九州大学学術情報リポジトリ
Kyushu University Institutional Repository

Datarol マシンの資源管理方式

日下部, 茂
九州大学大学院総合理工学研究科情報システム学専攻

星出, 高秀
九州大学大学院総合理工学研究科情報システム学専攻

谷口, 倫一郎
九州大学大学院総合理工学研究科情報システム学専攻

雨宮, 真人
九州大学大学院総合理工学研究科情報システム学専攻

https://doi.org/10.15017/17253

出版情報：九州大学大学院総合理工学報告. 13 (4), pp.393-400, 1992-03-01. 九州大学大学院総合理工学研究科
バージョン：
権利関係：

九州大学大学院総合理工学研究科報告
第13巻 第4号 393-400頁 平成4年3月

Engineering Sciences Reports, Kyushu University
(KYUSHU DAIGAKU SOGORIKOGAKU KENKYUKA HOKOKU)
Vol. 13, No. 4 pp. 393-400 MAR. 1992

Datarolマシンの資源管理方式

日下部 茂*・星出 高秀**
谷口 倫一郎*・雨宮 真人*

(平成3年11月30日 受理)

Resource Management of Datarol Machine

Shigeru KUSAKABE, Takahide HOSHIDE, Rin-ichiro TANIGUCHI
and Makoto AMAMIYA

Datarol machine can execute functional programs with high performance by supporting an effcient execution environment of tiny concurrent processes based on dynamic dataflow architecture. In the Datarol machine, a set of registers is allocated at run time to each function instance as a working memory, in order to share the operand data and eliminate redundant dataflows. In this paper, we propose a register file management mechanism that can allocate logically more register files than physically limited number of register files in order to support efficient multi-processing environment. We also evaluate this mechanism by simulating its performance.

1 はじめに

関数型言語は記述の簡潔性，プログラムの機械的な変換や検証の容易さなどに加え，副作用を持たない純関数的なプログラムの構造という高並列の処理を記述する上で魅力的な性質を持つ．Datarolマシンは，動的データフロー方式に基づく命令レベルのコンテキスト・スイッチにより多重処理機構を実現し，関数型プログラムのリダクション過程において生じる多数の小粒度プロセスを効率良く多重並行処理することによって，関数型プログラムを高速・並列に実行する[1)5)]．

データ駆動原理に基づく並列計算機では，データ依存関係に従って，問題中の並列性を最大限に抽出しプログラムを実行する．しかし，問題中の並列性を無制限に展開しても，実際のハードウェアで処理可能な並列度は有限であり，インスタンス（フレーム）用の作業領域や，トークンキューなどの計算機資源を浪費してしまうだけで性能向上は期待できない．そのため，データ駆動型の並列計算機では，並列展開の制御などの実行管理を行なう必要がある[2)]．

*情報システム学専攻
**情報システム学専攻修士課程

Datarolマシンも，データ駆動原理に基づいて問題中の並列性を抽出するため，プロセッサが持つハードウェア資源を管理しながら並列実行能力に見あっただけの仕事を供給する制御が必要である。Datarolマシンでは関数適用毎に生成される実行インスタンスをプロセスとしてとらえ，関数インスタンス単位で並列展開を制御し，計算機資源を管理する．

Datarolマシンが実行するDatarolプログラムは，メモリ上でのデータ共有概念によってデータフロー・プログラムから冗長なペアオペランドの存在チェックやゲート演算などのデータフロー制御を取り除いて最適化したマルチスレッド・コントロールフロー・プログラムである[3)]．オペランドデータを共有するため，インスタンス毎に固定サイズのメモリブロックを割り付けるが，この割り付け効率がDatarolマシンの性能を大きく左右する．

Datarolマシンでは，次のような実行制御・資源割付管理を行なう．

1. 関数インスタンス内の資源割付管理

関数インスタンス内の資源管理として，オペランドデータを保持する作業領域であるレジスタファイルの使用スケジューリングを行なう．コンパイラによるデータ依存解析によって，実行順の抽出，レジスタセ

ルの割り付けを静的に決定することができる[3].

