計算機アーキテクチャ 144−12
（2001．7．26）

キャッシュ最適化を考慮したマルチプロセッサシステム上での粗粒度タスクスタティックスケジューリング手法

中野啓史[†] 石坂一久[†] 小幡元樹[†]
木村啓二[†] 笠原博徳[†,††]

早稲田大学理工学部電気電子情報工学科[†]
アドバンスト並列化コンパイラ研究体[††]
[†]〒169-8555 東京都新宿区大久保3-4-1 TEL:03-5286-3371
[††]http://www.apc.waseda.ac.jp/
E-mail: {hnakano,ishizaka,obata,kimura,kasahara}@oscar.elec.waseda.ac.jp

近年のプロセッサの動作速度とメモリアクセスの速度差の拡大により，データローカリティを利用したキャッシュ最適化がますます重要となっている．また，マルチプロセッサシステム上での並列処理においては，従来のループ並列化のみの並列処理は限界を迎えつつある．そのため更なる性能向上を得るには粗粒度タスク並列処理の併用等マルチグレイン並列化が重要となっている．本稿では，Fortranプログラムをループ・サブルーチン・基本ブロックの3種類の粗粒度タスクに分割し，粗粒度タスク間の制御依存・データ依存を解析して並列性を抽出する粗粒度タスク並列処理において，粗粒度タスク間のデータ共有量を考慮してキャッシュ最適化を行う粗粒度タスクスタティックスケジューリング手法について述べる．本手法をOSCAR Fortranマルチグレイン並列化コンパイラに実装してSun Ultra80(4プロセッサSMP)上で評価を行った結果，SPEC 95fpのswim, tomcatvにおいて，本手法により，Sun Forte HPC 6 update 1の自動並列化に対してそれぞれ4.56倍，2.37倍の速度向上が得られ，本手法の有効性が確かめられた．

A Static Scheduling Scheme for Coarse Grain Tasks considering Cache Optimization on SMP

Nakano Hirofumi[†], Ishizaka Kazuhisa[†], Obata Motoki[†], Kimura Keiji[†] and Hironori Kasahara[†,††]

Dept. of Electrical, Electronics and Computer Engineering, Waseda University[†]
Advanced Parallelizing Compiler Research Group[††]
[†]3-4-1 Ohkubo Shinjuku-ku, Tokyo 169-8555, Japan Tel: +81-3-5286-3371
[††]http://www.apc.waseda.ac.jp/
E-mail: {hnakano,ishizaka,obata,kimura,kasahara}@oscar.elec.waseda.ac.jp

Effective use of cache memory based on data locality is getting more important with increasing gap between the processor speed and memory access speed. As to parallel processing on multiprocessor systems, it seems to be difficult to achieve large performance improvement only with the conventional loop iteration level parallelism. This paper proposes a coarse grain task static scheduling scheme considering cache optimization. The proposed scheme is based on the macro data flow parallel processing that uses coarse grain task parallelism among tasks such as loop blocks, subroutines and basic blocks. It is implemented on OSCAR Fortran multigrain parallelizing compiler and evaluated on Sun Ultra80 four-processor SMP machine, using swim and tomcatv from the SPEC fp 95 benchmark suite. As the results, the proposed scheme gives us 4.56 times speedup for swim and 2.37 times for tomcatv respectively against the Sun Forte HPC 6 loop parallelizing compiler on 4 processors.

