GPU 用自動並列化コンパイラを用いた Fortran コード最適化手法の提案と評価

田中 裕也
Yuya TANAKA

1　はじめに

近年，画像処理用のプロセッサである GPU (Graphics Processing Unit) を，画像処理以外に活用する GPGPU (General-purpose computing on GPUs) の研究が広く行われている．最近ではコンシューマ向けの GPU も GPGPU に対応し，高密度な演算能力の利用が広く一般に行えるようになることが期待されている．一方で，GPGPU の利用に必要な並列コードの作成には，整備された開発環境が存在する [1] 現在でも，学習や開発に莫大なコストの投入が必要である．このことから，プログラムコードに特別なコメント行 (ディレクティブ) を付加すると，GPU 用の並列コードを出力する自動並列化コンパイラが開発されている [2]．しかし，ディレクティブ型コンパイラを用いて十分な性能を持つ並列コードを得るためには，GPU に関する高度な知識が要求される．

そこで本研究では，ディレクティブの付加と実行時間の計測を繰り返すトランスレータを用いて，GPU 用のディレクティブ型コンパイラを用いた並列化とチューニングを自動的に行う並列化手法を提案する．

2　GPGPU

GPU は内部に多数の演算用コアを内蔵し，階層的な並列構造を持つプロセッサである．その利用には専用の並列コードを作成する必要がある．NVIDIA 社の GPGPU 開発環境 CUDA を用いる場合，並列化に適するアルゴリズムを対象に，下記の 2 つの実行単位を用いて実装することで並列コードを作成することができる．

- ブロック: 非同期の並列性 (SPMD)
- スレッド: 同期的な並列性 (SIMD)

GPGPU では，対象とするアルゴリズムその記述方法，計算資源の割り当てが実行性能に大きな影響を与えることが考えられる．そのため，並列コードの作成には適切なチューニングが必要であるが，その主な方法はプロファイル情報等に基づく経験的なものである．

3　PGI Accelerator

PGI Accelerator は GPU 用のディレクティブ型コンパイラ製品であり，ディレクティブを付加した逐次的な C または Fotran 言語のコードの一部を，CUDA の並列コードに変換する．Fig. 1 にディレクティブの利用例

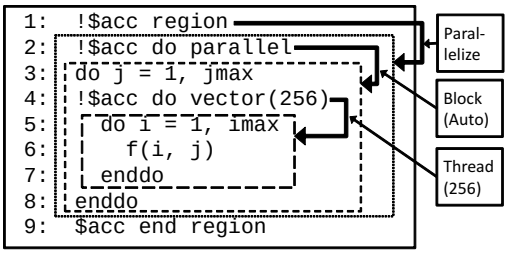

Fig. 1　ディレクティブの例

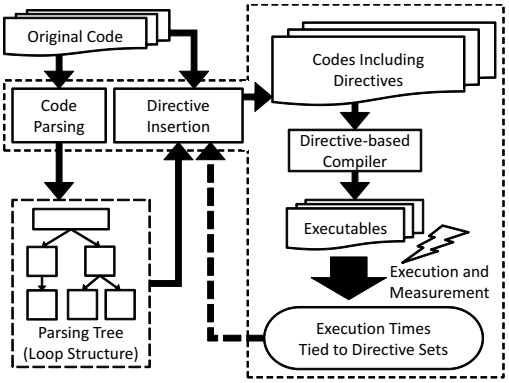

Fig. 2　最適化手法を実現するシステムの構成

を示す．図中の 1，9 行目のように，`!$acc region` と `!$acc end region` で並列化領域を指定すると，領域内のループの自動並列化を試みる．また，図中の 2，4 行目のように，ループ文に対して `!$acc do` および `parallel` と `vector` の組み合わせを用いて Kernel ループスケジューリングを指定すると，そのループ文の並列化に用いるブロックとスレッドの数を指定することができる．

4　ディレクティブ最適化

4.1　アルゴリズム

Fig. 2 に提案する並列化手法の全体像を示す．本手法では下記の手順により最適な並列コードを得る．

1. プログラムコード中のループ構造を解析する
2. 解析結果をもとにディレクティブを付加する
3. コンパイルし実行時間を計測する
4. (1)〜(3) の処理を繰り返し，実行時間が最も短いディレクティブ付加内容を得る

提案する並列化手法の利点として，使用するディレクティブ型コンパイラを変更するだけで，単一のプログラ

ムコードから複数のアーキテクチャに対応する並列コードを作成することが可能となる点が挙げられる．一方，この手法の欠点として，付加するディレクティブ内容の集合の大きさに応じて自動処理の工数が膨大となることが挙げられる．

