Vol.2011-ARC-197 No.9
Vol.2011-HPC-132 No.9
2011/11/28

GPU向け自動並列化コンパイラを用いたFortranコード最適化手法の評価

田中裕也[†1]　吉見真聡[†1]
三木光範[†1]　廣安知之[†2]

マルチコアプロセッサが一般的になった近年では並列処理の重要性が高まっている．GPU などの専用ハードウェアによる並列処理を利用したコストの削減と計算性能の向上が期待されている．しかしその利用には専門的な技術や知識が必要なことから開発コストが大きくなる．このためプログラムコードに指示子を付加することで用いる自動並列化コンパイラを活用した並列化が普及しつつある．一方計算時間は実行環境やアルゴリズムに依存する．指示する並列化領域やパラメータのチューニングにはプログラムの試作と評価の繰り返しが必要となる．本研究では自動並列化されたプログラムの実行時間をもとに最適なディレクティブ付加位置を決定するトランスレータを実装した．科学計算で広く利用されている Fortran 言語を対象とし，ベンチマークプログラムと科学計算アプリケーションで評価した結果を議論する．

Evaluation of Optimization Method for Fortran Codes with GPU Automatic Parallelization Compiler

Yuya Tanaka,[†1] Masato Yoshimi,[†1] Mitsunori Miki[†1]
and Tomoyuki Hiroyasu[†2]

Parallel processing has been more important according to recent trend of multi- and many- core processors. Even utilizing dedicated hardware such as GPU grows popular to achieve high performance maintaining low-operating costs, exploiting their potential requires architectural specific knowledge and techniques and following high cost of development. While recent automatic parallelization compilers which translate general program codes to the dedicated hardware by adding specifier to the program code have widely used, the tuning cost to optimize the region of parallelization and their parameters remains high. As the performance fluctuates according to the data of application and computing environment, obtaining the best combination of region and parameter requires circulation of prototyping and evaluation. To reduce developing load, this paper proposes the auto-tuning method by inserting specifier and measuring execution time. Implemented translator searching optimized location of specifier in programs written in Fortran is evaluated by several scientific benchmarks on two NVIDIA's GPUs and a microprocessor.

1. はじめに

計算タスクを分割し処理することで処理効率を向上させる並列処理の研究が，計算機科学の分野で広く行われてきた．複数タスクの同時実行が可能なマルチコアプロセッサが普及した近年では，これらの研究は重要な役割を果たすと考えられる．また，従来画像処理に用いられてきたグラフィックスプロセッサ (GPU: Graphics Processing Unit) を汎用計算に利用する技術も整備されてきた．GPU が内蔵する数百個の演算コアによる並列処理を利用し計算性能の向上と運用コストの低減を両立した計算システムも開発された[1)]．しかし，CUDA[2)] や OpenCL[3)] など複数の整備された開発環境が利用可能となった現在でも，GPU を用いて処理効率の向上を図るにはデバイスとアルゴリズムの双方に対する深い理解を要求され，学習や開発に莫大なコストの投入が必要となる．

並列プログラムの開発に必要なコストを低減するために，逐次的なプログラムコードをマルチスレッド化，ベクトル化されたプログラムコードに変換する自動並列化コンパイラの開発が複数進められている[4)5)]．ディレクティブ型の自動並列化コンパイラは記述の追加のみで並列化でき細かなチューニングが不要であることから，多数のアプリケーションで用いられている[6)7)]．しかし，並列プログラムの性能は並列化する領域の選択やデバイスのアーキテクチャと構成，対象アルゴリズムのパラメータなどの違いに大きな影響を受けるため，ディレクティブ型コンパイラで処理効率の向上を図るには試作と評価を繰り返す必要がある．並列処理により性能向上を図る計算システムは今後も増加していくことが予測されるため，プログラムコードや実行環境に依存せず並列プログラムの最適化を行うことが求められている．

