計算機アーキテクチャ 151－4
（２００３．１．３１）

# MAPLE 用近細粒度並列化コンパイラ

岩井啓輔[†] 阿部剛[††]
森村知弘[††] 天野英晴[††]

マルチプロセッサシステム ASCA の要素プロセッサである MAPLE は近細粒度並列処理のためのさまざまなハードウェアサポート機構を備えている．MAPLE 用近細粒度並列化コンパイラはこれら独自の機構を使用するために，通常の並列化コンパイラの機能に加え，実行予測部，レシーブレジスタ割り当て部，DTC コード生成部などを備えている．

数値計算ベンチマークに本コンパイラを用いて近細粒度並列処理を行ない，各機能の有効性を確認した．

# Near fine grain parallelize compiler for MAPLE

Keisuke IWAI,[†] Tsuyoshi ABE,[††] Tomohiro MORIMURA[††]
and Hideharu AMANO [††]

MAPLE is a processing element of ASCA multiprocessor which is designed for automatic parallelizing compilers/schedulers. For efficient near fine grain parallel execution, it provides a special communication mechanism called Receive Registers, a light weight barrier synchronization, and a software cache controller DTC(Data Transfer Controller) with a simple RISC processor core.

A near fine grain parallelizing compiler which aims efficient use of such mechanisms is designed and implemented. It consists of a scheduler for Receive Registers, DTC code generator, MAPLE core simulator and basic code generation module. Using the current version of compiler, 1.6 times performance improvement is achieved with two processing elements using only near fine grain parallelism.

## 1. まえがき

マルチプロセッサシステムにおいて，その性能を発揮させるためには，逐次プログラムから自動的に並列性を抽出し，並列プログラムを生成する自動並列化コンパイラが重要である．近年，自動並列化コンパイラの使用する並列化手法はループ並列化を中心に多岐にわたり様々な並列性を利用する傾向にある．その中で，複数の粒度の並列性を同時に抽出し，利用するマルチグレイン並列化コンパイラ[1]が提案されている．我々は，マルチグレイン並列処理の効率よい実行をサポートするハードウェアを持ったマルチプロセッサアーキテクチャである ASCA(Advanced Scheduling oriented Computer Architecture)[2]を提案している．

ASCA はネットワーク，要素プロセッサ，メモリ構成などを，マルチグレイン並列処理の各粒度の特性に合わせて設計している．特に要素プロセッサとなる専用プロセッサ MAPLE[3] は，近細粒度並列処理[4][5] を効率よく実行するために次のようなハードウエアを備えている．

- 完全に動作予測可能なパイプライン
- 低オーバヘッドのプロセッサ間通信を行なうレシーブレジスタ転送機構
- キャッシュなどメモリの制御を行なうプロセッサ DTC

これらは近細粒度並列処理において非常に強力な能力を発揮するが，コンパイラが完全に実行時の挙動を予測しコントロールすることが前提となる．

本稿では，これらの近細粒度並列処理をサポートするハードウェアを利用する，近細粒度並列化コンパイラの実装について報告する．

† 防衛大学校，神奈川県横須賀市
National Defense Academy, Hashirimizu 1-10-20, Yokosuka, 239–8686 Japan
†† 慶應義塾大学理工学部，神奈川県横浜市
Faculty of Science and Technology, Keio University, Hiyoshi 3-14-1, Yokohama, 223–8522 Japan

## 2. 要素プロセッサMAPLE

### 2.1 MAPLEにおける近細粒度並列処理

ASCAの要素プロセッサMAPLEは，数個を1チップ上に搭載しチップ上のマルチプロセッサ上で近細粒度並列処理を実行するための機構を備えている．この近細粒度並列処理とは命令レベル並列処理(ILP: Instruction Level Parallelism)よりやや大きい粒度であり，ステートメントまたは複数のステートメントを単位として並列処理を行なう．特にシーケンシャルループのボディなどに適用され，ループ並列化の適用が困難な場合などに大きな効果を発揮する．

近細粒度並列処理では，ステートメントまたは複数のステートメント（BPA：疑似代入文）をタスクとして定義し，コンパイラはこれらタスク間の依存解析を行ない，タスクグラフを生成する．この時，基本ブロック内でタスクグラフが生成されるので，実行時の不確定性は存在しない．このため，コンパイラはスタティックスケジューリングを適用し，場合によっては，無同期で近細粒度並列処理を行なうこともできる[6]．