1 はじめに

プロセッサの動作速度とメモリアクセスの速度差は年々拡大している．そのため，プログラムの持つデータローカリティを利用したキャッシュ最適化が計算機の性能を引き出す上で重要となっている．コンパイラによるキャッシュ最適化として，複数のループリストラクチャリングを統合して行うAffine Partitioning [1)~3)]が挙げられる．データローカリティを利用する手法としてはループ分割後のタスクの垂直実行[4)]が提案されている．また，シングルプロセッサ上での粗粒度タスク間のローカリティを利用したキャッシュ最適化手法[5)]も提案されている．

一方，マルチプロセッサシステムにおいて従来から行われてきたループ並列化による並列処理は，長期間に渡るデータ依存解析，リストラクチャリング手法の開発により成熟期を迎え，今後劇的な性能向上は難しいと考えられている．そのためマルチプロセッサシステム上で更なる性能向上を得るためにはループ並列処理に加え，ループ・サブルーチン・基本ブロック間の並列性を利用する粗粒度タスク並列処理の併用等プログラム全域より並列性を引き出すマルチグレイン並列化が必要となってきている [6]．

本稿ではループ並列性も利用した粗粒度タスク並列処理 [7)~9)] におけるデータローカライゼーション手法 [10),11)] を利用した，キャッシュ最適化粗粒度タスクスタティックスケジューリング手法について述べる．本手法は OSCAR Fortran マルチグレイン並列化コンパイラ [12)] 上に実装され，逐次 Fortran を入力とし，キャッシュ最適化を考慮した粗粒度タスクスタティックスケジューリングを行った OpenMP Fortran を出力する．

本論文の構成は以下の通りである．第 2 章では OSCAR Fortran マルチグレイン並列化コンパイラ上での粗粒度タスク並列処理手法，第 3 章ではキャッシュ最適化を考慮したスケジューリングアルゴリズム，第 4 章では性能評価についてそれぞれ述べる．

2 粗粒度タスク並列処理

粗粒度タスク並列処理とは，ソースプログラムを疑似代入文ブロック (BPA)，繰り返しブロック (RB)，サブルーチンブロック (SB) の 3 種類のマクロタスク (MT) に分割し，そのマクロタスクを複数のプロセッサエレメント (PE) から構成されるプロセッサクラスタ (PC) に割り当てて実行することにより，マクロタスク間の並列性を利用する並列処理手法である．

OSCAR マルチグレイン並列化コンパイラにおける粗粒度タスク並列処理の手順は次のようになる．

1. ソースプログラムを階層的に 3 種類のマクロタスクに分割
2. 各階層のマクロタスク間のコントロールフロー，データ依存を解析しマクロフローグラフ (MFG) を生成
3. マクロフローグラフ上の制御依存とデータ依存を考慮してマクロタスク間の並列性を抽出する最早実行可能条件解析を行いマクロタスクグラフ (MTG) を生成
4. MTG がデータ依存エッジしか持たない場合は，マクロタスクはスタティックスケジューリング

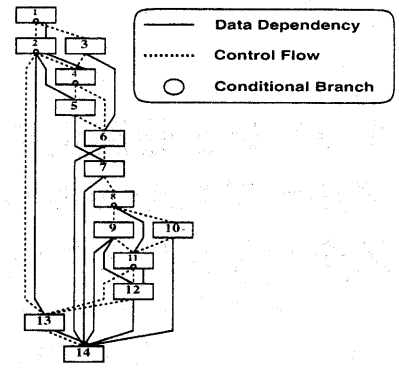

図 1: マクロフローグラフの例

によって PC または PE にコンパイル時に割り当てられる．一方，MTG が条件分岐などの実行時不確定性持つ場合は，コンパイラがユーザコード中に生成したダイナミックスケジューリングルーチンによって，マクロタスクを PC または PE に実行時に割り当てる．

2.1 マクロタスクの生成

粗粒度タスク並列処理では，まずソースプログラムを BPA，RB，SB の 3 種類のマクロタスクに分割する．

次に 3 章で述べるように，生成された RB がループイタレーションレベルの並列処理が可能な場合，その RB を PC 数やキャッシュサイズを考慮して異なる複数のマクロタスクに分割し，ループイタレーション間の並列性およびマクロタスク間でのデータローカリティを利用する．

