計算機アーキテクチャ　107－5
（１９９４．　７．　２１）

汎用マイクロプロセッサを用いた Datarol-II プロセッサエレメントの構成と評価

富安 洋史　川野 哲生　谷口 倫一郎　雨宮 真人

九州大学大学院総合理工学研究科

超並列計算で問題となるリモートメモリアクセス等のレイテンシの隠蔽のためには細粒度スレッド処理が有効である．そのような視点から我々は Datarol-II と呼ぶ細粒度スレッド処理に適したアーキテクチャを提唱してきた．しかし，Datarol-II は専用プロセッサを必要とするため設計コストが大きくなることが問題である．Datarol-II はスレッド内部の処理を逐次的に行なうため，スレッドの同期機構など細粒度スレッド処理を高速化する機構を外付け回路で実現すればスレッド内部の処理は汎用 micro processor に置換えることができ，大幅なコストダウンが可能となる．本稿では汎用 micro processor に細粒度スレッド処理機構を外付けして構成した Datarol-II プロセッサエレメントの性能評価について述べる．

Design and Evaluation of Datarol-II Processor Element, Using General Purpose Micro Processor

Hiroshi Tomiyasu, Tetsuo Kawano,
Rin-ichirou Taniguchi and Makoto Amamiya

Department of Information Systems, Graduate School of Engineering Sciences,
Kyushu University

In massively parallel processing, one of the most critical issues is the latency problem caused by remote memory accesses and remote procedure calls. To solve the problem, we have been developing a Datarol-II processor, a sophisticated fine-grain multi-thread processor, which is quite effective for hiding the latency. The Datarol-II processor has a problem, an expensive cost of its development.

However, we can replace the thread execution unit of a Datarol-II processor by a general purpose micro processor with some additional hardware for context switching, which can significantly reduces the development cost. In this paper, we present the design and the performance evaluation of a Datarol-II processor based on a general purpose micro processor.

1 はじめに

超並列計算における問題のひとつは，プロセッサ間通信によるレイテンシである．このレイテンシを隠蔽するためには細粒度のプロセスを処理の単位とし，レイテンシが生じる場合には他の実行可能な細粒度プロセスに切替えてプロセッサのスループットを保つ手法が有効である．

我々は細粒度スレッド処理に適した Datarol-II と呼ぶ並列計算機の設計開発を行なってきた．Datarol-II はデータフロー方式をより最適化した Datarol アーキテクチャーをもとに設計され，効率的なマルチスレッド処理の実現を目指したマシンである．

Datarol-II ではスレッドと呼ぶ細粒度の処理単位の概念を導入して Datarol アーキテクチャーのハードウェアコストを軽減している．ここで言うスレッドは同期なしに実行できる命令列であり，排他的に実行されるためプログラムカウンタによって制御される．Datarol-II はこの細粒度スレッド実行モデルを効率よく実行することを目的としており，特にコンテクストの切替によるオーバーヘッドの削減に主眼が置かれている．コンテクストの切替に伴うメモリアクセスを隠蔽するため，レジスタの自動ロードストア機構を持ち，メモリアクセスと ALU 演算をオーバーラップして行なうことにより，オーバーヘッドを削減する．

しかし，Datarol-II プロセッサエレメントは専用プロセッサを開発する必要があるため，設計コストの増大が問題となる．しかし，Datarol-II はスレッド内部の実行にプログラムカウンタを用いるため，このスレッド内部の処理をノイマン型プロセッサを用いて行なうことができ，汎用マイクロプロセッサを用いることによって設計コストの増大を押えることができる．

汎用マイクロプロセッサは細粒度スレッド処理のための機構を持たない，コンテクスト切替時のメモリアクセスがオーバーヘッドとなる，といった問題がある．しかし，Datarol-II で行なわれるスレッド同期処理や実行可能スレッドの待ち行列の管理を外付け回路で行なうことによって実現し，コンテクスト切替時のメモリアクセスはスーパースカラで ALU 演算とメモリアクセスをオーバーラップすることによってオーバーヘッドを削減することができる．汎用マイクロプロセッサにこれらの回路を付加することによって完全に専用設計としたものと，同等の能力をもつ Datarol-II を構成できれば，大幅なコストダウンが可能となる．