MAPLEでは，近細粒度並列処理を効率良く実行するために以下の機構を持つ．

- レシーブレジスタを用いたレジスタ間転送
- ソフトウェアキャッシュ機構

次節でこれらの並列処理機構を中心にMAPLEの構成を簡単に説明する．

### 2.2 MAPLEの構成

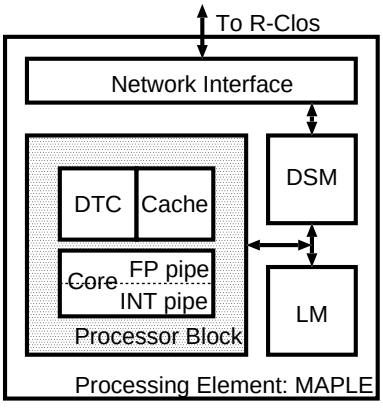


**図1** プロセッシングエレメントの構成

MAPLEは図1に示すようにローカルメモリ(LM)，通信用メモリ(COMM)，ネットワークインタフェース，そしてプロセッサで構成されている．プロセッサは，さらにプロセッサコア(MAPLE Core)，データ転送コントローラであるDTC[7]，およびオンチップキャッシュメモリから構成される．このうち近細粒度コンパイラにとって重要な部分である，コア部，DTC，及びレシーブレジスタ転送機構について簡単に解説する．

### 2.3 MAPLE Core

MAPLE Coreは，DLX ISA[8]にマルチプロセッサ用プリミティブ（通信，バリア同期）を加えた5段パイプラインを採用したRISCプロセッサである．動的な最適化技術を排除して実装されており，コンパイラはコンパイル時にクロックレベルでプロセッサコアの挙動を予測することができる．また，オンチップマルチプロセッサとして1チップ上に実装することを考慮し，少ないハードウェア量で実装されている．

### 2.4 レシーブレジスタ通信機構

レシーブレジスタ通信機構によるプロセッサ間データ転送の概念図を図2に示す．

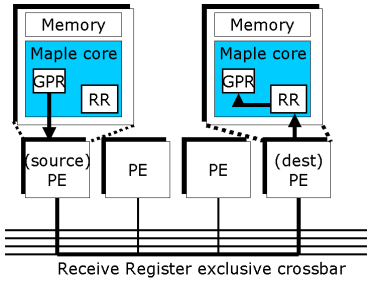


**図2** レシーブレジスタ通信機構

MAPLE coreは，GPRの他に受信専用のレジスタ（レシーブレジスタ）を持つ．近細粒度並列処理時など，主に小さいサイズのデータ転送が頻繁に起こる場合は，専用ネットワークを介して，自MAPLE coreのGPRの内容を直接相手MAPLE coreのレシーブレジスタに転送する．受信側は受信命令を発行して，受信したデータを汎用レジスタに移動することで受信動作が完了する．データの到着は，レシーブレジスタに到着を示すvalid bitを持たせることにより判断する．レシーブレジスタにデータが到着していなければ受信側プロセッサはストールする．一方で，送信は受信レジスタのvalid bitの状態に関わらず実行できる．これは，同一チップ上を想定するとはいえ，受信側PEのレシーブレジスタの状態をチェックして送信側PEの動作を停止させることが，実装上困難であることによる．したがって，この方式は，送信側がデータを上書きしないように制御する必要がある．この問題はレシーブレジスタを考慮したスケジューリング手法[9]によって解決している．

レシーブレジスタの操作プリミティブを次に示す．

- `sendr/sendri` ノンブロッキング送信命令．それ

ぞれレジスタオペランド，即値オペランドをとる

- `rcvr/rcvri` ブロッキング受信命令．それぞれレジスタオペランド，即値オペランドをとる

### 2.5 DTCとソフトウェアキャッシュ

MAPLEの静的予測可能な構成を利用して，プログラムのある区間でのデータのアクセスを予測し，その結果をキャッシュシステムに反映させることによってキャッシュのヒット率向上を狙うことができる．ASCAではこれを，DTCと呼ばれるキャッシュ制御プロセッサと，そのコンパイラによってこれを実現しており，ソフトウェア制御キャッシュと呼んでいる．

しかし，実際には，静的解析通りにハードウェアが動作出来なかった場合を考慮し，ASCAではハードウェアで管理する一般的な 4-wayキャッシュシステム機構も実装している．これらはソフトウェア／ハードウェアキャッシュモード切り替え機構を用意して対応している．