ループ並列処理不可能な実行時間の大きい RB やインライン展開を効果的に適用できない SB に対しては，その内部を階層的に粗粒度タスクに分割して並列処理を行う．

2.2 マクロフローグラフ (MFG) の生成

マクロタスクの生成後，マクロタスク間のコントロールフローとデータ依存を解析し，その結果を表す図 1 に示すようなマクロフローグラフ (MFG) を生成する．

図 1 の各ノードはマクロタスクを表し，実線エッジはデータ依存を，点線エッジはコントロールフローを表す．また，ノード内の小円は条件分岐を表す．MFG ではエッジの矢印は省略されているが，エッジの方向は下向を仮定している．

2.3 マクロタスクグラフ (MTG) の生成

MFG はマクロタスク間のコントロールフローとデータ依存は表すが，並列性は表していない．並列性を抽出するためには，コントロールフローとデー

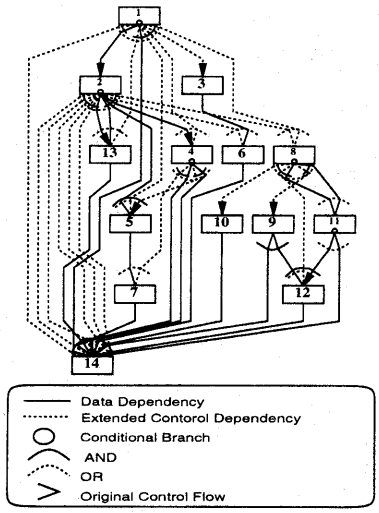

図 2: マクロタスクグラフの例

タ依存の両方を考慮した最早実行可能条件解析をマクロフローグラフに対して行う．マクロタスクの最早実行可能条件とは，そのマクロタスクが最も早い時点で実行可能になる条件である．

マクロタスクの最早実行可能条件は図 2 に示すようなマクロタスクグラフ (MTG) で表される．

MFG と同様に，MTG におけるノードはマクロタスクを表し，ノード内の小円はマクロタスク内の条件分岐を表している．実線のエッジはデータ依存を表し，点線のエッジは拡張されたコントロール依存を表す．拡張されたコントロール依存とは，通常のコントロール依存だけでなく，データ依存とコントロールフローを複合的に満足させるため先行ノードが実行されないことを確定する条件分岐を含んでいる．

また，エッジを束ねるアークには 2 つの種類がある．実線アークはアークによって束ねられたエッジが AND 関係にあることを，点線アークは束ねられたエッジが OR 関係にあることを示している．

MTG においてはエッジの矢印は省略されているが，下向きが想定されている．また，矢印を持つエッジはオリジナルのコントロールフローを表す．

2.4 スケジューリングコードの生成

粗粒度タスク並列処理では，生成されたマクロタスクはプロセッサクラスタ (PC) に割り当てられて実行される．PC にマクロタスクを割り当てるスケジューリング手法として，コンパイル時に割り当てを決めるスタティックスケジューリングと実行時に割り当てを決めるダイナミックスケジューリングがあり，マクロタスクグラフの形状，実行時不確定性などを元に選択される．

スタティックスケジューリングは，マクロタスクグラフがデータ依存エッジのみを持つ場合に適用され，コンパイラがコンパイル時にマクロタスクの PC への割り当てを決定する方式である．スタティックスケジューリングでは，実行時スケジューリングオーバーヘッドを無くし，データ転送と同期のオーバーヘッドを最小化することが可能である．

スタティックスケジューリングアルゴリズムの詳細については第 3 章で説明する．

3 キャッシュ最適化を考慮したスケジューリングアルゴリズム

ここでは粗粒度タスク間の並列性を利用したキャッシュ最適化を行うスタティックスケジューリングアルゴリズムについて述べる．

一般に，ある MT_i の実行終了時，MT_i 内で定義・参照されたデータはキャッシュ上に存在する可能性が高い．そこで，この MT_i と同じデータを定義・参照する量 (共有データ量) の多い MT_j を MT_i の直後に同一 PC 上で実行すればキャッシュを介したデータの授受が可能となる．本アルゴリズムの基本概念はこのようなスケジューリングを行うことにより，MT_j におけるメモリアクセスコストを減少させ，性能向上を図ろうとするものである．

