社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－ARC－155 (9)
2003／11／28

SMPマシン上での粗粒度タスク並列処理におけるデータプリフェッチ手法

宮本孝道[†] 山口高弘[†††] 飛田高雄[††]
石坂一久[†] 木村啓二[†] 笠原博徳[†]

現在多くのサーバで使用されている主記憶共有型マルチプロセッサシステム (SMP) では，プロセッサの動作速度向上と共に，メモリアクセスオーバヘッドの増大が，プロセッサ毎にスケーラブルな性能向上を得るための大きな障壁となっている．本論文では，このメモリアクセスオーバヘッドを軽減しスケーラブルな性能向上を得るために，粗粒度タスクのデータローカライゼーション (データ分割) によっても取り除けなかったデータ転送をプリフェッチによりタスク処理とオーバラップさせることにより軽減させるスタティックスケジューリングを提案し，その性能を評価する．提案するアルゴリズムは，コンパイル時のスタティックスケジューリングを前提とし，今回評価に使用した v880 用のプリフェッチディレクティブを挿入した OpenMP 並列化 Fortran を出力するものである．性能評価の結果，Sun Forte コンパイラの逐次処理プリフェッチなしの場合と比較すると，SPEC95fp の tomcatv では 8 プロセッサで最大 13.9 倍，swim では 8 プロセッサで最大 22.3 倍の速度向上を得るなど，スーパーリニアスピードアップが効率良く引き出せるだけでなく，Sun Forte コンパイラによる自動プリフェッチ命令挿入を用い同一プロセッサ台数で処理する場合どうしを比較しても tomcatv では 1 プロセッサで 1.11 倍，8 プロセッサで 3.86 倍，swim で 1 プロセッサで 1.44 倍，8 プロセッサで 1.85 倍の速度向上が得られ，本手法の有効性が確認された．

The Data Prefetching of Coarse Grain Task Parallel Processing on Symmetric Multi Processor Machine

Takamichi Miyamoto,[†] Takahiro Yamaguchi,[†††]
Takao Tobita,[††] Kazuhisa Ishizaka,[†] Keiji Kimura[†]
and Hironori Kasahara[†]

On the shared multi processor system used in current computing servers, the increase of memory access overhead with the speedup of CPU interfere to get the scalable performance improvement with the increase of the processors. In order to get scalable performance improvement, this paper proposes and evaluates the static scheduling algorithm which reduces the memory access overhead by using cache prefetch to overlap of data transfer and task processing. The proposed algorithm is used in static scheduling stage in a compiler, moreover the compiler generates a OpenMP parallelized Fortran program with prefetch directives for SUN Forte compiler for Sun Fire V880 server. Performance evaluation shows that the proposed algorithm gave us super linear speedup compared with sequential processing without prefetching by Sun Forte compiler such as 13.9 times speedup on 8 processors for SPEC95fp tomcatv program and 22.3 times speedup on 8 processors for SPEC95fp swim program. Futhermore, compared with automatic prefetching by SUN Forte compiler using the same number of processors, this algorithm shows that 1.1 times speedup on 1 processor, 3.86 times speedup on 8 processors for SPEC95fp tomcatv and 1.44 times speedup on 1processor, 1.85 times speedup on 8 processors for SPEC95fp swim.

1. はじめに

共有メモリ型マルチプロセッサシステムにおいて，効率の良い並列処理を行うためには，効果的な並列性の抽出，データ転送オーバヘッドを考慮したタスクのプロセッサへのスケジューリング，データの分割・配置を伴うキャッシュメモリ再利用，これらの最適化を実現する並列化コード生成が重要である．我々はこれらの問題の解決を目指し，ミレニアムプロジェクト IT 21 アドバンスト並列化コンパイラプロジェクトにおいて，マルチグレイン並列性抽出，データローカライゼーション，タスクスケジューリング，OpenMp 並列化コード生成等を行う自動並列化コンパイラを開発した[10),11)]．

このコンパイラでは粗粒度タスク並列処理にデータ依存を持つ複数のループ間でグローバルなキャッシュの有効利用を図るため，ループを分割し，キャッシュ上にある配列データに依存するループ部分を連続して実行するデータローカライゼーション[12)] と，その最適化効果をさらに引き出すためのパディングを用いてデータレイアウト最適化にあり，年々深刻になるメモリバリアの問題に対処している．さらにこのメモリアクセスオーバヘッドに対応するため，最近のプロセッサではプリフェッチキャッシュ