そこで我々は，プログラムコードからディレクティブ付加内容の候補を抽出と，並列プログラムの生成と評価を繰り返す作業を自動化し，最適な並列プログラムを得る手法を提案

†1 同志社大学 理工学部
Faculty of Science and Engineering, Doshisha University
†2 同志社大学 生命医科学部
Faculty of Department of Life and Medical Science, Doshisha University

Vol.2011-ARC-197 No.9
Vol.2011-HPC-132 No.9
2011/11/28

する，本研究では提案手法の評価のため，科学計算アプリケーションで広く用いられている Fortran 言語を対象に GPU 用ディレクティブ型コンパイラである PGI Accelerator を用いて，実行時間の計測結果をもとにディレクティブ付加位置の評価を行うトランスレータを実装した．ベンチマークや科学計算アプリケーションを対象に実装したトランスレータを適用し，得られた結果について議論する．

2. 関連研究

GPU 用並列プログラムの開発コストを軽減するために，GPU 向けの並列化に既存の並列プログラミング言語を利用可能とする方法が提案されている[8]．大島らは，プログラムコードに特別なコメント行 (ディレクティブ) を付加して用いる並列プログラミング言語 OpenMP[4] を対象とした評価を行った．プログラムコード中のディレクティブで指定された部分を GPGPU 用言語である CUDA のプログラムコードに変換するトランスレータを実装し，行列積を計算するプログラムに適用した．評価の結果，学習コストの軽減しつつ高速な GPU 向け並列プログラムを作成できることを確認している．

実行環境に依存しない最適化を実現する研究として，行列計算ライブラリ ATLAS の実装が挙げられる[9]．ATLAS は性能への影響が大きいルーチンのパラメータを変化させコンパイル計算時間の計測を繰り返す．パラメータと計算時間の関係を得ることで，最適なパラメータでコンパイルされたライブラリを生成する．Whaley らは ATLAS の性能を様々なプロセッサ上で計測し，プロセッサベンダによる実装に劣らない性能が得られることを確認した．

3. PGI Accelerator

本研究で用いるディレクティブ型コンパイラである，PGI Accelerator[5] について述べる．PGI Accelerator は C または Fortran 言語で記述された逐次的なプログラムコードの一部を，CUDA のプログラムコードに変換する機能を持つコンパイラ製品である．

図 1 に示した関数 $f(i,j)$ は，逐次的な実行では $nj \times ni$ 回だけ順番に実行される．**!$acc region** および **!$acc end region** の 2 つのディレクティブを用いて並列化領域を指定すると，PGI Accelerator は領域に含まれるループの変数操作や命令実行などの処理を分析する．分析結果から並列化できると判断されると，関数 $f(i,j)$ の処理内容を CUDA のプログラムコードに変換し，領域の前後にデバイスメモリの確保やホスト-デバイス間のデータ転送処理といった GPU 実行に必要な処理を追加する．ループ間のデータ依存性が発見されるなどして並列化ができないと判断されたループは，通常のコンパイル処理が行われ CPU で逐次的に実行される．