3.1 マクロタスク分割

一つのマクロタスクで定義・参照するデータ量がキャッシュサイズを越えてしまう場合，その直後にデータ共有量の大きいマクロタスクを割り当てたとしてもデータがキャッシュから掃き出されてしまい，キャッシュを用いたデータの授受が行えない．そこで，このような場合マクロタスクの分割を行い，1 つのマクロタスクで定義・参照するデータ量をキャッシュに収まる程度に縮小化する．

マクロタスク分割の際，異なるプロセッサクラスタに割り当てられたマクロタスク間でのデータ転送を低減し，同一プロセッサクラスタに割り当てられたマクロタスク間でキャッシュを介したデータ授受を行うためには，各マクロタスクにおけるデータの使用範囲が等しくなるように，複数のループを整合して分割する必要がある．そこで，マクロタスク間のデータ共有量と並列性の両方を考慮する分割手法であるループ整合分割 [10),11)] を利用してマクロタスク分割を行う．

ループ整合分割はデータ依存エッジで接続された任意形状のマクロタスクグラフに対して適用される．

ループ整合分割の適用単位となるターゲットループグループ (TLG) は以下のように定義される．ターゲットループグループはマクロタスクグラフのクリティカルパス上の RB，及びその RB に直接先行接続あるいは直接後続接続された RB で構成される．これらの対象 RB は Doall ループ，リダクションループ，シーケンシャルループを仮定している．

次に TLG 毎にループ間におけるイタレーションに関するデータ依存を解析し，データの使用範囲が等しくなるように各ループを整合分割する．

3.2 スケジューリングアルゴリズム

ターゲットループグループに選択されたマクロタスクの内，グラフのクリティカルパス上のマクロタスクをループ整合分割してできたマクロタスクをデータローカライゼーショングループ (DLG) として定義する．同一データローカライゼーショングループ内のマクロタスクは同一プロセッサクラスタへ割り当てられる．

データを共有している二つのマクロタスクを異なるプロセッサクラスタへ割り当てた場合，それらのプロセッサクラスタ間でデータ転送が必要になる．データ転送を最小化するためには，あるプロセッサクラスタ PC_i に既に割り当てられたマクロタスク MT_j とデータ共有量の多いマクロタスク MT_k は並列性を損なわない範囲で同一のプロセッサクラスタ PC_i に割り当てる必要がある．そこで，あるプロセッサクラスタとマクロタスクの組合わせ毎に次のようにデータ転送ゲインを定義し，プロセッサクラスタ間でのデータ転送の最小化を行う．ここで，プロセッサクラスタ PC_i とマクロタスク MT_j のデータ転送ゲイン $Gain_{ij}$ は，PC_i に既に割り当てられたマクロタスクと MT_j とのデータ共有量，及び PC_i への割り当てが決まっているデータローカライゼーショングループに属するマクロタスクと MT_j とのデータ共有量の和と定義する．

本論文で評価するスケジューリングアルゴリズムである DLG を考慮したデータ転送ゲイン/CP/MISF スケジューリング法 [11] の概要を以下に述べる．ここで，DLG_MT とはデータローカライゼーショングループに属するマクロタスク，NOT_DLG_MT とは DLG_MT 以外のマクロタスクとする．

DLG_MT はそのプロセッサクラスタに最後に割り当てられた DLG_MT と同じデータローカライゼーショングループに属するマクロタスクを優先して割り当てる．NOT_DLG_MT はデータ転送ゲインの最も大きなマクロタスクとプロセッサクラスタの組合わせを優先して割り当てる．

4 性能評価

本章では性能評価について述べる．

4.1 OSCAR Fortran マルチグレイン並列化コンパイラ

本手法を実装した OSCAR Fortran マルチグレイン並列化コンパイラについて述べる．OSCAR Fortran マルチグレイン並列化コンパイラは図 3 に示すように，フロントエンド，ミドルパス，複数のバックエンドから構成される．