† 早稲田大学
Waseda University
†† ソニー
SONY Corporation
††† 日本放送協会
Japan Broadcasting Corporation

あるいは，データキャッシュに対するデータプリフェッチ機能を持っているものが増え，この機能を最適に利用することが次世代のコンパイラに望まれている．

プリフェッチとは，演算などの命令実行と並行して，後の計算処理に必要なデータを含むキャッシュラインのデータを主メモリからキャッシュメモリなどCPUに近いメモリに先行的に転送することで，大きく分けるとソフトウエアで制御するソフトウエアプリフェッチ[1]とハードウエアベースのハードウエアプリフェッチ[2]の2つの手法がある．

ソフトウェアプリフェッチはスタンフォード大学のSUIFコンパイラ[3]に実装されている．これはループ内の配列データの依存を調べるデータローカリティ解析を用いてキャッシュミスを起こしそうな配列参照を予測し，プリフェッチを行う対象ループを分割し，ループ回転数を減らすためにループを展開(アンローリング)したのち，計算されたベストタイミングでプリフェッチすることで，無差別にプリフェッチ命令を挿入するアルゴリズムに対してプリフェッチ命令の削減に成功し，プリフェッチ有りの場合との比較は，プリフェッチなしの場合と比較して最高で約2倍の速度向上を得ている．なお同様なソフトウェアプリフェッチはSun Forteコンパイラでも実装されている．

またIBM Power4アーキテクチャ[4]はハードウェアプリフェッチをサポートしている．キャッシュミスのパターンを見て，キャッシュミスが連続したキャッシュラインで起きたときに，ハードウェアでプリフェッチをし，その後に使われる要素をキャッシュに持ってくることでキャッシュミスを減らしている．Sun Ultra SPARC IIIアーキテクチャ[5],[6]ではプリフェッチキャッシュというプリフェッチ専用のキャッシュが設けられており，ハードウェアプリフェッチも用意されている．

本稿では，プリフェッチキャッシュを持ちソフトウェアプリフェッチとハードウェアプロフェッチの両方が実現されているUltra SPARC III上で，従来のループ内での後続イタレーションでアクセスされるデータのプリフェッチのみならず，データローカライゼーション技術によりキャッシュサイズに適合するように分割された複数ループ間でのグローバルデータプリフェッチを実現するスタティックスケジューリングアルゴリズムを提案する．また，このアルゴリズムをSPEC95fpベンチマーク中のtomcatv, swimの各プログラムに適用してSun Fire V880上で性能評価を行い，提案するキャッシュプリフェッチを考慮したスケジューリング手法が有効であることを示す．

本稿の構成を以下に示す．第2章では，対象とする問題やアルゴリズム実現のための処理技術などについて述べる．第3章では，キャッシュ内のデータ配置シミュレーションを行いながら，データ転送オーバーヘッドの最小化を試み，プリフェッチディレクティブ挿入タイミングを決定するスケジューリングアルゴリズムを提案する．第4章では，提案するアルゴリズムの性能評価について述べ，最後に第5章で本稿のまとめを述べる．

2. データプリフェッチ

本章では，本稿で対象とするデータプリフェッチスケジューリング問題，対象アーキテクチャとともに，本研究で使用したOSCARコンパイラと，そのOSCARコンパイラに実装されているスケジューリングに必要な要素技術について述べる．

2.1 データプリフェッチスケジューリング問題

本稿では，逐次Fortranソースコードを入力とし，粗粒度タスク並列処理及びループ並列化，及びデータローカライゼーションを行うコンパイル作業において，条件分岐を含まないプログラム部分(タスク)のコンパイルにおいてデータプリフェッチディレクティブを挿入するスタティックスケジューリング問題を扱う．以下では，各タスクの処理コストはFortranソースコードの各ステートメントの演算命令クロック数をもとにコンパイラが推定し，データ転送コストは各タスクで定義参照されるデータ量(バイト数)を用い推定する．

2.2 対象アーキテクチャ

本研究で対象とするアーキテクチャは，キャッシュとデータプリフェッチ命令を持つマルチプロセッサシステムであり，本論文ではSun Fire V880(以下V880)を用いた．このシステムは8台のUltra SPARC IIIプロセッサ(750MHz)と，そのそれぞれのプロセッサごとに2KBのプリフェッチキャッシュ(ラインサイズ64バイト)，8MBのL2キャッシュ(ラインサイズ512バイト)を持つ．