#### 2.5.1 DTCの構成と動作

DTCは，主にソフトウェア制御キャッシュを実現するための DTC Processor と 実際にキャッシュを制御するコントローラ，キャッシュメモリに対応したタグメモリより構成される．通常DTCは，コンパイラのDTCスケジューラによりMAPLE命令コードに埋め込まれたDTC発行タイミングでDTC 命令をフェッチし，DTC Processor がコードを解析し，キャッシュコントローラへ動作の指示を出し，キャッシュを制御する．

#### 2.5.2 DTC Processor

DTC Processorは，3段のパイプラインで構成される専用プロセッサで，独立した命令メモリから命令を取り出して動作し，キャッシュおよびタグをデータメモリとしてアクセスすることができる．しかし，その機能は，キャッシュ制御用特殊命令以外は，基本的な整数演算，メモリ（キャッシュ）アクセス，分岐命令などきわめて制限されており，少ないハード量で充分な性能を実現できる．DTCのキャッシュ制御用特殊命令は以下の二種類である．

- メモリ／キャッシュ間データ転送命令
- キャッシュ／MAPLEコア部間データ転送命令

順にその命令の使用方法を説明する．

**メモリ／キャッシュ間のデータ転送**

DTCはMAPLEコア部が必要とするデータを予めキャッシュに準備し，また，不要となったデータをメモリに書き戻す． この動作はDTC用コードの中のPL(Preload:プレロード)，PS(Poststore:ポストストア)によって行われる．PLとPS命令の命令フォーマットは以下のようになっている．

**PL** `mem_addr_reg,cache_addr_reg`

**PS** `mem_addr_reg,cache_addr_reg`

`cache_addr`はキャッシュアドレス，`mem_addr`はメモリアドレスである．

**キャッシュ／MAPLEコア部間のデータ転送**

ソフトウェア制御モードではキャッシュはハードウェアだけでは何処にどのアドレスのデータがあるのか分からないため，コンパイラがそれ指示する必要がある．その指示を行うのがLAC(Load Address Converter:ロードアドレス変換命令)，SAC(Store Address Converter:ストアアドレス変換命令)である．これらの命令は実際のMAPLEからのアクセスに先行して発行されるようスケジューリングされるため，タグ検索やキャッシュからデータを引き出すための時間を稼ぐことができ，キャッシュアクセス時間を短縮する効果もある．LAC，SAC命令の代表的な命令を示す．

**LC** `cache_addr_reg`

**LCV** `cache_addr_reg`

**SC** `cache_addr_reg`

**SCV** `cache_addr_reg`

`cache_addr`はキャッシュアドレスである．

LAC命令では，キャッシュは指示されたキャッシュアドレスのデータを準備してMAPLEコア部からのデータのロードを待つ．もしMAPLEコア部が指定したアドレスと，キャッシュが準備したデータのアドレスが異なった場合にはミスキャッシュとなり，DTCとMAPLEコア部の同期がうまく行われていないとみなされて，キャッシュの制御がソフトウェア制御モードからハードウェア制御モードに移される．

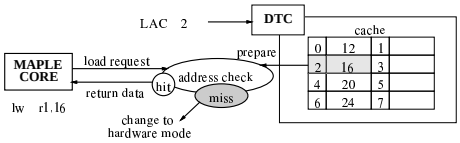


**図3** アドレス変換

SAC命令では，キャッシュは指示されたキャッシュアドレスにMAPLEコア部からストアされたデータを格納し，タグメモリにメモリアドレスを書きこむ．

## 3. 近細粒度並列化コンパイラ

今回実装した MAPLE 用近細粒度並列化コンパイラは複数のMAPLEを用いて効率のよい近細粒度並列処理を行なうために，以下の機構の有効利用を目指している．

- レシーブレジスタ転送機構の使用

- DTC の使用
- 無同期近細粒度並列処理の適用

このうち無同期近細粒度並列処理は現在のところ実装されていない．

### 3.1 コンパイルの流れ

本コンパイラによる近細粒度並列化のおおまかな流れを図4に示す．

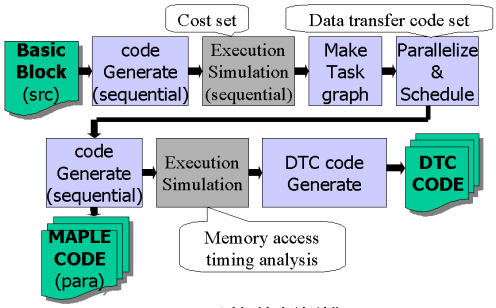


**図4** 近細粒度並列化

図に示した通り2パス構成を取り，実行シミュレーションを含む様々な処理を行なう．また，最終的に実行可能コードは，プロセッサ数分の MAPLE core の実行可能コードと，DTC のコードが出力される．