また，汎用マイクロプロセッサの速度向上は常に専用プロセッサを持った並列計算機を脅かしているが，汎用マイクロプロセッサを用いることによって Datarol-II もその速度向上の恩恵をうけることができる．

本稿では，汎用マイクロプロセッサを用いた Datarol-II プロセッサエレメントの基本設計を行ない，すべてを専用設計とした Datarol-II とコンテクスト切替のオーバーヘッドについて比較を行なった．

2 プロセッサエレメント (PE) 設計の方針

2.1 Datarol プロセッサエレメント

データフローモデルは，プログラムに内在する並列性の抽出が自動的に行なえるため，並列計算モデルに適している．しかし，ペアオペランドの同期のコストが大きい，メモリの概念がないため無駄なデータのコピーが発生してしまうといった問題点を抱えている．

我々はデータフロー方式の問題点を解決するため Datarol と呼ぶアーキテクチャーの提案を行なってきた．Datarol はデータフロー方式にレジスタを導入し，data-flow graph から冗長なフロー制御を削除した Datarol graph を実行のモデルとしている．

各関数インスタンス (以下単にインスタンスという) はそれぞれ固有のレジスタセット (論理レジスタとよぶ) を持ち，インスタンス内のデータ受渡しにはこのレジスタを用いる．

Datarol はデータフローから冗長なデータのコピーとフロー制御を削除したが，循環パイプライン方式であるため，並列度が低い場合にパイプラインに空きが生じやすい．専用メモリを多く持っているため，ハードウェアコストが大きい．などの問題があった．

2.2 スレッド 概念の導入

Datarol-II ではこの問題を解消するために thread 実行概念を導入している．図 1 に Datarol-II における thread 実行概念を示す．

Datarol-II で言う thread は thread 内部で必要とされる データ (引数) が揃った時点で発火され，発火後は thread 内部の命令列はノイマン型プロセッサと同様プログラムカウンタを用いて逐次的に実行される．

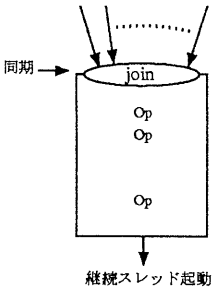

図 1: thread の概念

thread 内の命令を全て実行すると自動的に thread は終了し，他の実行可能な thread が起動される．

2.3 Native Datarol-II

Datarol-II(以下 Native Datarol-II とする) は thread 実行概念を効率良く実装することを目的として設計されている．図 2に Native Datarol-II の構成を示す．Datarol-II では thread を最小単位とした実行を行なうため，FU(Function Unit) は thread 内の処理を逐次的に行なう．thread の同期には同期カウンタを用い，AC (Activation Controller) が自動的にカウンタの更新を行なう．

また，Datarol-II ではコンテクスト切替を高速化するため，自動的にレジスタのロードストアを行なう機構を組み込んでいる．Datarol ではインスタンスごとに論理レジスタを持つが，実際には高速なレジスタは FU 内部に少数しか用意できないため，コンテクストの切替にともないレジスタの入れ替えが必要となる．レジスタの自動ロードストア機構によって算術論理演算とレジスタの入れ替えがオーバーラップして行なわれるため，コンテクスト切替時間が削減される．

また，論理レジスタを標準的な DRAM で構成することを想定し，RB(Register Buffer) と呼ぶ小容量の高速メモリを用いてメモリの階層化を行なっている．

2.4 汎用マイクロプロセッサを使用した Datarol-II

Native Datarol-II は細粒度 thread 実行モデルのために最適化されているが，専用プロセッサを前提としているため，PE 全てをはじめから設計する必要がある．また，クロックを上げて高速化を行なう場合も専用プロセッサ開発のためのテクノロジー条件を含めて検討する必要があり，設計コストが上昇する．汎用マイクロプロセッサが使用可能であれば，外付けの細粒度スレッド処理の高速化回路のみを設計し直せば良いため，大幅に負担が軽減される．