V880上のコンパイラとしてはForte6 update2を用いた．このForteコンパイラはプリフェッチ命令を自動挿入する機能を持っているが，本研究では提案するスケジューリングアルゴリズムによりプリフェッチディレクティブをFortranソースコード中に挿入することでプリフェッチを実現する．使用できるプリフェッチディレクティブとその動作は以下の通りである．

- SPARC_PREFETCH_READ_MANY

指定されたデータは主メモリからL2キャッシュ，あるいはL2キャッシュからプリフェッチキャッシュへプリフェッチされる．このディレクティブによるメモリリクエスト時にはRTS(Read to Share)が発行される．

- SPARC_PREFETCH_READ_ONCE

指定されたデータは主メモリあるいはL2キャッシュからプリフェッチキャッシュへプリフェッチされる．このディレクティブによるメモリリクエスト時にはRTSが発行される．

- SPARC_PREFETCH_WRITE_MANY

指定されたデータは主メモリからL2キャッシュへプリフェッチされる．このディレクティブによるメモリリクエスト時にはRTO(Read to Own)が発行される．

- SPARC_PREFETCH_WRITE_ONCE

指定されたデータは主メモリからL2キャッシュへプリフェッチされる．このディレクティブによるメモリリクエスト時にはRTSが発行される．

ここで，RTS命令はプリフェッチするデータがShare状態であることを，RTO命令はOwn状態であることを明確にするための命令である．

2.3 OSCARコンパイラ

本研究では，逐次Fortranソースコードにプリフェッチ命令を自動的に挿入するためにOSCARマルチグレイン並列化コンパイラ[7],[11]を用いる．OSCARコンパイラでは，Fortranソースからマクロタスク(MT)を生成し，MT間のコントロール依存，データ依存を解析し，マクロフローグラフを作成する．そして各MTの最早実行可能条件を解析し，マクロタスクグラフを生成し，タスクグラフがデータ依存のみならば，スタティックスケジューリングによってMTは各プロセッサに割り当てられる．

本研究で提案するスケジューリングアルゴリズムはこのスタティックスケジューリング対象MTに対しそのタスク実行前に，その先行タスクの実行とオーバラップしプリフェッチが可能となるようにプリフェッチディレクティブを挿入するものである．今回の評価ではSun Forteコンパイラのプリフェッチディレクティブの形で生成されFortranコードに埋め込まれた形で出力される．

本研究では，OSCARコンパイラの持つ多種の解析・

コード変換技術のうち, 効率的にメモリを使用するための配列次元入れ替え, 配列の最速変化次元 (Fortran においては一次元目) に置かれた添字をループインデックスとして持つループが一番内側となるようにするループインターチェンジのほか, データローカライゼーション, パディング, ループアンローリング技術を用いる. 以下ではこれらの技術について述べる.

2.3.1 データローカライゼーション

キャッシュサイズと比較して大きい共有データにアクセスするループの間 (マクロタスク上のデータ依存エッジで結ばれている RB 間) では, 後続ループ実行時に必要なデータが追い出されるためにキャッシュを効率よく使用することができない. そこで, 図 1 のように, ループで使用されるデータ量がキャッシュサイズにおさまるようにデータ依存を持つ複数のループを整合分割する[8),12)]. 整合分割とは, 複数のループ間のデータ依存を考慮し, ループ及びデータの分割後プロセッサ間でのデータ転送が最小となるよう複数のループを整合して分割する方式である. また整合分割された部分ループを各々粗粒度タスクと定義し同一のデータ (部分配列) にアクセスするタスク集合をデータローカライゼーショングループ (以下 DLG)[8)] と呼び, スケジューラにより同一プロセッサに割当てられる[12)].

