶ．
(2) 【評価】 個体の遺伝子情報を基に，各個体の適合度を設定する．適合度の良い個体ほど良好な個体と言える．
(3) 【選択】 適合度に依存した一定規則に従い，次世代に残す個体を選ぶ．
(4) 【交叉】 選択された個体間で遺伝子の一部を交換して子個体を生成する．
(5) 【突然変異】各遺伝子に対してある確率で変化を加える．
(6) 【終了判定】予め定められた終了条件に基づいて GA の処理を終了する．

3.3 提案手法のアルゴリズム

提案手法では，CPU と GPU で処理させるループ領域の配分を最適化する問題を，PGI のディレクティブ挿入位置の最適化問題と捉え，その問題を GA で解く．最適なディレクティブ挿入位置の最適化は下記の手順により行う．

(1) C ソースファイルを読み込む．
(2) ソースの先頭からループ文を順序に抽出する．
(3) 抽出したループをランダムに選択し，ディレクティブを挿入する．このディレクティブの挿入パターンを GA の母集団サイズの数だけランダムに作成する．
(4) 各挿入パターンを遺伝子に変換する．
(5) 変換した遺伝子を用いて，GA によるディレクティブの挿入位置の最適化を行う．

ディレクティブの挿入パターンを遺伝子に変換する際には，PGI アクセラレータの特性を考慮して行う (3.4 節で詳述)．そして，GA における評価では，遺伝子の情報によりディレクティブを挿入されたファイルの実行時間を適合度として行う．上記の処理の概要図を図 4 に示す．

3.4 遺伝子の設計

PGI アクセラレータでは，複数のループをまとめてディレクティブを挿入する事により，それらを一度に GPU で処理し，CPU と GPU 間のデータ通信を減らす事ができる．しかし，領域の並列性が低い場合，GPU への通信オーバヘッドが並列実行による高速化を上回る場合がある．その為，ディレクティブを挿入するループ文を適切に把握する必要がある．

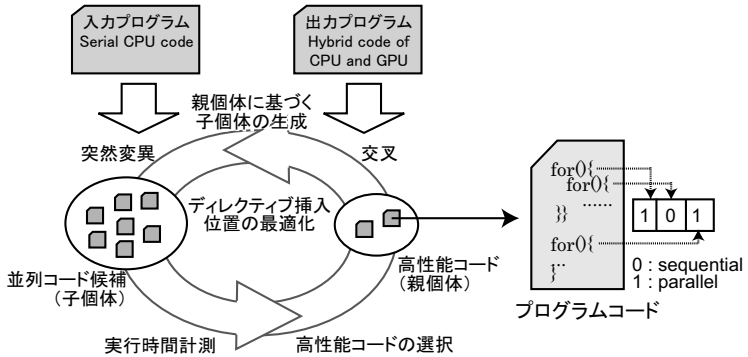

図 4 GA による並列化コードの自動生成
Fig. 4 Concept of the proposal translator

提案手法ではディレクティブを挿入するループ文の位置情報と，後続のループを含めてディレクティブを挿入するか否かの判定情報を遺伝子として表す．

上記の情報を表す為に，1 ループに対応する遺伝子は 2bit で表現する．図 5 に入力されたソースファイルのループから遺伝子への変換の概念図を示す．2bit 中の 1bit(A) は，そのループにディレクティブを挿入するか否かの判定を行い，もう 1bit(B) は後続のループを含めてディレクティブを挿入するか否かの判定を行う．

(A) の bit が 1 の時，その遺伝子に対応するループにディレクティブを挿入する．なお，ディレクティブが挿入するループの内側に別のループが存在する場合，遺伝子情報に関係なく内側のループを含めて GPU で処理を行う．(B) の bit が 1 の時，後続のループの遺伝子 (A) が 1 の場合に限り，後続のループも含めてディレクティブを挿入する．

上記の (A)(B) の遺伝子によりディレクティブの挿入位置と挿入範囲を表現し，この遺伝子を用いて GA により最適なディレクティブの挿入範囲の最適化を行う．

4. 提案トランスレータの予備評価

提案トランスレータの予備評価を行う為に，様々なループ構造を持った自作のテストプログラムを用いて検証を行った．本検証の評価は，提案トランスレータによりディレクティブの挿入位置の最適化が行われた入力プログラムの速度向上率と，そのプログラム内のディレクティブの挿入位置の判定により行う．

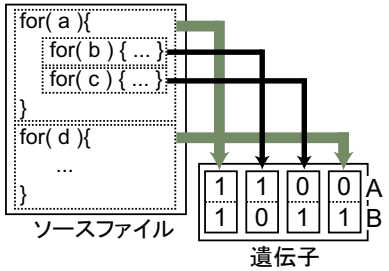

図 5 ループ文と遺伝子の関係
Fig. 5 Relations of loop sentences and the gene

表 1 システム環境
Table 1 Environment of evaluation

CPU	Intel Xeon w3530 (4 × 2.80GHz)
メインメモリ	DDR3 (6.0GB 1066MHz)
OS/kernel	Debian lenny5.0.6 / 2.6.26-2
GPU	NVIDIA Tesla C2050
コンパイラ	PGI 10.9-0