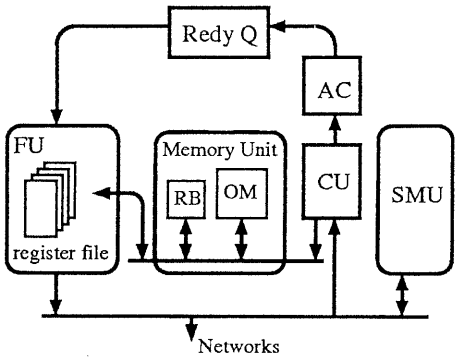

図 2: Native Datarol-II PE

従来の汎用マイクロプロセッサはコンテクスト切替のオーバーヘッドが大きく，細粒度 thread 実行には不向きであった．しかし，近年汎用のマイクロプロセッサの性能が大きく向上し，特にスーパースカラによりコンテクスト切替で生じるロードストアを他の命令とオーバーラップできるようになった．

thread 同期のためのカウンタのチェックや更新，あるいはインスタンスへの論理レジスタの割り当てなどは依然としてオーバーヘッドとなる．しかし，これらを外付け回路で処理することができれば，Native Datarol-II との性能差を縮小できる．

もし，汎用マイクロプロセッサに簡単な外付け回路を付加することにより十分な性能のプロセッサエレメントを実現することができれば，コストの点で大きく有利になる．

また，汎用マイクロプロセッサは年々高速化しているため，マイクロプロセッサを使用すればマイクロプロセッサの世代交替に応じて高速化をはかることができる．

2.5 解決すべき問題

汎用マイクロプロセッサを用いて Datarol-II を構成する場合に問題となるのは以下の 3 点である.

- thread 切替時に生じるコンテクスト切替のオーバーヘッド.
  Native Datarol-II はレジスタの自動ロード機構を備えている. 汎用マイクロプロセッサを用いた場合は明示的なロードストア命令を入れる必要がある. このロードストアをスーパースカラでどの程度隠蔽できるかが焦点である.

- thread の発火制御機構の実現法.
  Native Datarol-II と同等の発火制御機構を外付け回路で実現する必要がある. Datarol-II ではパケットの形で発火制御機構へ情報が渡されるが, 汎用マイクロプロセッサでは自由にパケットを作りにくいためメモリ書き込み等をつかって同等の機能を実現する.

- 外付け回路のハードウェアコスト.
  汎用プロセッサを用いても外付け回路が複雑になりすぎては意義がうすれるため, 単純な構成とする. また, 将来のマイクロプロセッサの速度向上に対応するためにも, 単純な構成であることが必要である.

以上の点に注意して汎用マイクロプロセッサを用いた Datarol-II の基本設計を行ない, コンテクストの切替のオーバヘッドを Native Datarol-II と比較した.

3 PE の構成

3.1 構成の概要

図 3 に汎用マイクロプロセッサを用いた Datarol-II PE の構成を示す. ここで, TAC (Thread Activation Controller) は thread 同期機構, RQ (Ready Q) は実行可能な thread を記憶する FIFO, ICU (Instance Control Unit) は インスタンス割り当て時の論理レジスタの管理を行なう. TAC,RQ 及び ICU はメモリ上にマッピングされており, CPU はメモリアクセス命令を用いてこれらを操作する.

CPU, メモリ, TAC 等の各構成要素がバスを介して接続された単純な構成となっている. thread 発火

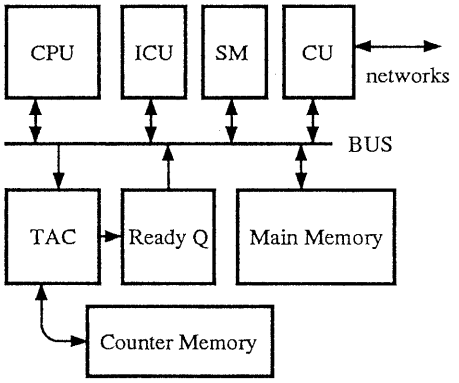

図 3: 汎用マイクロプロセッサを用いた Datarol-II PE

制御に使用される カウンタ のみは専用メモリで構成され, TAC によって管理される. 同期カウンタ以外には専用メモリをなるべく使用せず, 配線や I/O pin のコストを大きくしないようにした.