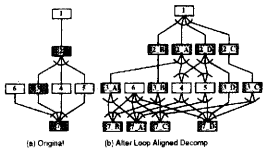

図 1 データローカライゼーショングループ
Fig. 1 Data localization groups

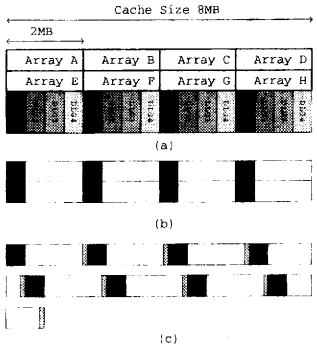

図 2 パディング
Fig. 2 Padding

2.3.2 データローカライゼーションのためのパディング

本論文で用いるパディング手法[13)] を図 2(a) のように, 8MB のダイレクトマップのキャッシュとサイズ 2MB の配列が 8 個ある場合を例にとり説明する. 図 2 は, その時のキャッシュ上での各配列の配置を示したもので, ここでは, 水平方向に配列データが確保されるものとして, 先頭アドレスからキャッシュサイズを超えると 1 段下げ, 垂直方向がそろう箇所はキャッシュメモリ上で同一の位置に割当てられることを示す. この場合, 図 2(b) のように配列 A と E, B と F, C と G, D と H がそれぞれ同一のキャッシュ領域に割当てられる, つまり DLG 中のタスク集合でアクセスされるデータは整合分割によりキャッシュサイズより小さくなるように分割されているにもかかわらず同一 DLG(図 2(a) における各部分) 内でアクセスされるデータ同士がラインコンフリクトを起こし, 同一 DLG 内タスクが同一プロセッサで優先的に実行されるにもかかわらずキャッシュを有効に使用できない.

そこで, 配置が図 2(c) のような関係になるように, 各 DLG でアクセスされるデータのキャッシュ上での割付け位置を配列宣言サイズを拡張する方式のパディングによりコンフリクトが起きないようにずらすことにより同一 DLG 内でのラインコンフリクトミスを削減する[9),13)].

2.3.3 ループアンローリング

本論文で対象とする Ultra SPARC III アーキテクチャの持つプリフェッチディレクティブは, 1 つの変数をプリフェッチすると, その変数の格納されるキャッシュライン上のデータがプリフェッチされる. このため, ある配列 A の 1 要素が 64 ビット (8 バイト) で, これをプリフェッチキャッシュ(ラインサイズ 64 バイト) にプリフェッチする場合は, 内側ループを 8 回アンローリングし, ディレクティブを挿入する.

3. プリフェッチスケジューリングアルゴリズム

本章では, 提案するキャッシュプリフェッチスケジューリングアルゴリズム DLG/DT-Gain/CP/MISF with Prefetch について述べる.

DLG/DT-Gain/CP/MISF with Prefetch アルゴリズムでは, プリフェッチ命令によりデータを格納できるキャッシュのサイズと MT で利用するデータ情報から, キャッシュ上に存在するデータの推定を行う. この際, キャッシュ容量を超えるデータを扱う場合は, LRU(Least Recently Used) によるラインリプレースを仮定する. また, キャッシュ内のデータと MT でアクセスされるデータのうち同一なデータの量を “データ共有量” と定義し, MT でアクセスされるデータのうちキャッシュ内に存在せずプリフェッチ命令によりキャッシュへプリフェッチするデータを “プリフェッチ量” と定義する.

提案する DLG/DT-Gain/CP/MISF with Prefetch アルゴリズムの手順を以下に示す.

1. スケジューリング前にコンパイラ中の解析モジュールによって各 MT で使用される配列の名前, 添字範囲, データサイズの情報を取得する.
2. 出口ノードからのパス長 (CP 長) の大きい順に各 MT に優先順位をつける.
3. ある時点で各実行可能 MT を各プロセッサに割り当てたときのデータ共有量, プリフェッチ量を計算する.
4. そのプロセッサに最後に割り当てられた MT と同じ DLG に属する MT があればそれを割り当てて手順 6 へ. 候補が複数ある, あるいは同一 DLG に属するタスクがなければ手順 5 へ.
5. データ共有量が最大の MT とプロセッサの組み合わせを選ぶ. 候補が複数ある場合, プリフェッチ量が最大の組み合わせを優先, それでも決定できないときは手順 2 の優先順位に従う. 手順 2 でも決定できないときは, 後続 MT の最も多い MT を優先する.
6. キャッシュシミュレーション, プリフェッチ対象配列を決定する.
7. 手順 3 から 6 を全 MT が割り当てられるまで繰り返す.