4.2 実装

本研究では Fortran 言語を対象とし PGI Accelerator をバックエンドに用いるトランスレータを実装した．本実装は並列化領域または Kernel ループスケジューリングを最適化する．並列化領域については，各ループ文に対する領域の有無の組み合わせを全て網羅する．Kernel ループスケジューリングについては，多重ループの各ループ文に対してパラメータ空間を網羅する．パラメータ空間はブロック数，スレッド数共に $4^n (n = 0, 1, ..., 5)$ かつ 1 ループ文のブロック数とスレッド数の積が 1024 以下，各ループのブロック数の積が 1024 以下，スレッド数の積が 256 以下とした．

5 評価

5.1 評価条件

Table 1 に示す環境で，実装したトランスレータを用いてベンチマークコードの並列化を行い，5 回実行した際の実行時間の平均を，CPU での逐次実行時と比較した．評価に用いたベンチマークコードを下記に示す．

- NPB EP 3.3.1 Serial CLASS=W (以下 ep)
- ルンゲ＝クッタ法による 4 変数連立 1 次方程式の解 (以下 runge)
- マンデルブロ集合 (以下 mset)
- 1023 次元の正方行列とベクトルの積 (以下 matvec)
- 1024 次元の正方行列の行列積 (以下 matmul)
- 積分の計算 (以下 intgl4)
- 姫野ベンチマーク f90 M サイズ (以下 himeno)

5.2 並列化領域の最適化

Fig. 3 に，各ベンチマークコードの並列化領域最適化による速度向上率を示す．縦軸は，CPU での逐次的な実行に対する速度向上率である．matmul，himeno，intgl4 は 4 倍から 96 倍の高速化が得られた．一方で，runge はアルゴリズムに並列性がないため，また，ep，mset，matvec はコードの記述方法が並列化に不適であったため，実行速度が向上しなかった．

Table 1 評価環境

	Machine 1	Machine 2
CPU	Xeon W3530 2.8GHz	Core i5 2400 3.1GHz
RAM	6GB	8GB
GPU	Tesla C2050	GeForce GTX 460
OS	Linux 2.6.26 x86_64	Linux 2.6.38 x86_64
Compiler	PGI Accelerator 2010 (10.9)	
Compiler Option	-Minfo=accel,inline,ccff -fastsse -Minline=size:1000,levels:10,reshape -ta=nvidia,cuda3.1	

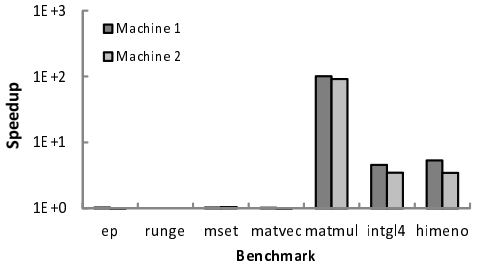

Fig. 3 並列化領域の最適化結果

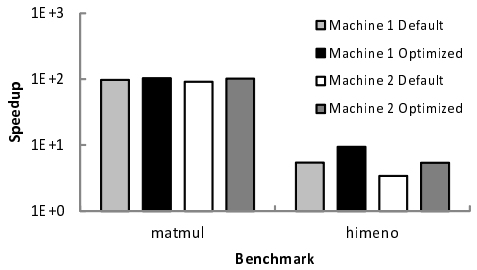

Fig. 4 Kernel ループスケジューリングの最適化結果

5.3 Kernel ループスケジューリング

Fig. 4 に，matmul と himeno のメインループに対する Kernel ループスケジューリングの最適化による速度向上率を示す．図中の Default は Kernel ループスケジューリングの指定なしで並列化を行った際の速度向上率を表している．最適化の結果，himeno ではさらに 1.5 倍の高速化を実現した．最適化の効果はアルゴリズムやデバイスのモデルによって異なることが確認された．

6 まとめ

本研究では，ディレクティブの付加と実行時間の計測を繰り返すことで，最適なディレクティブ内容を得る並列化手法を提案し，ベンチマークプログラムに適用することでその有効性を検討した．その結果，アルゴリズムやその記述方法の条件を満たすコードに対して，提案した並列化手法が有効であることを示した．

参考文献

1) NVIDIA. Compute Unified Device Architecture Programming Guide, 2007.

2) Wolfe, Michael. Implementing the PGI Accelerator model. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pp. 43–50, New York, NY, USA, 2010. ACM.