必要とする専用メモリは同期カウンタのみであり, バス以外に必要とする配線も少ないため, コストを低く押えることができるとともに, 高速化にも有利である.

汎用マイクロプロセッサを用いた実装方式と Native Datarol-II と大きく違う部分は, 細粒度スレッド処理の高速化を行なうために汎用マイクロプロセッサに付加した TAC である.

その他は CPU とのインタフェースに合わせるだけで Native Datarol-II と同じ設計とすることができる. また, PE 間ネットワークとのインタフェースからネットワーク側はまったく同一の仕様とすることができる.

3.2 CPU

プロセッサエレメントに使用する CPU には, 以下のような条件が求められる.

- 特殊目的でなく, 一般的に使用されているもの. 安価に入手可能であること. 継続して使用可能であること. 次世代のプロセッサに無理なくつながること. などが重要である.

- 高性能であること. 特にロードストアの性能が良いこと.

コンテクスト切替にともなうロードストアを高速に行なうものが望ましい.

これらの条件を満たす CPU の一つとして今回は Intel 社の Pentium を想定して評価した.

Pentium は PC 用として広く使われており一般性に関しては全く問題が無い. また, 性能的にも最新のマイクロプロセッサとして遜色ない. 特に 2 つの完全な整数演算ユニットを持ち, ロードストアパイプも 2 つあることから, コンテクスト切替が頻繁に起こる細粒度 thread 処理においては有利であると思われる.

他に一般に使用されているマイクロプロセッサとして Super Sparc や R4000 等についても検討したが, これらはロードストアパイプが 1 つであるため, Datarol-II のようにコンテクスト切替が頻繁に起こる場合は Pentium よりも不利になる.

ロードストアパイプを 2 つ以上持つマイクロプロセッサはまだ一般的では無い. しかし, スーパースカラ度が増え, 同時実行可能な ALU 演算命令が増えるに従い相対的にロードストアパイプが不足する. したがって今後ロードストアパイプを複数持つプロセッサが一般的になっていくと思われる.

3.3 メモリマップ

汎用マイクロプロセッサを用いた Datarol-II では TAC への情報として CPU がバス上に出力するアドレスを利用しているため, メモリマップが重要である.

図 4 にメモリマップを示す. Data_Area は各インスタンスが変数の格納に使用する領域である. 各インスタンスには固定長の frame (Datarol での論理レジスタ) が割り当てられる. この frame のアドレスがそのままインスタンス名として空き領域の管理などに使用される. Code_Area は命令を格納する.

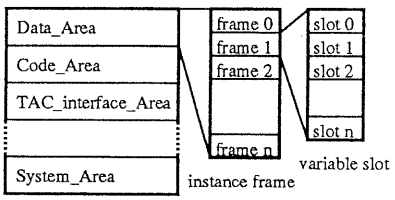

図 4: memory map

TAC_interface_Area は Data_Area と同じサイズを割り当てられ, この領域に CPU が書き込んだ場合は TAC への命令発行として取り扱われる. 従って実際にはメモリは実装されない. TAC はバス上のアドレスからアクセス先のインスタンス名 (単純にオフセット分を引くことによって得られる) を知り, CPU からのデータで対応する同期カウンタ及び, 発火すべき thread の開始アドレスを得る.

その他 ICU, RQ, CU 等は memory mapped I/O として System_Area にマッピングされている.

3.4 thread 発火制御機構 (TAC)

TAC による thread の発火制御は以下のように行なわれる.