提案するスケジューリングアルゴリズムでは図 3(a) の

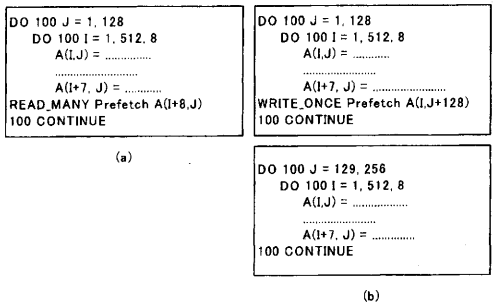

図 3 プリフェッチディレクティブの使用例
Fig. 3 Examples of using prefetch direcrtives

ように同一ループで何イタレーションか後に使用するデータをプリフェッチする方法と, 図 3(b) のように前の異なるループの実行とオーバラップし後のループで使用するデータをプリフェッチする方法を併用する. ここでは同一ループ内へのプリフェッチのため READ_MANY ディレクティブを使用する. また, 本研究で新たに提案する複数ループ間でのプリフェッチでは, 異なる DLG 間で後続の MT(ループ) で使用するデータをプリフェッチをも可能としている. 例えば図 4(b) の Loop2-1 でアクセスするデータを Loop1-1 実行中にプリフェッチする. これはプリフェッチキャッシュへプリフェッチしてもプリフェッチキャッシュのサイズが小さく使用されるまでに残っている可能性は低いだけでなく, 現在の CPU での計算に用いられているデータや同一ループでプリフェッチされるデータと競合し, 悪影響を与えることが考えられるため, サイズの大きい L2 キャッシュへのみプリフェッチするように WRITE_ONCE ディレクティブを使用する.

以下では, 実際に図 4(a) の MT をスケジューリングする例について説明する. ここで図 4(a) の MT 間の直線は先行制約を表し, 数字は MT 番号, MT を囲んでいる枠は DLG を表す. MT1, 2 と MT3, 4 は同じループをそれぞれ 2 分割したものである. MT1, MT3 の Fortran ソースコード例を図 4(b) に示す.

またこの例においては, キャッシュ容量は 4MB, 各 MT で使用されるデータは配列 1 個につき, 1MB として, 1PE(Processor Element) で逐次処理する際のスケジュールを求めることを考える. まず MT1 がプロセッサに割り当てられたものとするとき, キャッシュの推定データ割当て状態は, 図 5(a) となる. MT1 割り当て後, 実行可能となるのは, MT2, 3 である. ここで MT1 と同一 DLG に属するタスクは MT3 であるので, MT3 を優先して割り当てることを考え, 図 5(b) の状態になると推定する.

次に実行可能であるのは, MT2 である. MT3 と同一 DLG に属するタスクがないので, 図 5(b) のキャッシュ状態から考えて, データ共有量の大きい MT2 を割り当てる. ここでプリフェッチ判定を行う. ここでは MT2 で使用される A1(1:512,258:513), A2(1:512,257:512) のデータを分割前に同じループであった MT1 でプリフェッチすることを決定する. キャッシュの内容は, MT2 のデータをそのまま入れると, 容量を超えてしまうため, LRU アルゴリズムにより, A1(1:512,1:257) のデータを追い出されると推定してから, MT2 のデータを入れて図 5(c) のようになるとスケジューラは予測する.

このようにスケジューリングを続けていくと, 実行順は MT1 → 3 → 2 → 4 → 5 になる. このような順に実行することで, 元のプログラム (図 4(b)) では配列 A1, A2 だけでキャッシュが占められ, A3 をプリフェッチすることができないのに対し, 提案アルゴリズムではプリフェッチを効果的に実現できることがわかる.

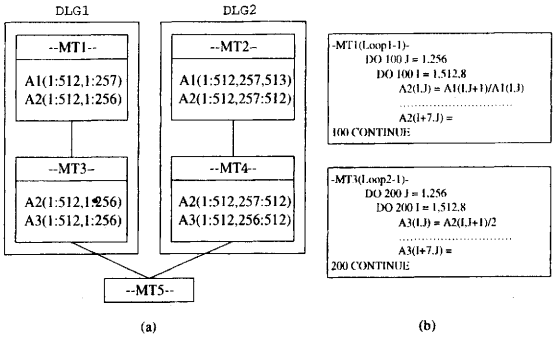

図 4 対象タスクグラフ
Fig. 4 Target task graph

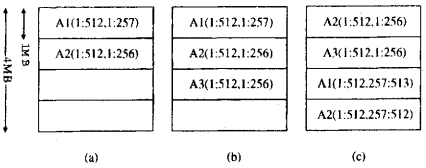

図 5 キャッシュの状態
Fig. 5 State of cache

4. 性能評価

本章では, 第 3 章で述べたスケジューリングアルゴリズム DLG/DT-Gain/CP/MISF with Prefetch の評価について述べる.