2. 関数インスタンス間の並列展開・資源割付管理

この問題は PE (Processing Element) 内および，PE 間での並列展開・資源割付管理の問題に分けられる.

i. PE 内管理

- PE 内に展開する関数インスタンスの数をある閾値に保ち，パイプライン充足率を落さず，かつ計算機資源の浪費を防ぐよう並列展開を制御する.
- 各 PE が持つ物理的レジスタファイルを仮想化し動的に割付管理することで，物理的に限られたハードウェア資源で効率良く超多重並列処理を実現する.

ii. プロセッサ割り付け管理

PE を結合するネットワークに，負荷分散機構を設け，軽負荷のプロセッサに関数インスタンスを割り付ける.

本稿では，上述の問題のうち特に 2.i. の PE 内資源管理方式について述べる. 2 節では説明のため Datarol の抽象的実行モデルについて述べる. 3 節では Datarol マシンが採り入れた計算機資源の管理方式の戦略について述べる. 4 節ではその管理方式のうちレジスタファイルの実行時割り付け管理の実現法と，その割り付け管理を支援するためにコンパイラが生成するコードについて述べる. 5 節ではシミュレーションによってその資源管理方式を評価する. 6 節では，シミュレーション評価から得られた結果をもとに，本方式で問題となる点を考察しその解決法について述べる.

2 Datarol 実行モデル

Datarol モデルの基本理念は by-reference メカニズムにより冗長なデータフロー処理を削除するものである. 共有すべきデータトークンに対しては，by-reference メカニズムを用い，共有する必要がないデータや Boolean データとゲート制御シグナルに対しては，by-value メカニズムを用いる.

関数適用命令実行時には，関数の実行インスタンスが自 PE 内，またはネットワークを通じて他 PE 内に生成される. このとき各関数インスタンスごとにレジスタファイルというメモリブロックを割り付け，共有するデータへのアクセスはインスタンス名（レジスタファイル名）とその中での局所アドレス（レジスタ番号）によって行なう. そのレジスタ番号はコンパイル時に静的に決定することができるが，インスタンスと物理的レジスタファイルの対応は実行時に決定される. 共有するデータを生成する演算は実行が終了した時点で直ちにその結果データを自インスタンスのオペランドメモリに書き込む. 例えば，**Fig. 1** において y δ z なので（z は y にデータ依存しているので），演算ノード w ではオペランド z が到着したとき，オペランド y の存在は保証され，オペランド y の到着を確認することなく直ちに演算が駆動される. このようにして，オペランドマッチング操作の冗長なものを削除しオー

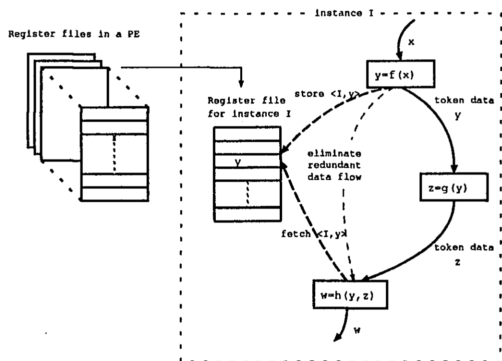

Fig. 1　Datarol computation model

```
function F(x_f,...)=
    ...
    G(p_g,...)
    ...
    H(p_h,...)
    ...  ;

function G(x_g,...)=
    ...
    I(p_i,...)
    ...  ;

function H(x_f,...)=
    ...
    J(p_j,...)
    ...
    K(p_k,...)
    ...  ;
```

Fig. 2　Example program

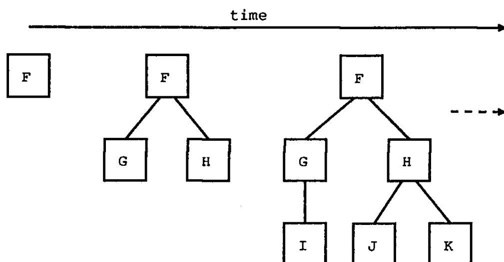

Fig. 3　Process tree

バヘッドを軽減する[1)5)].

