計算機アーキテクチャ　101－18
（１９９３．　８．　２０）

並列計算機Datarol-IIのプロセッサエレメントの構成

川野哲生　日下部茂　谷口倫一郎　雨宮真人
九州大学大学院総合理工学研究科

純粋な命令レベルのデータ駆動計算機では，各命令毎に行われる同期処理がネックとなり演算処理部への命令供給能力が制限されるといった問題がある．我々はこれまでデータフローモデルにレジスタ記憶を導入し最適化したDatarolアーキテクチャを提案してきた．今回はDatarolにスレッドの排他的実行機構を導入し，より最適化したプロセッサアーキテクチャを提案する．また，本プロセッサはレジスタの動的ロード機構，階層化されたオペランドメモリ，負荷制御機構等の特徴を持つ．

本稿ではDatarol-IIプロセッサと呼ぶ超並列計算機用のプロセッサエレメントの構成について述べ，また，ソフトウェアシミュレータを用い本プロセッサが効果的な細粒度マルチスレッド処理を実現できることを示す．

The design of Datarol-II processor

Tetsuo KAWANO, Shigeru KUSAKABE,
Rin-ichiro TANIGUCHI and Makoto AMAMIYA
Department of Information Systems Graduate School of Engineering Sciences,
Kyushu University
6-1, Kasuga-koen, Kasuga, Fukuoka 816 JAPAN
E-mail:{kawano,kusakabe,rin,amamiya}@is.kyushu-u.ac.jp

It has been pointed out that matching store mechanism in a pure instruction level dataflow computer requires complex and expensive hardware, and limits ability of injecting instructions into execution unit.

Datarol architecture, which we have proposed, removes redundant dataflow control by introducing a concept of by-reference data access. In this paper, we propose Datarol-II processor, which is optimized by introducing thread execution mechanism into Datarol architecture, in order to achieve more efficient instruction execution. This processor has dynamic register load mechanism, hierarchical memory system and load control system.

In this paper, we present an architecture of Datarol-II processor, showing by software simulation that it is able to execute fine grain programs in high performance.

1 はじめに

データ駆動型並列計算機は，プログラム中に内在する全ての並列性を実行時に自然に引き出すことができ，またプロセススイッチをハードウェアで高速に実行できることから超並列計算機アーキテクチャのベースとなるものである．また，高並列の処理を記述する上で有効であり，プログラムの検証性にも優れる関数型のプログラミング言語との適合性も高い．しかしながら純粋な命令レベルデータ駆動方式は以下のような問題点をもつ．

- 頻繁に発生する低レベルの通信によるオーバーヘッド
- 冗長なデータ複製によるオーバーヘッド
- マッチング・ストアに複雑で高価なハードウェアが必要
- 命令毎に行うオペランドチェックが演算処理部への命令供給速度を制限

我々の研究グループではDatarolと呼ぶ，データ駆動モデルを基にレジスタ記憶を導入し，冗長なフロー制御を削除し最適化した実行モデルの提案を行ってきた[1, 2]．今回の報告ではDatarolモデルに対し，複数命令のブロック化を行い，これを単位とした実行を行うことによりデータ駆動計算機の問題点を解決した並列計算機用のプロセッサエレメントの提案を行う．

本稿では，まず2章でDatarolアーキテクチャについて述べ，3章で新たに提案するDatarol-IIプロセッサの構成を示す．4章ではソフトウェアシミュレータによる本プロセッサの性能評価結果を示す．5章で関連研究との相違について述べ，最後に6章で本稿の結論を述べる．

2 Datarolアーキテクチャとその最適化

著者らはDatarolアーキテクチャを基に並列計算機Datarolマシンを設計し，計算機シミュレーションによりその性能評価を行ってきた．Datarolマシンのプロセッサエレメントは循環パイプラインにより構成される，プログラム中に十分な並列性があり，循環パイプラインが満たされた状態において高いスループットが得られることが確認された[3]．しかしながら，以下のような問題点も明らかになった．