4.1 検証環境

本検証に用いた計算機は NVIDIA 社の Tesla C2050 の GPU を搭載した SUPERMICRO 社の Super Workstation 5046A-X である．本検証に用いたシステム環境の詳細を表 1 に示す．

4.2 対象問題

作成したテストプログラムでは，グレースケール画像処理に用いられる 2 次元配列の行列計算を主に行う．具体的には，3400 × 3400 の大きさの配列を 3 つ用い，それらの 2 次元配列の各要素の代入，四則演算，数学関数を用いた演算，要素が異なる配列の演算を行う．このテストプログラムのループ構成の概要を図 6，図 7，および図 8 に示す．図 6 内の for(2) から for(6) のループは，3 つの 2 次元配列に対して代入や演算処理を行うので，2 階層の入れ子構造をとる．図 8 の for(8-a) 内には，GPU で演算出来ない逆三角関数が存在する．

4.3 提案トランスレータのパラメータ

提案トランスレータのパラメータを表 2 に示す．本トランスレータの最適化手法では GA オペレーションを用いており，トランスレータのパラメータは GA のパラメータとな

```
for(1)：並列性の無い処理
for(2)：固定値を 3 つの配列に代入
for(3)：ループ数により変動する値を
        3 つの配列に代入
for(4)：四則演算
//逐次処理
for(5)：数学関数を用いた演算
for(6)：要素が異なる配列の演算
for(7)：分岐により配列の異なる要素
        に固定値を代入
for(8)：分岐により内部のループの処
        理を分ける
```

図 6 テストプログラムのループ構成の概要
Fig. 6 Outline of the test program

```
for(7)
   for(7-a)
      if( )
        //3 つの配列への代入
      else
        //3 つの配列への代入
```

図 7 Conduct of for(7)

```
for(8)
   if( )
      for(8-a)：GPU で処理できない数学関数を
                用いた演算
   else
      for(8-b)：数学関数を用いた演算
      for(8-c)：要素が異なる配列の演算
```

図 8 conduct of for(8)

表 2 提案トランスレータのパラメータ
Table 2 Parameter of the proposal translator

最大世代数	200
母集団サイズ	20
遺伝子長	38
突然変異率	1/遺伝子長
トーナメントサイズ	4

る．なお，遺伝子長は，入力プログラムのループ数に依存する為に，テストプログラムでは $19 \times 2 = 38$ となる．

4.4 検 証 結 果

7 試行における，提案トランスレータの各ステップ毎のテストプログラムの実行速度向上率の平均と中央値の変動履歴を図 9 に示す．また，4.1 節で説明したテストプログラムにおける最適なディレクティブ挿入位置を図 10 に示し，7 試行中最も速度向上率が良かったプログラムにおけるディレクティブ挿入位置を図 11 に示す．なお，図 10 のプログラムの速度向上率は 1.92 倍であり，速度向上率はこの数値に近い程良いとする．本検証では，図 10 のプログラムの速度向上率を最適解と呼ぶ事にする．

図 9 を見ると，ステップ毎に最適解に向けて速度向上率が改善されている事が分かる．これは，GA オペレーションにより，最適なディレクティブの挿入位置の探索が正常に動作している事を意味する．また，図 9 と図 11 を見ると，最適なループの位置にディレクティブが挿入されず，速度向上率が最適解に達していない事が分かる．これは，本トランスレータで実装した最適化手法は単純 GA（SGA:Simple GA）であることが原因と考えられる．すなわち，今後，探索能力に優れた GA により解の探索を行う必要がある．

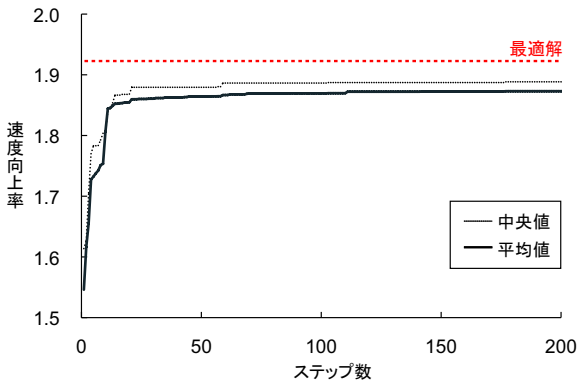

図 9 提案手法による速度向上率の履歴
Fig. 9 History of performance rate

5. 姫野ベンチマークプログラムによる評価

ベンチマークを使った評価として，姫野ベンチマーク[10] を用いた提案トランスレータの評価を行った．本検証に用いた提案トランスレータのパラメータ，および検証環境は，4 章と同じである．