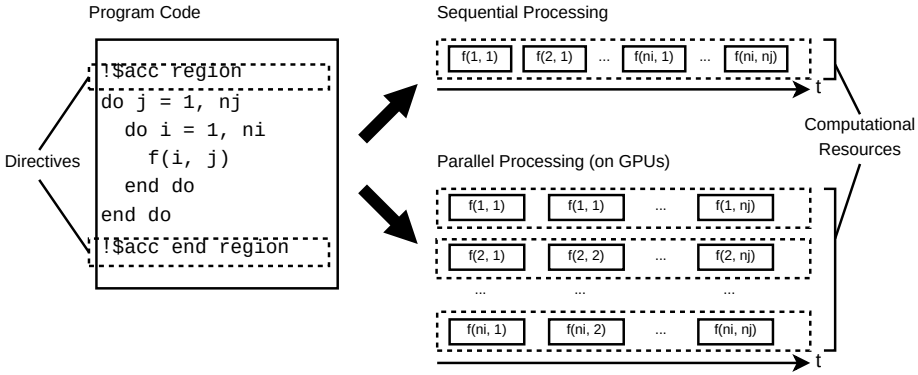

図 1 ループの並列化

GPU は内部に階層化された計算資源と独立した記憶領域を持つハードウェアである．GPU 用の並列プログラムでは計算資源に対する処理の割り当てが性能に大きな影響を与えることが報告されている[10]．また，ホスト-デバイス間のデータ転送の最適化が重要であることが知られている．PGI Accelerator はコンパイル時にデータと計算の適切な割り当てやデータ転送タイミングの決定を自動的に行うが，専用のディレクティブを用るとこれらのパラメータのチューニングを行うこともできる．

4. 並列化領域の最適化

プログラムコードの並列化により計算時間の短縮を図る際，プログラムコード中の並列化可能な部分を発見して並列プログラムに移植し，プロファイル情報を参考にして修正を行う作業を繰り返すことになる．この作業には，対象とするアーキテクチャやアルゴリズムの深い知識が要求される．アーキテクチャやアルゴリズムについての知識習得などの開発コストを低減するには，プログラムコードの分析による並列化可能な領域の抽出，各領域に対応する並列プログラムの作成，コンパイルとオブジェクトファイルの実行時間計測という単純な手順の反復によりプログラムコードの並列化を行う方法が考えられる．ディレクティブ型コンパイラを用いると，上記の手順のうち並列コードの作成をコメント行の付加のみで容易に行える．

Vol.2011-ARC-197 No.9
Vol.2011-HPC-132 No.9
2011/11/28

本研究では，ディレクティブ型コンパイラを用いて上記の手順を自動的に行う並列プログラムの最適化手法を提案する．この手法の利点としてプログラムコードの解析方法とディレクティブ付加内容の変更のみで，多種のアーキテクチャやプログラミング言語への対応が容易に行えることが挙げられる．その一方で，大規模なプログラムコードを対象とする場合に探索回数が非常に多くなるという欠点が存在する．

5. 実　　装

本研究では前章で述べた手法の評価を行うため，Fortran 言語で記述されたプログラムコードを対象とし，並列化領域指定ディレクティブの付加位置を最適化するトランスレータを実装した．バックエンドのディレクティブ型コンパイラとして PGI Accelerator を使用した．今回実装したシステムでは，プログラムコード内の全てのループを対象に，並列化領域の有無の組み合わせを網羅するように実行時間の測定を行った．並列化領域はループごとに設定し，領域の連結については考慮しない．

5.1 トランスレータの動作

システムは Fortran 言語のプログラムコードを入力として**図 2** に示すような，ループのネスト構造を表現する解析木を作成する．解析木をもとにループの前後にディレクティブを挿入したソースコードの生成を行いながら，PGI Accelerator コンパイラの呼び出しとオブジェクトファイルの実行時間計測を行う．

提案した最適化手法による探索回数はループ数に対応して指数関数的に増加する．しかし，PGI Accelerator の並列化領域指示ディレクティブはネストが認められていないため，対象コードのネスト構造によって有効な組み合わせの数が変化する．最適化の所要時間の目安とするため，解析木をもとにして並列化領域がネストする組み合わせを除いた試行回数を計算して表示する機能を実装した．解析木のノード $node_x$ の子ノードが持つ，ネストを許さない組み合わせの数は，次の式で計算することができる．

$$variation(node_x) = \begin{cases} \prod_{node_i \in Chidren_x} (1 + variation(node_i)) & (\mathrm{x} \neq \emptyset) \\ 1 & (\mathrm{x} = \emptyset) \end{cases} \quad (1)$$