4.1 評価環境

本論文では, プリフェッチ命令を持つ CPU の 1 つである Ultra SPARC III を搭載した 8 プロセッサ SMP サーバ Sun Fire V880(以下 V880) を用いる. V880 は 750 MHz の Ultra SPARC III (ただし, Sun Microsystems は UltraSPARC III 750MHz のプリフェッチキャッシュをオンにして使用する構成は正式にはサポートしていない) 8 台のそれぞれに, 32KB の L1 命令キャッシュ(4 ウェイセットアソシアティブ, ラインサイズ 32 バイト), 64KB の L1 データキャッシュ(4 ウェイセットアソシアティブ, ラインサイズ 32 バイト), 2KB のプリフェッチキャッシュ (4 ウェイセットアソシアティブ, ラインサイズ 64 バイト), ならびに 8MB の L2 キャッシュ(2 ウェイセットアソシアティブ, ラインサイズ 512 バイト) を持ち, 32GB の共有メモリを持つ主記憶共有型マルチプロセッササーバである.

また, コンパイラとしては, Sun Forte6 update2 の f95 を用いた. 使用オプションを表 1 に示す. 表 1 のように, Forte コンパイラによる自動プリフェッチ命令挿入を使用するときには, "-xprefetch=auto" を用い, 提案するアルゴリズムで挿入したプリフェッチディレクティ

表 1 各条件のコンパイルオプション
Table 1 Compiler options for each condition

Forte 逐次	-fast -xarch=v8plusb -xprefetch=
Forte 並列	-fast -autopar -stackvar -reduction -xarch=v8plusb -xprefetch=
OSCAR 逐次	-fast -xarch=v8plusb -xprefetch=
OSCAR 並列	-fast -mp=openmp -explicitpar -stackvar -xarch=v8plusb -xprefetch=

ブによるソフトウェアプリフェッチを行うときには, "-xprefetch=explicit" を用いる.

また本論文では, SPEC95fp ベンチマーク中の tomcatv と swim を用いて提案アルゴリズムの性能を評価する.

tomcatv はベクトルメッシュを生成するプログラムであり, サブルーチンや関数を持たず, 実行時間の大部分は一つのメインループによって占められる. このメインループにはイタレーション間の並列性がないが, 内部は複数のループから構成される. これらすべてのループでローカライズ及びプリフェッチが適用できる. OSCAR コンパイラを用いた場合の評価においては, 第 2 章で述べた技術を使用して並列化しており, 整合分割の分割数は 8 としている. tomcatv では DOUBLE 型の配列を用いるので, 冗長なプリフェッチ命令を実行しないように 8 回アンローリングし, キャッシュコンフリクトを削減するためにパディングにより配列サイズを 513 × 513 から 513 × 544 に拡張している.

swim は Shallow water 方程式の求解プログラムである. 実行時間のほとんどは, メインループから呼ばれるサブルーチン calc1, calc2, calc3 によって占められる. これらサブルーチン内のループではローカライゼーション及びプリフェッチが適用できる. OSCAR コンパイラを用いた場合の評価においては, 第 2 章で述べた技術を使用して並列化しており, 整合分割の分割数は 8 としている. swim では REAL 型の配列を用いるので, ループは 16 回アンローリングし, キャッシュコンフリクトを削減するためににパディングにより配列サイズを 513 × 513 から 513 × 528 に拡張している.

4.2 tomcatv による性能評価結果

図 6 に tomcatv を各スケジューリング手法で 1, 2, 4, 8PE にスケジューリングした場合の実行時間を示す. また図中各 PE 数ごとの各 6 種の棒グラフは, 左から順に以下の手法でスケジューリングした場合の実行時間を示し, 図中の矢印とその上部の数字は, 比較する棒グラフとそれに対する速度向上率を示している.