Datarol では関数適用時に新たな関数インスタンスを生成し，各インスタンスは並列プロセスとして実行される．例えば **Fig. 2** に示したプラグラムでは，実行時には **Fig. 3** のようなプロセス木が生成される．（上下関係にあるプロセスのうち上が呼び側のプロセス。実際はプロセス木の構造は動的にしか定まらない。）プロセス木の葉にあたるインスタンスだけでなく，すべてのインスタンスは実行可能で，例えばＦとＧは並列にＩとＪは並列に（実際は命令レベルでインターリーブされて）計算が進む．

このように，実行時に生じる多数のインスタンスを，データ依存関係だけによって無制限に展開すると，並列の爆発により計算機資源の浪費，枯渇につながる．物理的に有限であるハードウェアで効率の良い多重並行処理を実現するためには，計算機のハードウェア能力に見合った並列展開制御や，オペランドデータを保持するメモリの各インスタンスへの割り付け方式が重要な研究課題となる．

3　並列展開・資源割付管理

Datarol マシンでは，関数起動ごとに活性化されるインスタンスを単位として並列展開を制御し計算機資源の割付を管理する．本節では，管理単位であるインスタンスの状態の定義と，その状態間の遷移について述べ，Datarol マシンの並列展開・資源割付管理戦略について述べる．

3.1　インスタンス状態

Datarol マシンでは関数適用毎に，実行環境を持つ実行体であるインスタンスを生成し，その関数の計算が終了した時点でインスタンスを解放するということを繰り返してプログラムの実行を進める．**Fig. 4** に資源管理を行なわない時のインスタンスの状態とその遷移を示す．

まだ起動される前の状態で，何の実行環境も持たない状態を本稿では非割付（null）状態と呼ぶ．親インスタンスがインスタンス生成命令を実行すると，非割付状態にあったインスタンスに論理的インスタンス名とオペランドデータを保持するためのレジスタファイルが割付けられる．このようなインスタンスの状態を本稿では割付（allocated）状態と呼ぶ．この場合何ら展開制御などを行っておらず，割付状態のインスタンスの論理名と割付けられた物理的レジスタファイルは１対１に対応する．割付状態のインスタンスはインスタンス解放命令を実行すると論理名とレジスタファイルを解放し非割付状態に戻る．

3.2　インスタンス並列展開の制御

Datarol マシンは，プログラムに内在する並列性を自然に抽出して実行するが，プロセッサ内で並列実行可能なインスタンス数は，高々パイプラインの段数程度である．そのため計算機ハードウェアの処理能力を越えてプログラム中の並列性を無制限に展開してもインスタンス実行領域やトークンキューなどのハードウェア資源を浪費するだけである．

この問題を解決するために Datarol マシンでは計算機ハードウェアの処理能力に応じた閾値（Nt）を設定し，割付状態のインスタンス数（Nal）が閾値に達している場合は，Nal が Nt を下回るまで新たなインスタンスの起動を遅延（pending）する（**Fig. 5** 参照）．起動を遅延するインスタンスには論理的インスタンス名だけを与え，物理的なレジスタファイルは与えない．Nal<Nt となったときに起動を遅延したインスタンスにレジスタファイルを与え実行を起動する．関数起動の遅延制御は，関数型のプログラムの再帰構造をしたプロセス木の葉プロセスから効率良く処理されるようスタックを用いた深さ優先制御を行なう．幅方向優先にインスタンスを展開すると，分割統治法などを用い