- Datarolでは各命令毎にその継続命令の指定を行っている．これにより実行時にプログラム中の全ての並列性を自動的に抽出でき，また複数スレッドの命令が循環パイプライン上で動的にインターリーブされ，多重並行処理が効果的に行われる．しかしながら，並列度の低い逐次的なプログラムでは循環パイプライン上にバブルを生じ処理効率が低下する．
- Datarolでは純粋なデータ駆動に比べかなりの数のオペランドマッチングを削減できる．しかしながら，依然としてペアオペランドマッチ失敗時には演算処理部にバブルを生じる．

そこで，これら問題点を解決するため以下のようなDatarolアーキテクチャの更なる最適化を行う．

- 逐次的な処理に対する実行効率を上げるため，Datarolにスレッドの排他的実行機構を導入する．すなわち，逐次的なスレッド内の命令を命令メモリ内の連続したアドレスに配置し，スレッド内を従来のノイマン型と同様にプログラムカウンタを用い逐次的に実行する．
- オペランドデータに対する同期処理をスレッドを単位として行う．これにより同期処理部から演算実行部へはスレッドを単位としたディスパッチが行われ，同期処理部からの命令供給能力が向上する．

3 Datarol-IIプロセッサ構成

Datarol-IIプロセッサ(以下PEと略記)の構成を図1に示す．各PEは以下のような機能ブロックから構成される．

- **FU(Function Unit)**
  スレッド内の命令を逐次的に読みだし演算処理を行う．遠隔メモリアクセスや関数呼び出し等のメッセージの発行も行う．
- **CU(Communication Unit)**
  関数引数引渡し等のメッセージを受取り，メッセージ内のデータをメモリへ書き込みスレッド起動要求を発行する．
- **AC(Activation Controller)**
  ACはMM(Matching Memory)を持つ．MMは各インスタンス毎に8つの4ビットカウンタ(計32ビット)を保持しこれを用いたスレッド間の同期処理を行う．

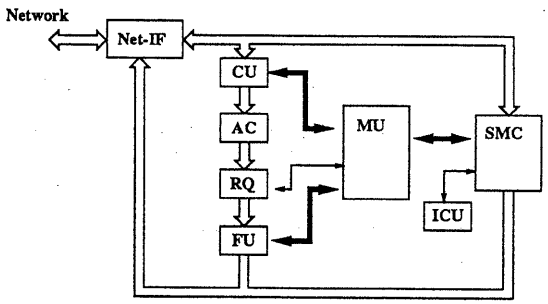

図 1: Datarol-II プロセッサの構成図

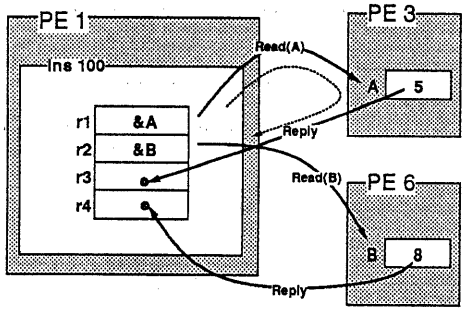

図 2: 遠隔メモリアクセスの例

- **RQ(ReadyQueue)**
  実行可能となったスレッドの情報を格納する.
- **MU(Memory Unit)**
  各インスタンス毎に割り当てられたレジスタファイルの内容を格納する.
- **SMC(Structure Memory Controller)**
  遠隔メモリアクセス等の要求に対する処理を行う.
- **ICU(Instace Control Unit)**
  PE 内の負荷状況を監視しインスタンス割り付けの制御を行う.

以下では図 2に示すような遠隔メモリアクセスを例に各部の動作を簡単に説明する.