以下，コンパイルの流れに沿って各ステージの動作を概説する．

**a) シーケンシャルコード生成**

コンパイラは最初に，MAPLE コード生成部を利用し，並列化を一切行なわない通常のコード生成を行なう．この時，実レジスタ割り当ては行なわず，仮想レジスタを使用する．また，パージングと合わせてタスクの定義も行なっておく．

**b) タスクコストの設定**

生成された実行可能コードを利用し，実行シミュレーションを行なう．これによって，各近細粒度タスクのより正確なコストを見積もることが可能になり，後のステージで行なわれるスケジューリングの精度をあげることが可能となる．

**c) タスクグラフ生成**

2パス目の最初の処理となる．タスク間の依存解析を行ない，タスクグラフを生成する．

**d) スケジューリング，RR割り当て**

タスクグラフに基づいてリストスケジューリングを行なう．スケジューリングによりデータ転送が決定するこの時点で，転送命令なども付加される．コンパイラの中間表現ではこの時点で，並列化が完了する．

**e) コード生成**

並列化中間コードから各プロセッサの実行可能なコードを生成する．

**f)DTCコード生成**

MAPLE 用実行可能コードとシミュレータにより，詳細なメモリアクセスタイミングを予測する．このタイミングを用いて，ソフトウェア制御キャッシュを用いる DTC コードの生成を行なう．

### 3.2 コンパイラの構成

近細粒度コンパイラ本体はモジュール化されており，その構成を図5に示す．

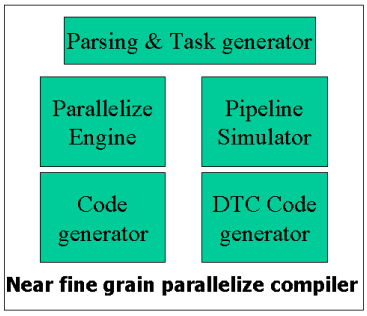


**図5** コンパイラの構成

**フロントエンド＆タスク生成部**

ここでは基本ブロックの字句，構文解析と共に，ステートメント単位でタスクを生成し，タスクグラフを生成する．コンパイルの最初に一度だけ使用される．

**MAPLE core コード生成部**

通常のコンパイラ同様，中間表現から実行可能コードを生成する部分である．このモジュールは一切並列化などは行なわない．現在の近細粒度コンパイラでは，1パス目に並列化されていない命令列が入力され，シーケンシャルな実行可能コードを生成する．また，2パス目では，既に並列化された，各プロセッサ毎の中間コードがここに入力され，各プロセッサ毎に実行可能コードを生成する．生成されるコードの最適化については，現在 gcc の -O1 オプション相当の最適化が実装されている．

**実行シミュレータ部**

コンパイラにより生成された実行可能コードを入力として，MAPLE core のパイプラインを，詳細にシミュレートする．完全な MAPLE core シミュレータで，整数パイプラインに加え，浮動小数点パイプライン，レシーブレジスタ転送命令，バリア命令も実行可能であり，マルチプロセッサ動作についてのシミュレーションも行なうことができる．

**並列化部（スケジューリング部）**

近細粒度タスクの並列化を行なう．タスクグラフに基づきリストスケジューリングを用いて並列化を行なう．リストスケジューリング時には実行シミュレータ部を使用し，詳細な実行見積もりを行ないスケジュー

リングの精度をあげている．

ここではデータ転送命令も付加されるが，上書きを回避するレシーブレジスタ割り当て手法を適用し，レシーブレジスタの割り当てを済ませておく．

**DTC コード生成部**

ここでもシミュレータを用いて，実行時の詳細なメモリアクセスタイミングを解析し，MAPLE コードと DTC コードを生成する．コード生成は，基本的に以下の手順を踏む．