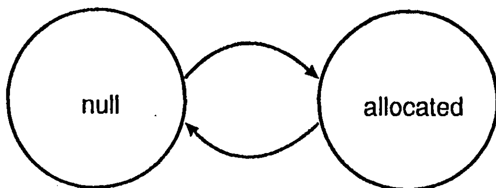

Fig. 4　States of instances and transitions between them

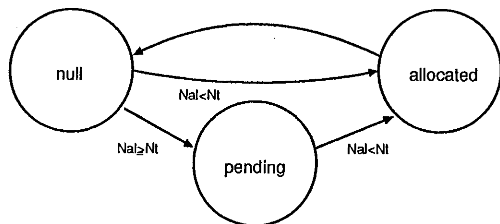

Fig. 5　States of instances and transitions between them under parallelism control

たプログラムの実行では，プログラム終了よりかなり早い段階で全インスタンスが展開されてしまうので，必要となるハードウェア資源量はかなり大きなものとなる．Datarol マシンの並列展開方式では，幅方向に Nt だけの並列性を持つ深さ優先制御が実現でき，計算機資源を浪費することなく効率良くプログラムを実行できる．

3.3　レジスタファイルの割付管理

割付状態にあるインスタンスはさらに次のように活性化 (active) 状態と休止 (suspended) 状態に分類でき，実行時にはこの2つの状態を繰り返す．前者は，処理可能なトークンが自インスタンス内に存在する状態で，後者は計算途中で一時的に自インスタンス内にトークンが存在しなくなり，他インスタンスからトークンをもらうまで自身内に処理可能なトークンを持たない状態である．休止状態インスタンス内では，計算が進行しないためレジスタファイルがアクセスされることはない．そのため，実行時に次のようなレジスタファイルの割付管理を行ない，レジスタファイルの有効利用をはかる．新たなインスタンスを活性化する時に，空いているレジスタファイルがなく，休止状態にあるインスタンスが存在すれば，休止状態にあるインスタンスの内容を外部メモリに退避 (swap) させ，空いたレジスタファイルを他のインスタンスに割り付ける．この資源管理機構が理想的に動作した場合，実行木の中途に位置する結果待ちのインスタンスは次々と退避され，深さ方向優先の並列展開制御も促進されることになる．

このような資源管理下でのインスタンスの状態とその遷移を **Fig. 6** に示す．

4　資源割付管理の実現方式

この節では，前節で述べた並列展開・資源割付管理方式のうち，資源割付管理方式に重点をおき，実現法およびその管理方式を支援するためにコンパイラが生成するコードについて述べる．

以下に，インスタンスの休止状態の検出方法，レジスタファイルのスワップ管理の実現方式，その管理方式を支援するためのコンパイラのコード生成について述べる．

4.1　インスタンスの休止状態の検出

インスタンスが休止状態に陥る場合としては，

1．関数適用などで子インスタンスにトークンを渡し，結果を待っている場合
2．関数本体内で呼び側からの一部の引数が未到着の時に，部分計算を行なった場合
3．書き込みの行なわれていないノンストリクトな構造データにアクセスを試みている場合

などがある．我々は，これらのうち，1．の関数適用に起因する場合が休止状態にある時間が最も長いと考え，関数適用による休止状態の検出を行なう．

休止状態を説明するための抽象的な Datarol グラフを **Fig. 7** に示す．例えば，(1)部分グラフG（関数適用を持たない）に対応する部分の計算が終了→(2)ノードUにそのトークンが到着し，オペランドマッチングに失敗→(3)関数適用Fに引数を全て渡す，という順序

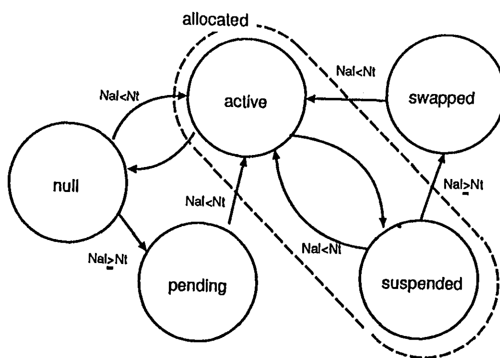

Fig. 6　States of instances and transitions between them under parallelism control and resource management