1. PE1 の FU で遠隔メモリアクセス命令が実行され, 遠隔メモリアクセス要求メッセージが送出される. メッセージには読み出し先のアドレスおよび結果値書き込み先アドレス (PE 番号, インスタンス番号, レジスタ番号) が含まれる. また, この時結果値の書き込み先であるレジスタ[1]r3, r4(MU 内のインスタンス番号とレジスタ番号によりアクセスされるメモリセル) には結果値到着後に起動されるスレッドの起動情報 (同期カンウタ番号, 同期数, スレッド開始アドレス) が書き込まれる.
2. PE3,PE6 の SMC が遠隔メモリアクセス要求を受け取り, メモリ読み出しを行い結果パケットを送出する.
3. PE1 の CU で結果パケットを受け取り, パケット中に含まれる結果値書き込み先アドレスを取り出し MU へアクセスする. ここでは, まず MU の当該アドレスの内容, すなわち 1 で書き込まれたスレッド起動情報が読み出され, その後メッセージ内のデータが書き込まれる. ここで読み出されたデータはスレッド起動要求として AC へ送られる.
4. AC はスレッド起動要求を受け取り, MM を用いて同期を行う. この場合 r3,r4 の両方のパケットが到着した時, RQ へ実行可能なスレッド情報 (インスタンス番号, スレッド開始アドレス) が送られる.
5. 4 で RQ に書き込まれたスレッド情報が RQ の先頭に到着し, FU で現在実行中のスレッドが終了した時これが取り出され, スレッド開始アドレスが FU 内の IP(Instruction Pointer) にセットされる. FU では IP に従い IM(Instruction Memory) 内の命令を順次読み込み実行する. 各命令には terminate ビットがあり, これが 1 である命令がスレッドの終端命令である.

以下では Datarol-II プロセッサのメモリ構成, FU, SMC についてより詳しく説明する.

3.1 メモリ構成

一般に多重並行処理を行う上で各処理に対する環境の演算処理部への読み込み, あるいは他の処理へ切替える時に発生する環境の退避が必要となり, これがオーバーヘッドとなる. 細粒度の多重並行処理を効果的に実現するためにはこのオーバーヘッドの最小化が必要である.

Datarol では各関数インスタンス毎に固有のレジスタファイルが割り当てられ, 関数インスタンス

[1]Datarol では各関数インスタンスは固有のレジスタファイルを持つ. レジスタの値は MU 内に格納される

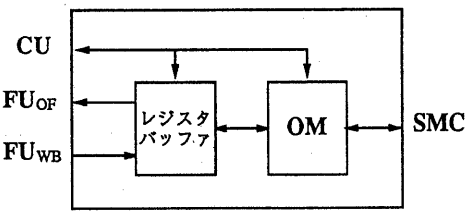

図 3: MU の構成図

内の処理はこのレジスタファイルを用いて行われる．しかしながら，演算処理部に各関数インスタンス用のレジスタファイルを全て用意するのはハードウエアコスト上困難である．そこで，Datarol-II プロセッサではレジスタファイルの内容を外部のメモリ OM(Operand Memory) に格納し，実行時に演算に必要なデータを演算処理部 (FU) に読み込み処理を行う．このとき，効果的な多重並行処理を実現するためにはレジスターメモリ間のデータ転送のオーバーヘッドの解消を行う必要がある．

Datarol-II プロセッサでは FU 内のレジスタと OM の間にレジスタバッファと呼ぶ高速のメモリが置かれる．MU の構成を図 3に示す．レジスタバッファはページ番号とページ内オフセットによりアクセスされる．各ページはそれぞれ各関数インスタンスのレジスタファイルに相当し，またページ内オフセットは各関数インスタンス内のレジスタ番号に対応する．

各関数インスタンスのレジスタファイルの内容は FU でその関数内のスレッドの実行が開始される以前に RQ からのリクエストによりあらかじめ OM からレジスタバッファへと読み込まれ，FU からのアクセスは全てレジスタバッファに対して行われる．また，CU からのアクセスは，レジスタバッファ内に存在するものについてはレジスタバッファへ，それ以外のものは OM へ直接アクセスする．