5.1 姫野ベンチマークプログラム

このベンチマークはポアッソン方程式をヤコビの反復法で解く主要なループの処理速度を計測するものである．その為，プログラムコードは，13 個の 3 次元配列への値の代入処理を行うループが 2 つ，ヤコビの反復法の計算を行うメインループ 1 つで構成されている．このメインループは 3，および 4 階層の入れ子構造になっている．

5.2 結　　果

このベンチマークを用いて 10 試行の評価を行い，10 試行間の探索履歴の平均と中央値をプロットしたグラフを図 12 に示す．

```
for(1){}
#pragma acc region{
for(2){}
for(3){}
for(4){}
}
//逐次処理
#pragma acc region{
for(5){}
for(6){}
for(7){}
}
for(8){
  for(8-a){}
  #pragma acc region{
  for(8-b){}
  for(8-c){}
  }
}
```

図 10 最適なディレクティブの挿入位置
Fig. 10 Optimal location of directive

```
for(1){}
for(2){}
#pragma acc region{
for(3){}
for(4){}
}
//逐次処理
#pragma acc region{
for(5){}
for(6){}
}
#pragma acc region{
for(7){}
}
for(8){
  for(8-a){}
  #pragma acc region{
  for(8-b){}
  for(8-c){}
  }
}
```

図 11 7 試行中最も速度向上率が良かったプログラムのディレクティブ挿入位置
Fig. 11 Optimal location of directive

図 12 から，5 ステップまでに速度向上率が収束しており，テストプログラムの結果である図 9 と比べると収束速度が速くなっている事が分かる．これは，姫野ベンチマークプログラムの主な処理部分が 1 つのループ内で行われている為に，ディレクティブを挿入する位置を簡易に発見出来るからである．この結果より提案トランスレータにより実行速度が 2.5 倍近くなっている事から，ベンチマークに対しても高速化が実現出来る事が確認出来た．

6. ま と め

本報告では，コードから GPU で処理する領域の探索を繰り返し，実行速度の最適化を図るとともに，開発者の負荷の低減を目的とするトランスレータを提案した．種々のループ構造を持つ自作のテストプログラムと姫野ベンチマークを提案トランスレータに用いる事により評価を行った．評価の結果，テストプログラムと姫野ベンチマークの双方の高速化を確認する事が出来た．しかし，使用した GA が SGA であり，解探索が十分に行えない．そのため，今後はさらに解探索能力の高い GA を利用する必要がある．

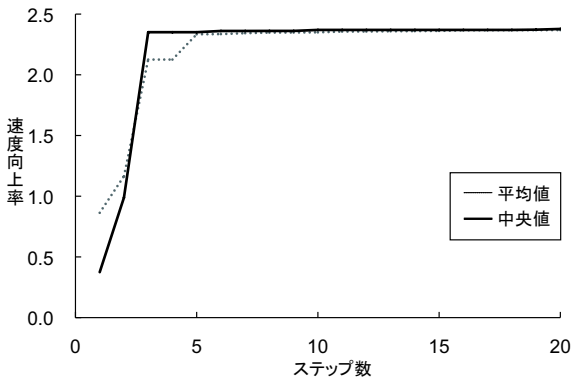

図 12 10 試行における速度向上率の履歴
Fig. 12 History of performance rate

参 考 文 献

1) Cevahir, A.: High performance conjugate gradient solver on multi-GPU clusters using hypergraph partitioning, *Computer Science - Research and Developmen*, Vol.25, No.1-2, pp.83–91 (2009).
2) Matsuoka, S.: GPU accelerated computing-from hype to mainstream, the rebirth of vector computing, *Physics: Conference Series*, Vol.180, No.1 (2009).
3) Dixon, P.R.: Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition, *Computer Speech and Language*, Vol.23, No.4, pp. 510–526 (2009).
4) NVIDIA: CUDA Programming Guide 2.3 (2009).
5) Cardinal, P.: GPU Accelerated Acoustic Likelihood Computations, *Proc. of INTERSPEECH* (2008).
6) Fujimoto.N: Dense Matrix-Vector Multiplication on the CUDA Architecture, *Parallel Processing Let- ters*, Vol.18, No.4, pp.511–530 (2008).
7) : Top500 Supercomputing Sites, `http://www.top500.org/`.
8) Buck, I., e.a.: Brook for GPUs : Stream Computing on Graphics Hardware, *ACM Transactions on Graphics*, Vol.23, No.3, pp.777–786 (2004).
9) Group, T.P.: PGI Fortran & C Accelerator Programing Model (2010). ver 12.
10) : High Performance Computing, `http://accc.riken.jp/HPC/HimenoBMT.html/`.