フロントエンドは Fortran77 のソースコードを読み込み，中間言語を生成する．

ミドルパスは，コントロールフロー解析，データ依存解析を行い，プログラムのリストラクチャリング，マクロタスクの生成および，並列性の自動抽出を行なう．さらに，キャッシュ最適化を考慮した粗粒度タスクのスタティックスケジューリングを行い，これらの解析結果に基づき並列化された中間言語を生成する．

OSCAR Fortran マルチグレイン並列化コンパイラは，OSCAR マルチプロセッサシステム，UltraSparc，富士通 VPP，STAMPI，OpenMP といった様々なターゲット用のバックエンドを持ち，ミドルパスが出力した並列化中間言語から，各ターゲット用のアセンブリコード，もしくは並列化 Fortran を生成する．今回は OpenMP バックエンドを使用し，キャッシュ最適化を考慮した粗粒度タスクスタティックスケジューリング手法を実現した．OpenMP を用いた粗粒度タスク並列処理の実現 [8),9)] では，スレッドの fork/join は 1 度しか行なわず，スレッド生成のオーバーヘッドを抑えている．本稿の評価では，キャッシュ最適化を考慮した生成された OpenMP Fortran プログラムを対象マシン上のネイティブコンパイラでコンパイルし，評価を行った．このように，OSCAR Fortran マルチグレイン並列化コンパイラを Fortran77 を入力とし，最適化された OpenMP Fortran を出力するプリプロセッサとして利用した．

4.2 評価環境

本節では本手法の評価に用いたマルチプロセッサシステム Sun Ultra80 とそのコンパイラ及びベンチマークについて述べる．性能評価に使用した 4 プロセッサの SMP Sun Ultra80 のスペック及び使用した Forte ループ自動並列化コンパイラの諸元を表 1 に示す．また，評価を行ったときの Forte コンパイラのコンパイルオプションを表 2 に示す．表中，

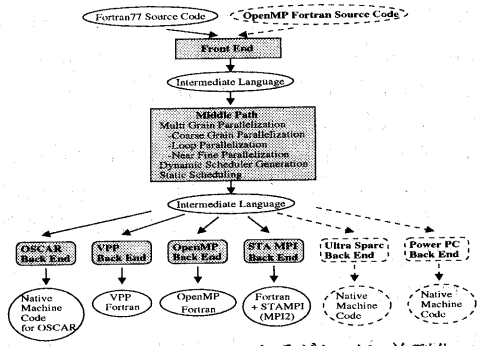

図 3: OSCAR Fortran マルチグレイン並列化コンパイラの構成

Forte とは Forte コンパイラのみによるコンパイルに用いたオプション，OSCAR とは本手法を適用後のプログラムのコンパイルに用いたオプションをそれぞれ意味する．評価ベンチマークには， SPEC 95fp の swim 及び tomcatv を用いた． swim は有限差分近似を用いた shallow water 方程式の求解プログラム，tomcatv はベクトル化メッシュ生成プログラムである．入力データとしては ref を用いた．

表 1: Ultra80 の仕様

Vender	Sun Microsystems
CPU	450MHz UltraSPARC-II 4 台からなる SMP
L1 命令 キャッシュ	16Kbyte Pseudo 2-Way Set Associative line size: 32byte
L1 データ キャッシュ	16Kbyte, Direct-Map line size: 32byte (two 16byte sub-blocks) write-through non-allocating
L2 ユニファイド キャッシュ	4Mbyte, Direct-Map line size: 64byte write-back, allocating
メインメモリ	1024Mbyte
OS	Solaris8
コンパイラ	Forte[tm] HPC 6 update 1

本手法の評価のために各プログラムのソースに以下のような改変を加えた．swim は CALC1, CALC2, CALC3 という三つのサブルーチンが実行時間の大半を占める．これらの異なるサブルーチン内部のループ間同士でデータ共有量が大きい．これら三つのサブルーチンをインライン展開後，フレキシブルクローニング [13] により一つのサブルーチンとし，新たにできたサブルーチンに対して本手法を適用した． tomcatv は実行時に使用配列サイズをファイルから読み込むが，スタティックスケジューリングによるキャッシュ最適化を考えているので，コンパイル時にデータサイズが確定している必要がある．