3.2 Function Unit

FU の構成を図 4 に示す．以下では，FU のレジスタ構成，パイプライン構成について説明する．

3.2.1 FU レジスタ構成

FU は数枚のレジスタセットを持ち，各レジスタにはレジスタ内容の存在ビットがある．各関数

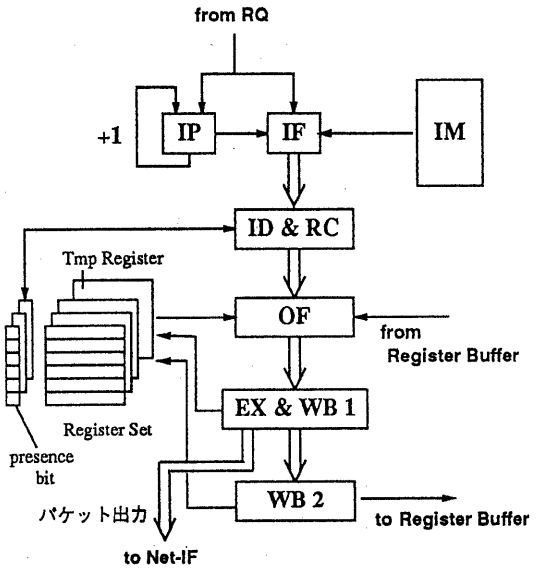

図 4: FU の構成

インスタンス内の処理はいずれか 1 つのレジスタセットを用いて行われる．レジスタバッファから FU 内のレジスタへの読み込みはレジスタ内容の存在有無により動的 (暗黙的) に行われる．すなわち，レジスタの読み出し時に存在ビットのチェックを行ない，存在ビットが 0 の場合レジスタバッファの対応するアドレスからデータを読み込む．またレジスタへの書き込み時にはレジスタバッファの対応するアドレスへも同時に書き込みを行う．また，FU にはテンポラリ用のレジスタも用意され，これは同一スレッド内命令間でのデータの受渡しに用いることができる．

3.2.2 FU パイプライン構成

FU は IF (Instruction Fetch), ID&RC (Instruction Decode & Register Check), OF (Operand Fetch), EX&WB1 (Execution & Write Back 1), WB2 (Write Back 2) の 5 段のパイプラインにより構成される．本パイプラインはレジスターレジスタバッファ間のデータ転送と命令実行とをオーバーラップして実行できるように設計されている．以下に各パイプラインステージでの動作を述べる．

- IF ステージ
  IP (Instruction Pointer) に従い IM (Instruc-

tion Memory) から命令を読み込む．このとき，読み出した命令がスレッドの終端命令であるとき，すなわち命令コード中のterminateフラグが1のときはRQから新たなスレッド情報を取り出しIPへセットする．終端命令でないときにはIPを1増加させる．

- **ID&RCステージ**
  命令デコードおよびレジスタの存在チェックおよび更新を行なう．レジスタ存在チェックは，各ソースレジスタについて行ない，この結果を次のステージに送る．またデスティネーションレジスタの存在ビットも1にする．また，call, rlink 等のレイテンシを伴う命令のデスティネーションレジスタについては存在ビットを0にする．これにより，結果の到着後に起動されるスレッドがこのレジスタを読み出そうとする時，必ずレジスタバッファから読み込まれるようになる．
- **OFステージ**
  RCステージの出力をもとにオペランドフェッチを行なう．レジスタ内容が存在している場合には該当するレジスタからオペランドデータを読み込む．また，レジスタ内容が存在していない場合はレジスタバッファへアクセスしデータを読み込む．レジスタバッファへのアクセスは各クロックに1度だけ許され，2オペランドの双方の読み込みが必要な場合は1インターロックがかかる．
- **EX&WB1ステージ**
  演算実行を行う．関数の引数引渡し等のメッセージもここで作られる．前ステージでレジスタバッファからの読み込みを行なった場合，ここでそのデータを該当するレジスタへ書き込む．
- **WB2ステージ**
  演算結果をデスティネーションレジスタへ書き込む．同時に，デスティネーションレジスタがテンポラリレジスタ以外の場合は，レジスタバッファへ結果データの書き込みを行なう．

ここで，OFステージとWB2ステージで，レジスタバッファへのアクセスが同時に発生する場合がある．この場合はパイプラインインターロックが発生しOFステージの処理が待たされる．

分岐命令は，すべてスレッドの終端命令とし，EXステージで決定された分岐先アドレスがスレッド起動パケットとして送り出される．

3.3 Structure Memory Controller

SMCではOMの一部を用いI-構造記憶による同期メモリアクセスを実現する．各メモリセルは2ビットのタグを持ち，これより各メモリセルの状態を以下のように示す．

00:Empty
　読み出し，書き込み双方ともに未到着
01:Suspended
　読み出しアクセスのみ到着
10:Full
　データ書き込み済み

Emptyセルに対する同期読み出しアクセスが到着した時，タグはSuspendedに変更され読み出しパケット内のリターンアドレスがメモリセルへ書き込まれる．Suspennded セルに対する書き込みパケットが到着した時，セルの内容が読み出されサスペンドされていた読み出しアクセスに対するパケットを返送する．そして当該メモリセルへの書き込みを行う．