6. 評　　価

6.1 評 価 方 法

実装したシステムについて，**表 1** に示す環境で，以下の 8 種類プログラムコードを用いて評価を行った．

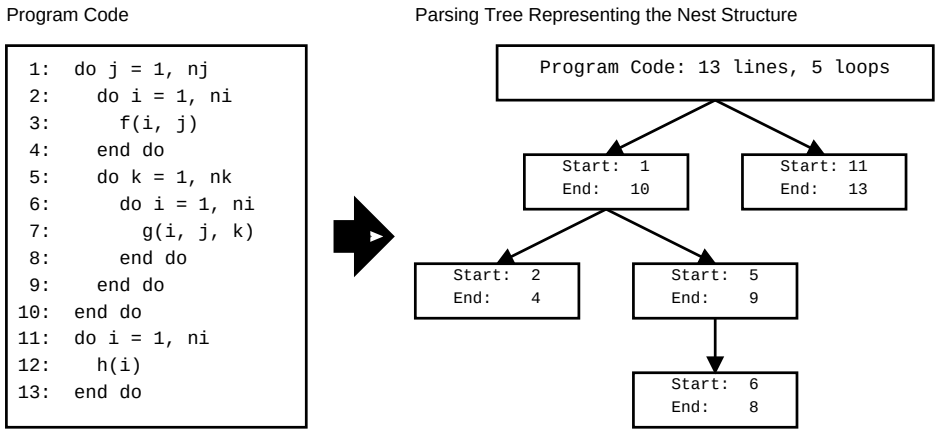

図 2 ネスト構造の解析木
Fig. 2 Parsing Tree of the Nest Structure

- NAS Parallel Benchmarks[11] 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 次方程式の解計算 (以下 runge[12])
- 512 × 512 ピクセルでのマンデルブロ集合の計算 (以下 mset[13])
- 1023 × 1023 次元行列と 1023 次元ベクトルの積 (以下 matvec[12])
- 1024 × 1024 次元行列の行列積の計算 (以下 matmul[13])
- 積分の計算 (以下 intgl4[12])
- 姫野ベンチマーク[14] Fortran90 M サイズ (以下 himeno)

表 1 評価環境
Table 1 Environment for Evaluation

	マシン 1	マシン 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)	
Option	-Minfo=accel,inline,ccff -fastsse -Minline=size:1000,levels:10,reshape -ta=nvidia,cuda3.1	

Vol.2011-ARC-197 No.9
Vol.2011-HPC-132 No.9
2011/11/28

- Large-Eddy Simulation による乱流モデル[15](以下 les)

プログラムコードの特性について**表 2** および**図 3** に掲載した．実行時間は 5 回の測定で得られた中央値を採用し，並列化領域を指定しない場合の実行時間に比べ 5 倍を超える場合は測定を打ち切った．プログラムの実行回数が動的に決定されるものは，実行回数を定数とした．

les はプロファイル情報に基づき，最も多くの実行時間を占有した sgs_model サブルーチンを対象に並列化を行った．当該サブルーチンの並列化領域の組み合わせは 54 億通りであったため，サブルーチン内で実行時に一度も用いられない部分を削除し現実的な評価回数での探索を可能にした．

6.2 評価結果

各プログラムコードの並列化領域パターンと実行時間の関係を，**図 4** から**図 11** に示す．各図の横軸は並列化領域の指定内容の違いを表し，CPU に対する速度向上率が高い順に並べて表示した．

matmul，intgl4，himeno，les の 4 つについて，本手法により CPU での実行速度に比べ 2 倍以上に高速化したパターンが見られた．

速度向上率が 1 付近となっているパターンが多いプログラムコードが見られた．これは以下の理由が考えられる．

- 並列化できないコードが含まれていた
- 並列化した部分は全体の実行時間に対して十分小さく影響を与えなかった
- CPU での実行時間と GPU での実行時間に大きな差がなかった