**i)** 最初に通常のコンパイラ (従来のコンパイラ) 技術によりアセンブリコード (MAPLE コード) 生成

**ii)** アセンブリコード中のデータをロードする Load 命令，データを書き込む Store 命令に注目し，データのキャッシュ内配置の決定，対応する DTC 命令 (LAC,SAC,PL,PS) 及び各アドレス演算用コードを生成

**iii)** Load, Store するデータはキャッシュにヒットするという前提条件のもとで，生成したアセンブラコードを CPU シミュレータにかけ，各命令の正確な実行クロックを算出

**iiii)** CPU が Load,Store をする際に確実にキャッシュ上にデータが存在するように LAC,SAC,PL,PS 命令の発行タイミングを調整

これらのスケジューリングは全てアセンブリ言語レベルで行なう．LAC, SAC は全ての Load, Store の直前に発行することで，CPU との同期を図ると共に，実際の実行時と予測との整合性をとる．そのために厳密に配置する必要があり，最初に位置を固定する．PL は，対応する LAC の前には必ず終わるよう保証し，また PS は，対応する Store 命令の後に発行するよう保証すればよい為，LAC,SAC 命令の間に後から配置される．PL, PS の発行は変数 (仮想レジスタ) のライフタイムの最初と最後で行なうため全ての Load, Store に対して行なうわけではない．LAC,SAC を発行を決定する際，実際にはさらにキャッシュの上書き回避のための LC,SC 命令から LCV,SCV 系命令への変更，調整を行なう．

## 4. 性能評価

### 4.1 評価環境

#### 4.1.1 MAPLE cluster

評価には，オンチップマルチプロセッサを想定したマルチプロセッサ MAPLE cluster を用いる．MAPLE cluster は 複数 MAPLE をクロスバで接続し，レシーブレジスタ通信機構を用いてプロセッサ間通信を行なうことができる．

MAPLE チップは既に実装されており，そのモジュールは全て Verilog-HDL によって記述されている．そこで，今回の評価は，実際の設計データに基づく RTL シミュレーションで行なっている．MAPLE cluster の各パラメータを次に示す．

- 相互結合網:クロスバ
  - 1 クロックで転送可能
  - 調停 1 クロック
  - 転送データ幅 32bit
- レシーブレジスタ数 8 本/pe
- 動作周波数 50MHz☆

#### 4.1.2 アプリケーション

評価プログラムにはベンチマーク SPEC FP 95[10] から量子化学計算を行なうアプリケーションである fpppp を用いた．fpppp は幾つかのサブルーチンから構成されるが，実行時間の大部分を担当するサブルーチン `fpppp` は大規模な近細粒度タスクをもつ．今回はこのサブルーチン部を抜き出し，使用した．

#### 4.1.3 コード生成

コード生成は今回実装した近細粒度並列化コンパイラを用いた．

比較対象としては，従来の MAPLE cluster 用の評価に用いられていた近細粒度並列化プリプロセッサによる MAPLE 用並列化 C を gcc-dlx でコンパイルしたものを用いた．近細粒度並列化プリプロセッサによるスケジューリングは，今回提案した近細粒度並列化コンパイラと同じなので，今回の評価は実質，コード生成部の評価となる．gcc-dlx によるコード生成ではコンパイラオプションに -O3 を用いた．また，gcc-dlx のバージョンは 2.7.2.3 である．評価に用いるアプリケーションのデータセットが小さいので，キャッシュミスヒットはないと仮定してシミュレーションを行なった．このため，DTC コードは生成されているが，それによる効果は一切ないので省略する．

### 4.2 実行結果

プロセッサ数を 1〜3 まで変化させた時の実行時間を図 6に 示す．また，gcc-dlx を用いたコンパイラと近細粒度並列化コンパイラの実行時間の比を表 1に示す．

評価に用いた近細粒度ブロックは 8 程度の並列性があるが，今回の評価では 3 プロセッサ (図中 3pes) で頭打ちになっている．これはタスクのスケジューリングによる損失が原因と考えられる．DT/CP 法にレシー

☆ 実装された最新の MAPLE core の動作周波数は 80MHz である

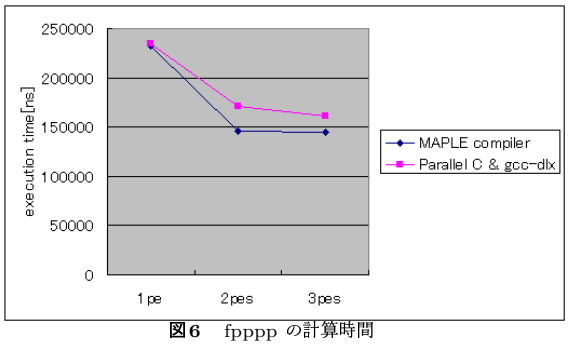


**図6** fpppp の計算時間

**表1** 各実行時間の比較
(本コンパイラによる実行時間を1とする)

| プロセッサ数 | gcc-dlx | 本コンパイラ |
|---|---|---|
| 1 | 1.11 | 1 |
| 2 | 1.17 | 1 |
| 3 | 1.01 | 1 |