また，SMCにはICU(Instace Control Unit)が付加され，インスタンスの割り付け，回収および負荷制御機構を実現する．SMCにインスタンス割り付け要求が到着した時，まずICUによりPEの負荷がチェックされる．PEの負荷が軽い時には直ちにインスタンスの割り付けが行われ，新たに割り当てられたインスタンス番号を結果パケットとして送り返す．PEの負荷が重い時にはインスタンス割り付け要求はペンディングされ，割り付け要求パケットはOM内に設けられたインスタンス割り当て待ちバッファへ格納され，再びPEの負荷が軽くなった時点でバッファから読み出され，インスタンス割り付けが行われる．

4 性能評価

我々はDatarol-IIプロセッサの性能評価のため，各クロックでのパイプライン動作を忠実に再現したソフトウェアシミュレータを作成した．以下では本シミュレーターを用いたDatarol-IIプロセッサの性能評価を行う．

ここでは例題としてフィボナッチ数を分割統治法により求めるプログラム(fib)，Nクイーンの全

解探索プログラム (queen) を用いる．これらのプログラムは細粒度のプロセス (インスタンス) を大量に生成するもので，細粒度プロセス間の高速なコンテクストスイッチ能力が必要とされる．また，fib は規則的なプロセス木を生成するのに対し，queen のプロセス木は不規則であり，人為的な負荷制御を行いにくい例題である．

以下ではまずスレッド化の効果と，レジスタ動的ロード機構の効果について，レジスタバッファのページ数が十分あり，レジスタバッファーOM 間のスワップ処理が行われない場合を仮定して評価する．次にスワップ処理を行った場合について，負荷制御の効果と併せて述べる．

4.1 スレッド化の効果

表1にフィボナッチ数を分割統治法により求めるプログラム，表2にクイーンの全解探索プログラムを Datarol-II プロセッサ単体を用いて実行した結果を示す．各命令毎に同期を行う純粋な命令レベルでのデータ駆動では，循環パイプライン中でデータの待ち合わせ処理部がネックとなる．例えば全実行命令中の 2 オペランド命令の割合を k とするとデータ待ち合わせ部は演算処理部に対し $(1+k)$ 倍のパケットを処理する必要がある．しかしながらデータ待ち合わせ部の処理はメモリアクセスにより実現されるため高速化が困難である．これに対しスレッドを単位とした実行を行う Datarol-II プロセッサでは表1,2に示すようにスレッド長が極端に短い fig の場合においても待ち合わせ処理部 (AC) への入力パケット数は命令実行数 (FU 稼働クロック) の半分程度，queen では半分以下とかなり軽減されていることが分かる．これから循環パイプラインを設計する上でボトルネックとなっていた AC の速度要求が緩和され，より高速な演算機の使用が可能となる．

表 1: fib(15) の実行結果

	クロック数	実行クロックに対する割合
実行クロック	22507	1
FU 稼働クロック	17761	0.79
AC パケット入力	9866	0.44
AC 同期処理	5918	0.26

表 2: queen(6) の実行結果

	クロック数	実行クロックに対する割合
実行クロック	60737	1
FU 稼働クロック	52334	0.86
AC パケット入力	24504	0.40
AC 同期処理	14304	0.24

4.2 レジスタ動的ロード機構

Datarol-II では FU 内のレジスタへのデータの読み込みは実行時のデータ存在チェック機構により動的に (暗黙的に) 行われる．図5，6に FU-レジスタバッファ間のデータ転送を明示的な命令により実行した場合と Datarol-II の動的ロード機構により行った場合について，その実行時間の内訳を示す．同図に示すように細粒度のマルチスレッド処理においては演算処理に対しデータ転送の割合が大きく，明示的な命令による実行ではオーバーヘッドとなることが分かる．一方動的ロード機構を用いた場合，データ転送と演算実行をオーバーラップして実行でき，これによりデータ転送のオーバーヘッドの多くを隠蔽することができる．ここで，動的ロード機構を用いた場合のロード数が用いない場合に比べ少ないのは，新たなスレッドの実行を開始した時にそのインスタンスに対するレジスタが FU 内に存在し，これによりレジスタバッファへのアクセスが削減されたためである．図7に queen(6) を Datarol-II プロセッサで実行した場合の演算実行部 (FU) の稼働率とレジスタバッファへのアクセス頻度の時間変化を示す．同図からも Datarol-II では演算実行部が高い稼働率を保ち，そ