特に ep，runge では計算の中心となるループにループ伝搬依存の原因となるコードがあり，並列コードが生成されなかった．

表 2 評価に用いたプログラムコード
Table 2 Program Codes Used for Evaluation

bench	lines	loops	variations
ep	290	7	80
runge	97	1	2
mset	58	4	9
matvec	77	10	129
matmul	101	8	60
intgl4	57	4	5
himeno	326	5	10
les (sgs_model)	142	15	1536

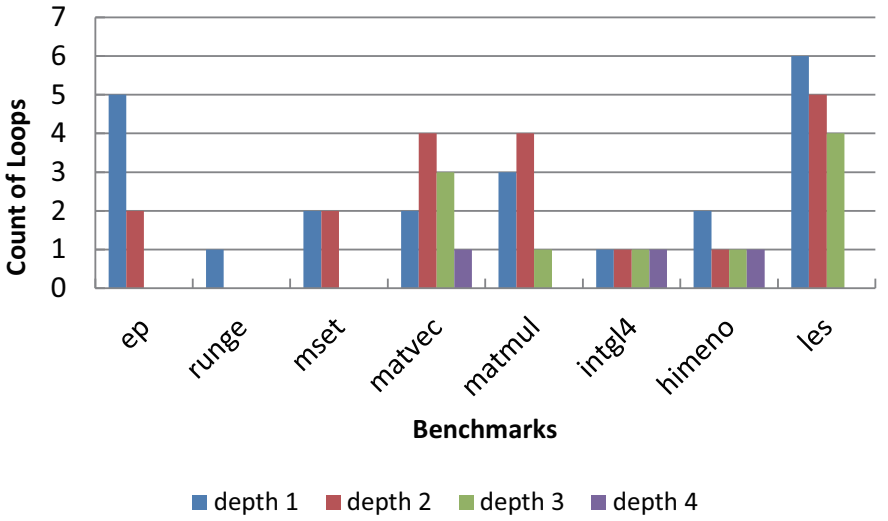

図 3 ネストの深さ別のループ数
Fig. 3 the Number of Loops on Each Nest Level

CPU での実行時よりも実行速度が低下した並列化領域のパターンが多数みられた．その原因として，多重ループの内側が並列化領域となり，データ転送やカーネル関数の呼び出しのオーバヘッドがループの反復回数分だけ発生したことが考えられる．また，GPU での計算処理よりもデバイス間のデータ転送時間が非常に大きいことも原因として考えられる．

matmul や matvec の図では高速化率が特定の値に集中する傾向が見られた．これはプログラムコード中に実行時間に大きな影響を及ぼすループが存在することが原因として考えられる．

7. 議　　論

実行速度が向上する並列化領域の指定方法は全体のうちのわずかであり，全ての組み合わせについて評価を行うとその大半が実行速度が低下するものとなっている．また，les については 1 つのサブルーチンを対象としていたが，実際に全ての組み合わせを探索すること

Vol.2011-ARC-197 No.9
Vol.2011-HPC-132 No.9
2011/11/28

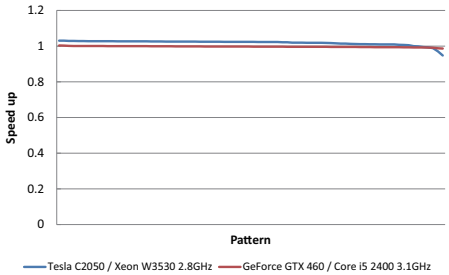

図 4 ep の並列化領域と実行時間の関係
Fig. 4 Execution Speed Comparision between Patterns of Regions, ep

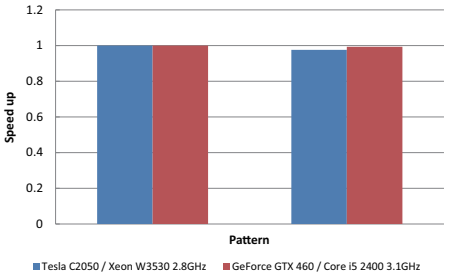

図 5 runge の並列化領域と実行時間の関係
Fig. 5 Execution Speed Comparision between Patterns of Regions, runge