0	IP Address	Counter No.

(a) continuation 情報

1	Init Counter	Counter No.

(b) counter 初期化コマンド

図 5: continuation 情報 及び初期化コマンド

step 1 continuation 情報の書き込み
CPU は引数をメモリに書き込むと同時に TAC に thread 起動情報 (以下 continuation 情報と呼ぶ) を書き込むことによって TAC を起動する.

step 2 同期カウンタのロード
図 5 (a) に continuation 情報のフォーマット, 図 6 (b) にカウンタ memory の構成を示す. continuation 情報には カウンタ No. が含まれており, 同期カウンタのアドレスは CPU がバス上に出力したアドレスの上位と カウンタ No. から得られる. TAC は continuation 情報の書き込みによって起動し, 同期カウンタをロードする. もし, continuation 情報によって書き込み先が 1 つの引数しか必要としない場合はカウンタのロードは行なわない.

step 3 thread 発火, 同期カウンタの書き戻し
カウンタをチェックして同期が成立すれば thread

の発火を行なう．TAC は RQ に thread の開始アドレスとインスタンス名を出力する．thread の開始アドレスは continuation 情報から得られ，インスタンス名 (frame No.) は CPU の出力するアドレスから得られる．
同期が成立しなければ同期カウンタを減算して書き戻しを行なう．

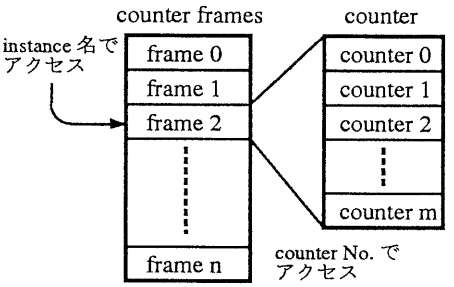

図 6: counter memory の構成

同期カウンタの初期化は CPU が TAC に対して初期化コマンドを書き込むことによって行なう．図 5(b) に初期化コマンドを示す．TAC は CPU に代わって同期カウンタを管理することによって thread の同期のオーバーヘッドを軽減する．CPU はカウンタの減算，書き戻し，thread の発火 (Ready Q への出力) などを行なう必要が無い．オーバーヘッドは同期カウンタの初期化と continuation 情報の書き込みのみである．

3.5 RQ(Ready Q)

RQ は実行可能な thread を蓄える FIFO である．CPU の負担を減らすため図 7 に示すように Ready Q の先頭は常に特定番地にマッピングされ，各 thread の終了時に CPU は RQ の先頭のアドレスを計算せず，常に特定番地をアクセスして次の thread に jump する．

RQ に実行可能な thread がない場合は RQ は dummy の frame No. とシステムで用意された dummy の thread の開始番地を先頭にセットする．この thread はすぐに実行を終了し RQ の先頭を読む処理を行なう．したがって，RQ が空であるかどうかをチェックするコードは必要としない．機械的に RQ の先頭 (特定番地にマッピングされているので即値で埋め込むことができる) を読むコードを 各 thread の終りに付加

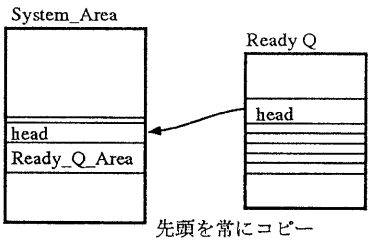

図 7: Ready Q

するだけでよい．Pentium の場合は 2 命令で thread の切替を終了することができる．

4 性能予測

我々は汎用プロセッサを用いて構成した Datarol-II の性能評価のため，ソフトウェアシミュレータを作成して，Native Datarol-II との比較を行なった．これらのシミュレータはクロックレベルで動作し，CPU 部分は Pentium を想定して Pentium のサブセットの動作ができるものを用意した．メモリアクセスのレイテンシはいずれも 3 clock である．また，Pentium 版のデータキャッシュサイズ及び Native Datarol-II の RB サイズはいずれも 16KByte である．

実際の Native Datarol-II は専用プロセッサを開発する必要があり，クロックは専用プロセッサによって決まる．Native Datarol-II のような，専用プロセッサは汎用マイクロプロセッサのように大きなコストをかけて設計できないため，クロックを上げることが難しい．したがって，Native Datarol-II の動作クロックは Pentium より低くなると思われるが，クロック速度の正確な見積りが非常に難しいため以下の評価では同一とした．