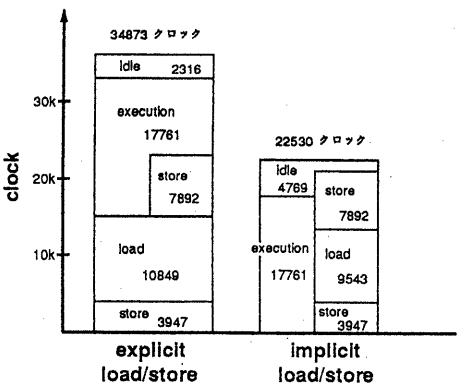

図 5: fib(15) の実行時間

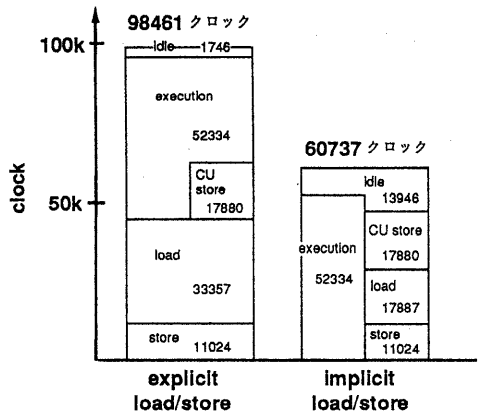

図 6: queen(6) の実行時間

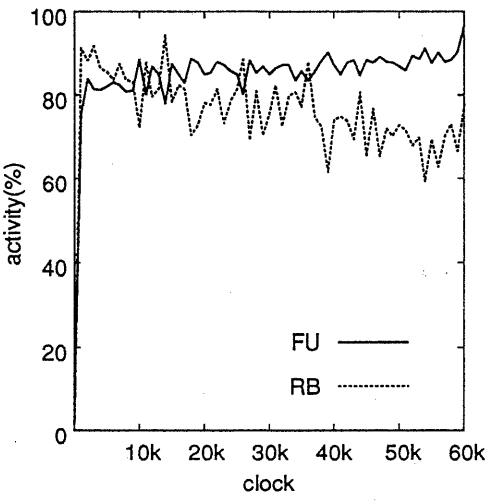

図 7: FU およびレジスタバッファの稼働状況

れに並行してレジスタバッファへのアクセスが行われていることが分かる．以上から Datarol-II プロセッサでは演算実行部およびレジスタバッファのデータ転送能力を最大限に利用できることが確認できた．

4.3 スワップ処理と負荷制御

次にレジスタバッファのページ数が限られ OM との間でスワップ処理を行った場合について述べる．図 8にレジスタバッファのページ数を変化させ，それぞれ負荷制御[2]を行わない場合，行った場合について，スワップ処理を行わない場合を基準とした実行時間比を示す．一般にデータ駆動方式による多重並行処理では参照の局所性はほとんど無く，レジスタバッファ－OM 間のスワップ処理はスラッシングを招きやすく，性能低下の原因となる．そこで Datarol-II では負荷制御機構により必要以上の並列展開を抑止しこれにより処理に局所性を持たせ，レジスタバッファ－OM 間のスラッシングの発生を抑える．図 8から負荷制御を行わない場合はスワップ処理が頻繁に発生しこれが実行クロックの増加の原因となっていることが確認できる．レジスタバッファには高速なメモリを使用する必要があり，あまり大きな容量を用意することは困難である．負荷制御を行うことによりスワップ頻度が抑えられ，ページ数が 64 程度で，レジスタバッファに十分なページ数がある場合に近い性能が得られている．1 レジスタセットあたり 16 ワードとして 64 ページで 4kbyte であり，この程度ならプロセッサをシングルチップ化し，チップ内にレジスタバッファを内蔵することのできるサイズである．