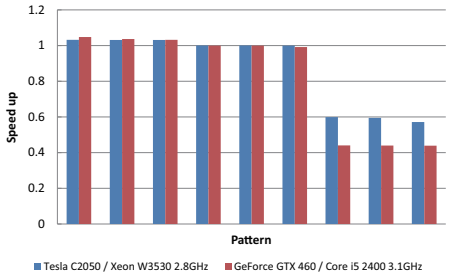

図 6 mset の並列化領域と実行時間の関係
Fig. 6 Execution Speed Comparision between Patterns of Regions, mset

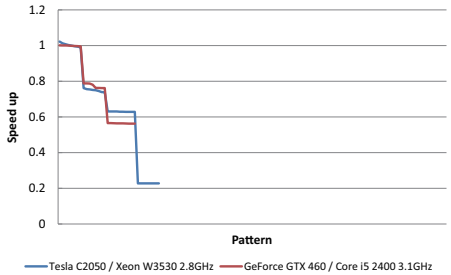

図 7 matvec の並列化領域と実行時間の関係
Fig. 7 Execution Speed Comparision between Patterns of Regions, matvec

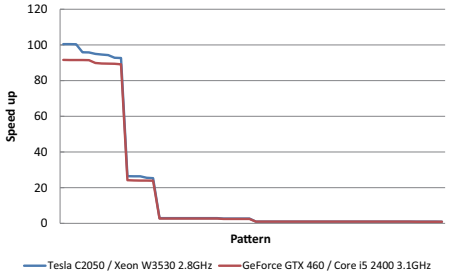

図 8 matmul の並列化領域と実行時間の関係
Fig. 8 Execution Speed Comparision between Patterns of Regions, matmul

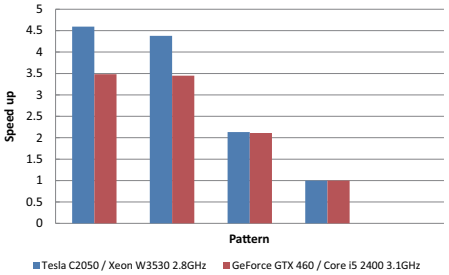

図 9 intgl4 の並列化領域と実行時間の関係
Fig. 9 Execution Speed Comparision between Patterns of Regions, intgl4

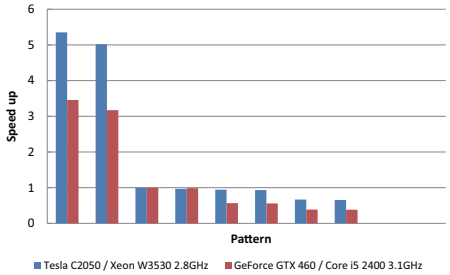

図 10 himeno の並列化領域と実行時間の関係
Fig. 10 Execution Speed Comparision between Patterns of Regions, himeno

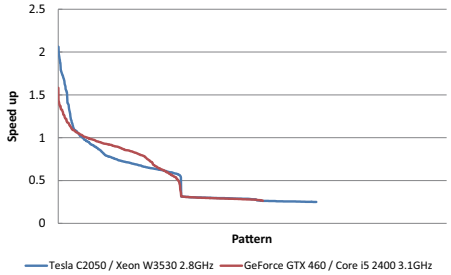

図 11 les の並列化領域と実行時間の関係
Fig. 11 Execution Speed Comparision between Patterns of Regions, les

は，試行回数の観点から困難である．並列化領域以外にもディレクティブで指定可能なパラメータが存在し，今後パラメータの種類を追加することも考えられる．以上のことから，特に大規模なプログラムコードに対して適用する場合に，探索空間の中から高速化効果が見込まれる並列化領域を効率的に得る方法を検討するべきである．探索範囲の削減を行う方法として，遺伝的アルゴリズム[16] を用いる方法が提案されている．