- Forte_no

Forte コンパイラによる自動並列化 (プリフェッチなし)

- Forte_sw

Forte コンパイラによる自動並列化 (ソフトウェアプリフェッチを使用)

- Forte_sw+hw

Forte コンパイラによる自動並列化 (ソフトウェアプリフェッチとハードウェアプリフェッチを併用)

- OSCAR_no

OSCAR コンパイラ (プリフェッチなし)

- OSCAR_sw

OSCAR コンパイラ (ソフトウェアプリフェッチ)

- OSCAR_sw+hw

OSCAR コンパイラ (ソフトウェアプリフェッチとハードウェアプリフェッチを併用)

まず, 図 6 においてプリフェッチを使用しない場合の Forte コンパイラ (Forte_no) と OSCAR コンパイラ (OSCAR_no) の性能を比較すると, データローカライゼーションやパディングの効果により, OSCAR コンパイラは Forte コンパイラより実行時間を 8PE で 20.2%まで短縮できることがわかる. さらに Forte_no に対する OSCAR_no の実行時間の割合は PE 数が 1 の時に 61.9%, PE 数が 8 の時に 20.2%と PE 数が多くなるほど小さくなっている. これは tomcatv の総データサイズが約 14MB であり, 2PE 以上では各 PE で使用するすべてのデータが, L2 キャッシュ(8MB) におさまり, また OSCAR コンパイラのパディングによりラインコンフリクトを最小化しているためであると考えられる.

また, 提案するプリフェッチスケジューリングアルゴリズムを用いたとき (OSCAR_sw) は, さらに性能向上しており, プリフェッチしないとき (OSCAR_no) と比較して, PE 数が 1 の時に 1.51 倍, PE 数が 8 の時に 1.24 倍の速度向上を得ている. ここで PE 数が多いほどプリフェッチの効果が小さくなった原因としては, 各 PE が使用するデータが全て L2 キャッシュにおさまってしまうため, プリフェッチによって隠蔽されるデータ転送は, 主メモリから L2 キャッシュへの転送よりも L2 キャッシュからプリフェッチキャッシュへの転送となり利得が小さくなるためと考えられる.

図 6 より, 本手法は 8PE のときに, Forte コンパイラの 1PE ソフトウェアプリフェッチと比較して 6.37 倍, 8PE ソフトウェアプリフェッチと比較して 3.87 倍の速度向上を得た. これは, Forte コンパイラの 1PE プリフェッチなしと比較して 13.9 倍の速度向上にあたるなど, プリフェッチを効果的に利用することで, プロセッサ台数以上の速度向上を得るスーパーリニアスピードアップが効果的に引き出せていることが確認された.

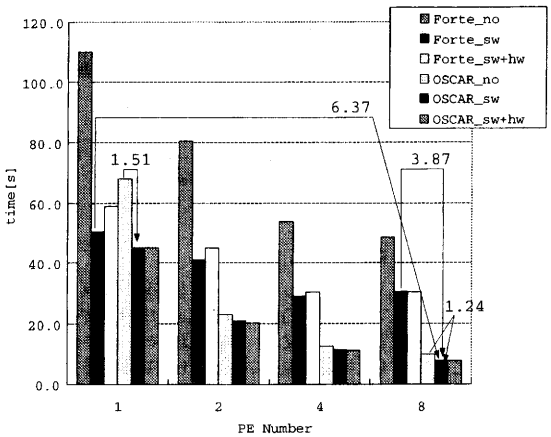

図 6 tomcatv におけるプリフェッチの効果
Fig. 6 Execution time of tomcatv

4.3 swim による性能評価結果

図 7 に swim を各スケジューリング手法で 1, 2, 4, 8PE にスケジューリングした場合の実行時間を示す.

まず, 図 7 においてプリフェッチを使用しない場合の Forte コンパイラ (Forte_no) と OSCAR コンパイラ (OSCAR_no) の性能を比較すると, データローカライゼーションやパディングの効果により, OSCAR コンパイラは Forte コンパイラより実行時間を 8PE で 37.9%と短縮できることがわかる. さらに Forte_no に対する OSCAR_no の割

合は PE 数が 1 の時に 64.4%, PE 数が 8 の時に 37.9%と PE 数が多くなるほど小さくなっている. これは swim の総データサイズが約 13MB であり, 2PE 以上では各 PE で使用するすべてのデータが, L2 キャッシュ(8MB) におさまるためパディングによるラインコンフリクト最小化が有効に働いているためであると考えられる.

また, 提案するプリフェッチスケジューリングアルゴリズムを用いてプリフェッチを使用したとき (OSCAR_sw) は, プリフェッチしないとき (OSCAR_no) と比較して, PE 数が 1 の時に 1.51 倍, PE 数が 8 の時に 1.04 倍のようにさらに性能向上が得られる.