ブレジスタを組み合わせたスケジューリングではプロセッサ数が増加した場合の近細粒度タスクの並列性を活かしきれないので，スケジューリングに関してはさらにチューニングを行なう必要がある．しかし，2プロセッサでの並列効果としては1.6倍の性能向上と，まずまずの値を得ることができている．

コード生成に関しては，おおむね -O3 オプションを使用した gcc-dlx を上回るパフォーマンスを得ている．これは，レシーブレジスタ割り当てと汎用レジスタ割り当てをコード生成時に一本化するなどの MAPLE 用命令の最適化がうまく機能しているためと考えられる．

## 5. むすび

近細粒度並列処理を効率良く実行するためのプロセッサアーキテクチャMAPLE 用の近細粒度並列化コンパイラを実装した．このコンパイラでは，通常の近細粒度並列化を行ないさらに MAPLE 専用ハードウェアであるレシーブレジスタの割り当てや，DTC のコードを生成することが可能である．

アプリケーションを用いて性能を評価した結果，レシーブレジスタを使用するコードが生成可能であり，コード生成については十分な性能を持つことが確認できた．特に専用命令のコード生成による利得があることもわかった。

一方，現在のスケジューリング手法でプロセッサ数を増やした場合に並列性を十分に利用できないという問題も明らかになっている．

今後は本コンパイラについてはスケジューラの改良を中心に行ない，さらにマルチグレイン処理も考慮した評価を行なう．

## 謝　辞

本研究の一部は、半導体理工学研究センター (STARC) の援助による。

## 参考文献

1) 小幡 元樹, 松井 巌徹, 松崎 秀則, 木村 啓二, 稲石 大祐, 宇治川 泰史, 山本 晃正, 岡本 雅巳, 笠原 博徳. OSCAR FORTRAN Compilerを用いたマルチグレイン並列性の評価. 情報処理学会研究報告98-ARC-130-3, pp. 13–18, Aug. 1998.

2) 岩井 啓輔, 森村 知弘, 川口 貴弘, 酒井 敦, 阿部 剛, 天野 英晴. コンパイラ主導型プロセッサASCAのアーキテクチャ. JSPP2000論文集, pp. 3–10, June 2000.

3) T.Abe, T.Morimura, T.Suzuki, K.Tanaka, M.Koibuchi, K.Iwai, and H.Amano. ASCA chip set: Key components of multiprocessor architecture for multi-grain parallel processing. In *Proc. of COOL Chips IV*, pp. 223–247, Apr. 2001.

4) 笠原 博徳. マルチプロセッサシステム上での近細粒度並列処理. 情報処理, Vol. 37, No. 7, pp. 651–661, Jul. 1996.

5) 岩井 啓輔, 川口 貴弘, 鈴木 貴幸, 森村 知弘, 阿部 剛, 天野 英晴. 並列計算機ASCAの要素プロセッサによる近細粒度並列処理. 信学論, Vol. J85-DI, No. 6, pp. 491–502, Jun. 2002.

6) 尾形 航, 吉田 明正, 合田 憲人, 岡本 雅巳, 笠原 博徳. スタティックスケジューリングを用いたマルチプロセッサシステム上の無同期近細粒度並列処理. *JSPP '93*, 1993.

7) 坂本 勝人, 藤原 崇, 川口 貴裕, 岩井 啓輔, 森村 知弘, 天野 英晴. マルチグレイン並列処理を利用したソフトウェア制御キャッシュの開発. *CPSY '98*, No. 26, pp. 9–16, April 1998.

8) John L. Hennessy and David A. Patterson. Computer Architecture A Quantitative Approach. Morgan Kaufmann Publishers, 1993.

9) 岩井 啓輔, 阿部 剛, 森村 知弘, 笠原 博徳. MAPLE core によるレシーブレジスタ割り当て手法. 情報処理学会研究報告2002-ARC-146, pp. 37–42, Feb. 2002.

10) SPEC Corporation. SPEC Web page (http://www.spec.org).