8. まとめと今後の展望

本研究では，ディレクティブ型コンパイラを用いて，実行時間を元にディレクティブの付加内容を最適化するプログラムコードの並列化手法を提案した．GPU 用ディレクティブ型コンパイラである PGI Accelerator をバックエンドに，Fortran プログラムに対するディレクティブ付加内容の最適化を行う手法を最適化するトランスレータを実装した．今回実装したトランスレータによる評価を通して，提案した並列化手法により高速化の効果を得られることを確認した．

今後大規模なプログラムに対して対応するために，並列化領域の候補を効率的に絞り込む方法について検討する．また，未使用のディレクティブを用いた最適化についても検証を行いたい．

Vol.2011-ARC-197 No.9
Vol.2011-HPC-132 No.9
2011/11/28

参 考 文 献

1) Matsuoka, S.: Making TSUBAME2.0, the world's greenest production supercomputer, even greener: challenges to the architects, *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, ISLPED '11, Piscataway, NJ, USA, IEEE Press, pp. 367–368 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2016802.2016887⟩ (2011).
2) NVIDIA: Compute Unified Device Architecture Programming Guide (2007).
3) Stone, J., Gohara, D. and Shi, G.: OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems, *Computing in Science Engineering*, Vol.12, No.3, pp.66 –73 (2010).
4) Dagum, L. and Menon, R.: OpenMP: an industry standard API for shared-memory programming, *Computational Science Engineering, IEEE*, Vol.5, No.1, pp.46 –55 (1998).
5) Wolfe, Michael: Implementing the PGI Accelerator model, *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, New York, NY, USA, ACM, pp.43–50 (2010).
6) Christian Terboven and Dieter an Mey: OpenMP in the Real World. http://cobweb.ecn.purdue.edu/ParaMount/iwomp2008/documents/OpenMP_in_the_Real_World (2011 年 10 月に確認).
7) WRF Model Users Site: . http://www.mmm.ucar.edu/wrf/users/ (2011 年 10 月に確認).
8) 大島聡史，平澤将一，本多弘樹：OMPCUDA：GPU 向け OpenMP の実装 (高性能計算)，情報処理学会研究報告. [ハイパフォーマンスコンピューティング]， Vol.2008, No.125, pp.121–126（オンライン），入手先⟨http://ci.nii.ac.jp/naid/110007123623/⟩ (2008-12-09).
9) Antoine, C.W., Petitet, A. and Dongarra, J.J.: Automated Empirical Optimization of Software and the ATLAS Project, *Parallel Computing*, Vol.27, p.2001 (2000).
10) 吉見真聡，廣安知之，三木光範：GPU プログラムの進化的計算によるパラメータチューニング手法の提案，先進的計算基盤システムシンポジウム SACSIS 2011 論文集， No.11-623, pp.229–230 (2011).
11) NAS Parallel Benchmarks: . http://www.nas.nasa.gov/Resources/Software/npb.html (2011 年 10 月に確認).
12) N. Tajima's fortran benchmark tests (Ver.2): . http://serv.apphy.fukui-u.ac.jp/˜tajima/bench/ (2011 年 10 月に確認).
13) Fortran Benchmarks (University of Western Ontario): . http://www.stats.uwo.ca/faculty/aim/epubs/benchmark/fortran.htm (2011 年 10 月に確認).
14) 姫野ベンチマーク：. http://accc.riken.jp/HPC/HimenoBMT.html (2011 年 10 月に確認).
15) Nakanishi, M.: Large-Eddy Simulation Of Radiation Fog, *Boundary-Layer Meteorology*, Vol.94, pp.461–493 (2000).
16) 戸松祐太，吉見真聡，廣安知之，三木光範：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，情報処理学会研究報告. 計算機アーキテクチャ研究会報告，Vol.2010, No.9, pp.1–6（オンライン），入手先⟨http://ci.nii.ac.jp/naid/110007997691/⟩ (2010-12-09).