表 2: コンパイルオプション

	シングル プロセッサ用	マルチ プロセッサ用
Forte	-fast	-fast -parallel -reduction -stackvar
OSCAR		-fast -explicitpar -mp=openmp

そこで，ref データセットの使用配列サイズをプログラム中に明記した．また，データレイアウトの変換を行った．

4.3 評価結果

swim を用いた性能評価結果を図 4 に， tomcatv を用いた性能評価結果を図 5 にそれぞれ示す．なお，プロセッサ台数が 4 台と少ないので 1 プロセッサエレメントを 1 プロセッサクラスタとした．図中，速度向上率とは “ Forte のみでコンパイルしたプログラムの 1 プロセッサでの実行時間/本手法を適用したプログラムの実行時間 × 100 [%]” である．計測はそれぞれ 5 回ずつ行い，最速値を計測値とした．

図 4 において，本手法を適用したプログラムの実行時間は 1 プロセッサ時 81.3 秒，2 プロセッサ時 47.9 秒，3 プロセッサ時 27.8 秒，4 プロセッサ時 13.2 秒となっている．2, 3, 4 プロセッサ時の実行時間は 1 プロセッサ時の実行時間に対し，それぞれ 1.70 倍，2.92 倍，6.16 倍とプロセッサ台数に対しスケーラブルな速度向上を示している．また，4 プロセッサ時の性能評価結果はスーパーリニアとなっている．まず， swim で使われている配列の総サイズは約 13MB である．一方，4 プロセッサ時の L2 キャッシュの総容量は 16MB となる．このため，配列がほとんど L2 キャッシュに収まり，キャッシュの有効利用とそれに伴うパイプラインストールの減少により，このような結果が得られたと考えられる．

図 5 において，本手法を適用したプログラムの実行時間は 1 プロセッサ時 101.1 秒，2 プロセッサ時 64.0 秒，3 プロセッサ時 50.9 秒，4 プロセッサ時 33.1 秒となっている．2, 3, 4 プロセッサ時の実行時間は 1 プロセッサ時の実行時間に対し，それぞれ 1.58 倍，1.98 倍，3.05 倍とプロセッサ台数に対しスケーラブルな速度向上を示している．

次に本手法を適用したプログラムの実行時間と Forte のみでコンパイルしたプログラムの実行時間

を比較する．Forte のみの場合プロセッサ台数に対して速度向上があまり見られないのに対して，本手法はスケーラブルな速度向上を示している．また，いずれのベンチマーク，いずれのプロセッサ台数に対しても本手法は Forte よりも高い速度向上を示している．図 4 において，4 プロセッサ時，Forte のみでコンパイルしたプログラムの実行時間が 60.2 秒であるのに対し，本手法を適用したプログラムの実行時間は 13.2 秒となり，4.56 倍の速度向上を得た．また，図 5 において，4 プロセッサ時，Forte のみでコンパイルしたプログラムの実行時間が 78.6 秒であるのに対し，本手法を適用したプログラムの実行時間は 33.1 秒となり，2.37 倍の速度向上を得た．