図 7 より, 本手法は 8PE のときに, Forte コンパイラの 1PE ソフトウェアプリフェッチと比較して 13.0 倍, 8PE Forte ソフトウェアプリフェッチと比較して 1.85 倍の速度向上を得た. これは, Forte コンパイラの 1PE プリフェッチなしと比較して 21.5 倍の速度向上であり, ハードウェアプリフェッチも併用すると 23.6 倍の速度向上が得られている.

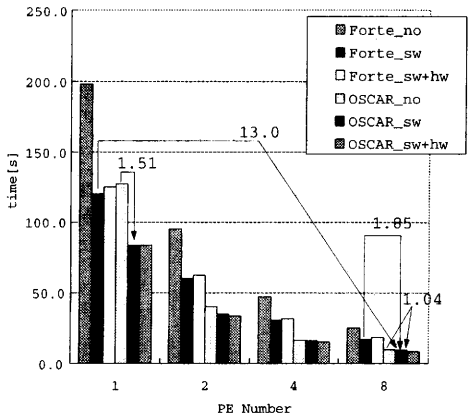

図 7 swim におけるプリフェッチの効果
Fig. 7 Execution time of swim

5. まとめ

本稿では, キャッシュプリフェッチを考慮したスケジューリングアルゴリズム DLG/DT-Gain/CP/MISF with Prefetch を提案し, それを OSCAR マルチグレイン並列化コンパイラに組み込み, Sun Fire V880 上で性能を評価した.

その結果, プリフェッチの効果的な利用と並列化により, SPEC95fp ベンチマーク中の tomcatv では Forte 1PE プリフェッチなし時と比較して, 8PE で最高 13.9 倍, swim では Forte 1PE プリフェッチなし時と比較して, 8PE で最高 21.5 倍の速度向上が得られた. また同じ 8 プロセッサ上では Sun Forte コンパイラによりプリフェッチディレクティブを自動挿入した場合と比較し, tomcatv では 3.86 倍, swim では 1.85 倍の速度向上が得られ, 提案したプリフェッチスケジューリングアルゴリズムの有効性が確認された.

今後の課題としては, 評価ベンチマークのさらなる充実や, プリフェッチ手法のダイナミックスケジューリングへの拡張などが挙げられる.

参考文献

1) Joseph, D and Grundwald, D.: Prefetching using Markov predictions, IEEE Trans. Comput., Vol. 48, No. 2, pp. 121–133 (1999).
2) Fredrik, D. and Per, S.: Evaluation of Hardware Based Stride and Sequential Prefetching in Shared Memory Multiprocessors, IEEE Trans. Parallel and Distributed Systems, Vol. 7, No. 4, pp. 385–398 (1996).
3) Mowry, T.C. and Lam, M.S. and Gupta, A.: Design and Evaluation of a Compiler Algorithm for Prefetching, Proc.Fifth Int'l Conf.Architectural Support for Programming Languages and Operating Systems, pp. 62–73 (1992).
4) IBM Server Group: POWER4 System Microarchitecture (2001).
5) Sun microsystems: Ultra SPARC III Cu User's Manual (2002).
6) Sun microsystems: An Overview of the Ultra SPARC III Cu Processor (2002).
7) 岡本雅巳, 小幡元樹, 松井巌徹, 松崎秀則, 笠原博徳, 成田誠之助: マルチグレイン並列化 FORTRAN コンパイラ, 情報処理学会論文誌, Vol. 40, No. 12, pp. 4296–4308 (1999).
8) 吉田明正, 越塚健一, 岡本雅巳, 笠原博徳: 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. 2053–2063 (1999).
9) 石坂一久, 中野啓史, 小幡元樹, 笠原博徳: ラインコンフリクトミスを考慮した粗粒度タスク間キャッシュ最適化, 情報処理学会 ARC 研究報告 (SWoPP2002) (2002).
10) ミレニアムプロジェクト IT 21 アドバンスト並列化コンパイラプロジェクト, http://www.apc.waseda.ac.jp/
11) 笠原博徳:最先端の自動並列化コンパイラ技術, 情報処理学会誌, Vol. 44, No. 4, pp. 384–392 (2003)
12) K. Ishizaka , M. Obata and H. Kasahara:Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multiprocessor, Proc. of 14th International Workshop on Languages and Compilers for Parallel Computing (2001).
13) K. Ishizaka , M. Obata and H. Kasahara:Cache Optimization for Coarse Grain Task Parallel Processing using Inter-Array Padding, Proc. of 16th International Workshop on Languages and Compilers for Parallel Computing, (2003)