また，今回単体 PE のみの評価を行なった．
使用した例題は以下の 2 つである．

i fib
: フィボナッチ数を再帰的に求める
thread 長が非常に短い．汎用プロセッサ版ではコンテクスト切替のオーバーヘッドが大きく現れる．関数展開の幅が大きいためキャッシュミスヒットが起こりやすい．汎用プロセッサ版にとっての最悪値として使用できる．

ii **Queen**

N-queen を全解探索で求める

thread 長は一般的な長さに近付く．探索問題の典型的な例であり，実際の性能はこちらに近いと思われる．

以下に，2 つの例題の性能予測結果について示す．

fib

fib の thread 長はロード命令を含めない場合は 4 命令程度，ロード命令を含めて 6 命令程度であり，非常に短いため fib は汎用プロセッサを用いた実装方式にとって最悪値に近い結果となる．一般的なプログラムにおいては，少なくとも fib より性能が落ちることはないと思われる．図 8 に fib の実行時間の比較を示す．

thread 長が短いためコンテクスト切替のオーバーヘッドが大きく出ている．特に問題サイズの小さいところで顕著で,Pentium を用いた実装方式の性能は Native Datarol-II の 0.3 倍程度しかない．問題サイズが大きくなるにつれ，性能差は小さくなり，0.8 倍程度にまで，改善される．

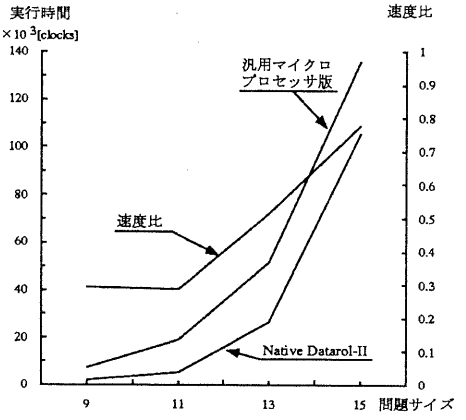

図 8: fib の実行時間

問題サイズが大きな部分で性能差が小さくなるのは Native Datarol-II でキャッシュに相当する RB へのロード時のオーバーヘッドが原因である．

表 1に Pentium を用いた実装方式におけるキャッシュヒット率と Native Datarol-II における RB のスワップ時間を示す．Native Datarol-II の RB は一般的なキャッシュと違いレジスタセットを自動的にロードする構成になっており，単純にキャッシュヒット率を算出することができないため，表中に RB スワップ時間を全実行時間中に占める RB の入れ替えに要した時間の割合で示している．

問題サイズ	fib(9)	fib(11)	fib(13)	fib(15)
データキャッシュヒット率 [%]	79.78	74.62	69.93	68.41
RB スワップ時間 [%]	0.0	0.0	34.7	47.5

表 1: fib のキャッシュヒット率

Native Datarol-II では，fib(11) までは全くスワップが生じておらず，全てのデータは RB 上に存在しているが，fib(13) 以降では RB の容量が不足するためスワップが行なわれ始めており，同時に，Native Datarol-II と Pentium を用いた実装方式と性能差が大きく縮まっている．

これは Native Datarol-II における RB のミスヒットのペナルティが大きいためである．Native Datarol-II では自動ロードの対象は論理レジスタセットとなっており，RB へのミスヒットが生じた場合は毎回 16 word のデータすべてをストア，ロードしている．しかし，fib の thread で必要なデータは 2,3 個程度しかなく，16 word のデータのうち，ほとんどが無駄になっている．このため，先行読み込みを行なってもスレッド長が短いため自動ロードストア機構のスループット自身が不足し，RB のミスヒットが大きなペナルティとなっている．

対して，Pentium 版では明示的にロードストア命令を挿入しているため，必要以上にロードストアを行なうことがない．加えて Pentium にはデータキャッシュとメモリ間にラインフィルバッファがあり，キャッシュミスヒット時に読み込まれたデータは，ラインフィルバッファに読み込まれると同時に使用可能になるためラインサイズ分のデータを待つ必要が無く，キャッシュミスヒット時のペナルティが少なくなっている．Native Datarol-II では RB 上に必要な論理レジスタセットがない場合は，RB 上に 16 word のデータ全てをロードするまで遅延が生じる．

Queen

Queen の thread 長さはロード命令を含めない場合は 7〜8 程度, ロード命令を含めて 11〜12 程度とやや長く，一般的な問題に近付く．したがって，一般的な問題における性能は fib よりも Queen の結果に近い

と思われる.