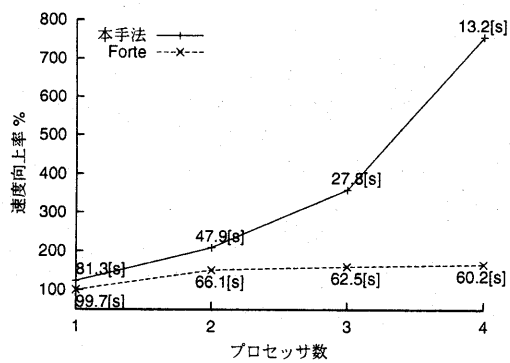

図 4: swim の速度向上率

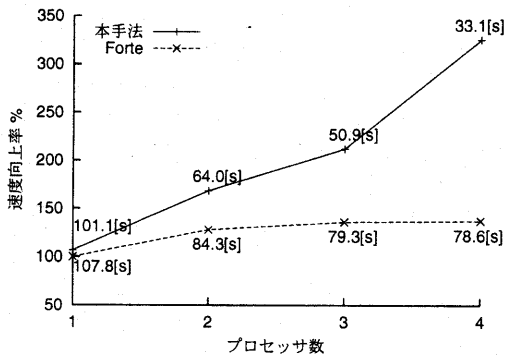

図 5: tomcatv の速度向上率

5 まとめ

本稿では，キャッシュ最適化を考慮したマルチプロセッサシステム上での粗粒度タスクスタティックスケジューリング手法について述べた．本手法は OSCAR Fortran マルチグレイン並列化コンパイラ上に実装され，逐次 Fortran を入力とし，最適化された OpenMP Fortran を出力する．性能評価の結果，本手法を適用したプログラムを Sun Ultra80 上，4 プロセッサ で実行したとき，シーケンシャル実行時間に対し，swim で 6.16 倍 (Forte 自動並列化の 4.56 倍)，tomcatv で 3.05 倍 (Forte 自動並列化の 2.37 倍) の速度向上が得られ，本手法の有効性が確認された．

本研究の一部は NEDO アドバンスト並列化コンパイラプロジェクト及び STARC "コンパイラ協調型シングルチップマルチプロセッサの研究開発" により行われた．

参考文献

[1] Lim, A. W., Chèong, G. I. and Lam, M. S.: An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication, *Proc. 13th ACM SIGARCH International Conference on Supercomputing* (1999).

[2] Lim, A. W., Liao, S. and Lam, M. S.: Blocking and Array Contraction Across Arbitrarily Nested Loops Using Affine Partitioning, *Proc. of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2001).

[3] Lim, A. W. and Lam, M. S.: Cache Optimizations With Affine Partitioning, *Proc. of the Tenth SIAM Conference on Parallel Processing for Scientific Computing* (2001).

[4] Vajracharya, S., Karmesin, S., Beckman, P., Crotinger, J., Malony, A., Shende, S., Oldehoeft, R. and Smith, S.: SMARTS: exploiting temporal locality and parallelism through vertical execution, *Proc. of the 1999 international conference on Supercomputing* (1999).

[5] 稲石, 木村, 藤本, 尾形, 岡本, 笠原: 最早実行可能条件を用いたキャッシュ利用の最適 化, 情報処理学会研究報告 ARC (1998).

[6] APC: http://www.apc.waseda.ac.jp/.

[7] 岡本, 合田, 宮沢, 本多, 笠原: OSCAR マルチグレインコンパイラにおける階層型マクロデータフロー処理, 情報処理学会論文誌, Vol. 35, No. 4, pp. 513–521 (1994).

[8] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. 12th Workshop on Languages and Compilers for Parallel Computing* (2000).

[9] 石坂, 八木, 小幡, 吉田, 笠原: 共有メモリマルチプロセッサシステム上での粗粒度タスク並列実現手法の評価, 情報処理学会研究報告 ARC (2001).

[10] 吉田, 越塚, 岡本, 笠原: 階層型粗粒度並列処理における同一階層内ループ間データローカライゼーション手法, 情報処理学会論文誌, Vol. 40, No. 5, pp. 2054–2063 (1999).

[11] 吉田, 八木, 笠原: SMP 上でのデータ依存マクロタスクグラフのデータローカライゼーション手法, 情報処理学会研究報告 2001-ARC-141 (2001).

[12] 笠原: 並列処理技術, コロナ社 (1991).

[13] 吉井, 松井, 小幡, 熊澤, 笠原: マルチグレイン自動並列化のための解析時インライニング, 情報処理学会研究報告 ARC/HPC (2000).