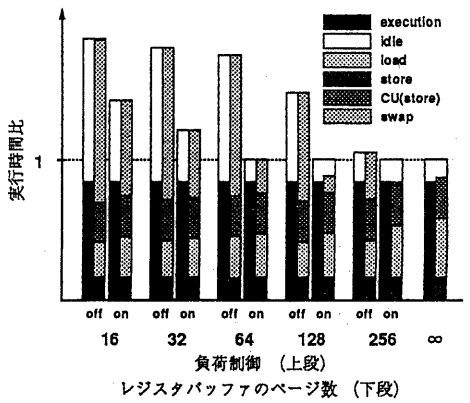

図 8: スワップ処理を行った場合の queen(6) の実行時間

[2] Datarol-II での負荷制御機構は，前のバージョンである Datarol マシンとほぼ同様である (文献 [3, 4] 参照)．この詳しい実現方式については別の機会に述べる

5 関連研究

データ駆動型計算機の最適化として，複数の命令を1まとまりのブロックとし，このブロックを単位とした実行を行うモデルがいくつか提案されている．ここでは，その主なものとの相違について述べる．

EM-4 [6]

電子技術総合研究所において開発された高並列データ駆動計算機 EM-4 では強連結枝モデルにより強連結ブロックを定義する．EM-4 では強連結ブロック内の命令を適切にスケジューリングし，これらの命令間の値の授受にレジスタを用いることによりデータ待ち合わせやデータのコピーのオーバーヘッドの軽減を行っている．しかしながら，この最適化は強連結ブロック内に限られたものである．これに対し Datarol-II ではインスタンス内の命令間のデータの授受を全てレジスタを介して行い，スレッド (命令ブロック) を越えてのデータ待ち合わせやデータコピーに対しても最適化を行っている．また， EM-4 では同期処理部と演算実行部が直接結合されているため同期失敗時に演算実行部にパイプラインバブルを生じる．これに対し Datarol-II では同期処理部 (AC) と演算実行部 (FU) の間にバッファ(RQ) を設けることにより，同期処理部と演算実行部との間の負荷のギャップを吸収する．

*T [8]

MIT で開発されている並列計算機*T は同期処理部と演算実行部にそれぞれ専用のプロセッサを用いる方式で，これらの間をバッファにより結合する．演算実行部には従来のノイマン方式によるマイクロプロセッサを用いる．このことから，効率的な実行を行うためには，かなり長いスレッドを抽出する必要があり，コンパイラへの負担が大きいと思われる．細粒度スレッドの効率的実行を目指す EM-4 や Datarol-II とは異なる点である．

6 結論

本報告では Datarol-II プロセッサと呼ぶ並列計算機用のプロセッサエレメントの提案を行った．本プロセッサはデータフローモデルを最適化した Datarol アーキテクチャを基に設計され，スレッドの排他的実行機構，レジスタの動的ロード機構を備える独自のパイプライン設計，階層化されたオペランドメモリ，負荷制御機構等の特徴を持つ．本プロセッサについてソフトウェアシミュレータによる評価を行い，効果的な細粒度マルチスレッド処理を実現できることを確認した．

参考文献

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

[2] 立花, 谷口, 雨宮, "データフロー解析による関数型言語 Valid のコンパイル法— Datarol プログラムの抽出アルゴリズム —", 情報処理学会論文誌, 30, 12, pp.1628-1638, (1989)

[3] 星出, 園田, 谷口, 雨宮, "並列計算機 Datarol マシンにおける資源管理と負荷制御方式", 信学技法, CPSY91-7, pp.25-32, (1991)

[4] S.Kuskabe, T.Hoshide, R.Taniguchi and M.Amamiya, "Parallelism Control and Storage Management in Datarol PE", Proc. IFIP world congress, Vol.1, pp.535-541, (1992)

[5] R.Taniguchi, T.Kawano and M.Amamiya, "A Distributed-Memory Multi-Thread Multiprocessor Architecture for Computer Vision and Image Processing: Optimized Version of AMP", Proc. 26th ICSS, Vol.1, pp.151-160, (1993)

[6] 児玉, 坂井, 山口, "データ駆動型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, (1991)

[7] William J.Dally, Andrew Chien, Stuart Fiske, Wakdemar Horwat, John Keen, Michael Larivee, Rich Lethin, Peter Nuth and Scott Wills, "The J-Machine: A Fine-grain Concurrent Computer", Proc. 11th IFIP, pp.1147-1153, (1989)

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