図9 に Queen の実行時間を示す. fib よりも thread が長いためコンテクスト切替のオーバーヘッドは相対的に小さくなっており, 性能比は 0.42 ～0.85 程度となる.

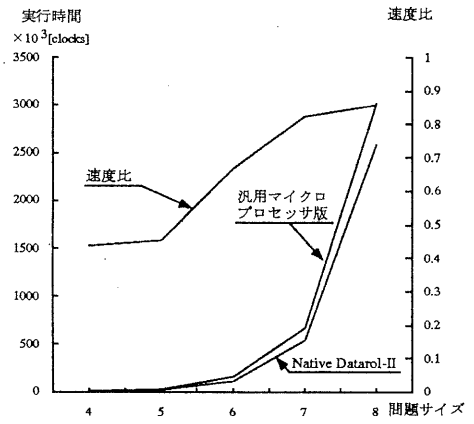

図 9: Queen の実行時間

また, 表2に Queen 実行時のキャッシュヒット率を示すが, fib の場合と同様に問題サイズが大きくなって RB のスワップが現れはじめると性能差が小さくなり, 問題サイズが大きいqueen(8) では 0.85 倍の性能となる.

問題サイズ	queen (4)	queen (5)	queen (6)	queen (7)	queen (8)
データキャッシュヒット率 [%]	81.42	78.50	75.06	74.10	71.89
RB スワップ時間 [%]	0.0	0.0	29.6	41.6	43.9

表 2: Queen のキャッシュヒット率

全体として Queen 程度の thread 長があれば実行時間は 2 倍以内におさまると考えられる. 実際には汎用マイクロプロセッサは高いクロックで動作させることができるため, 十分実用になるといえる.

5 結論

我々の研究室では細粒度 thread 処理に適した Datarol-II アーキテクチャを提唱してきた. しかし, Datarol-II は専用プロセッサを前提としているため設計コストが大きいことが問題であった.

本稿では, Datarol-II は汎用マイクロプロセッサに細粒度スレッド処理機構を簡単な外付け回路によって付加した構成で実現でき, 大幅なコストダウンが可能であることを示し, その性能予測を行なった.

この結果, thread 長が N-queen のプログラム程度であれば同一クロック速度においても実行時間は Native Datarol-II の 2 倍以下になることが予測できた. 実際には汎用マイクロプロセッサは高いクロックで動作するため, 汎用マイクロプロセッサを用いた Datarol-II は十分に実用的な速度で動作する. したがって, 専用プロセッサ設計によるコスト増大をまねくことなく, 細粒度スレッド処理に適したプロセッサエレメントを構成可能であることが確認できた.

参考文献

[1] M. Amamiya and R. Taniguchi, "Datarol: A Massively Parallel Architecture for Functional Language", Proc. SPDP, pp. 726-735, (1990)

[2] R. S. Nikhil, G. M. Papadopoulos and Arvind, "*T: A Multithred Massively Parallel Architecture", Proc. 19th ISCA, pp. 156-167, (1992)

[3] W. J. Dally, A. Chien, S. Fiske, W. Horwat, J. Keen, M. Larivee, R. Lethin, P. Nuth and S. Wills, " The J-Machie: A Fine-grain Concurrent Computer", Proc. 11th IFIP, pp. 1147-1153, (1989)

[4] 川野, 日下部, 谷口, 雨宮, "並列計算機 Datarol-II のプロセッサエレメントの構成", 情報処理学会研究会報告, 93-ARC-101, pp. 137-144, (1993)

[5] 児玉, 坂井, 山口, "データ駆動シングルチッププロセッサ EMC-R の動作原理と実装", 情報処理学会論文誌, 32, 7, pp. 849-858,(1191)

[6] "Pentium プロセッサデータブック", インテルジャパン株式会社