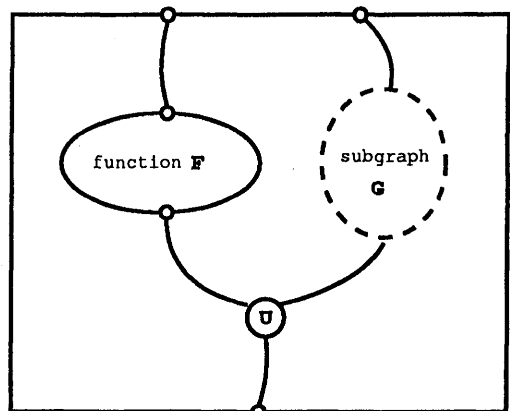

Fig. 7　Abstract Datarol graph for the explanation of suspended instance

で計算が進めばFから結果が返されるまでこのインスタンスにはトークンが存在せず，休止状態にある．また，(1')関数適用Fに引数を全て渡す→(2')部分グラフGに対応する部分の計算が終了→(3')ノードUにそのトークンが到着し，Fからの結果待ちのためオペランドマッチングに失敗，という順序で計算が進んでもFから結果が返されるまでこのインスタンスは休止状態にある．この場合，Gの計算終了より早くFから結果値が返されれば，このインスタンスが休止状態になることはない．

インスタンスの休止状態を検出するために以下のような検討を行なった．

- **その実行により休止状態になる演算ノードは静的には解析できない**

  Datarol プログラムの実行順は半順序関係でしか定まらず，一般にその実行によりインスタンスが休止状態になる命令を静的に一意に決定できない．実行順によっては，休止状態にならない場合もあり，実行時に動的に監視する必要がある．

- **全ての演算ノードでトークンの増減を調べる必要はない**

  インスタンスの休止状態を検出する方法としては，命令の種類によってインスタンス内のトークンの増減を監視し，トークンの数が0になった時を休止状態とするものが提案されている[2]．しかし，我々が検出したいのはトークンの数ではなく，トークンの有無なので例えば **Fig. 7** の部分グラフG中でトークンがいくつに増減するか正確に知る必要はない．我々は文献(6)で提案されているインスタンスの終了を検出する方法を応用する．その実行によりインスタンスが休止状態に陥る可能性を持つ演算ノード（以下終端命令と呼ぶ）のプログラム上の位置は，コンパイル時に解析可能であるため（例えば関数適用Fに引数を渡すノードやノードU），それらのノードをコンパイル時に登録しておき，その登録された命令だけを監視することで休止状態を検出できる．

以上のような考察より，本資源管理方式では次のようにしてインスタンスの休止状態を検出する．

1．ある時点より後に実行される終端命令の数を tc (token counter) と呼ぶカウンタが保持
2．終端命令が叩かれる度に tc の値をディクリメント
3．tc が0になればインスタンスは休止状態

4.2 コード生成

本節では，レジスタファイルの管理を支援するためインスタンスの休止状態を検出するためのコード生成アルゴリズムの概要について述べる．コンパイラで解析すべき主要事項は，次の通りである．

1．終端命令の抽出
2．最初に設定すべき tc の値の決定
3．inctc 命令の挿入

これらの処理について，**Fig. 8** の（説明に必要な部分だけを抜き出した）マージソートプログラムの例を用いて説明する．

処理1において終端命令は関数適用に関するものを描出するが，Datarol プログラムは関数適用に関して call, link, rlink の3つの基本演算を持つ．演算（w call f）は関数 f を活性化し，結果として活性化された関数 f のインスタンス名が返される．演算（link i w x）はアーギュメント x を関数インスタンス w の第 i エントリに引き渡す．演算（z rlink j w）は実行結果を受けとるため，関数インスタンス w に自分のインスタンス名と受け取り先の情報を送る．関数適用の結果値は z で参照される．

関数適用に着目した場合，終端命令になる可能性の

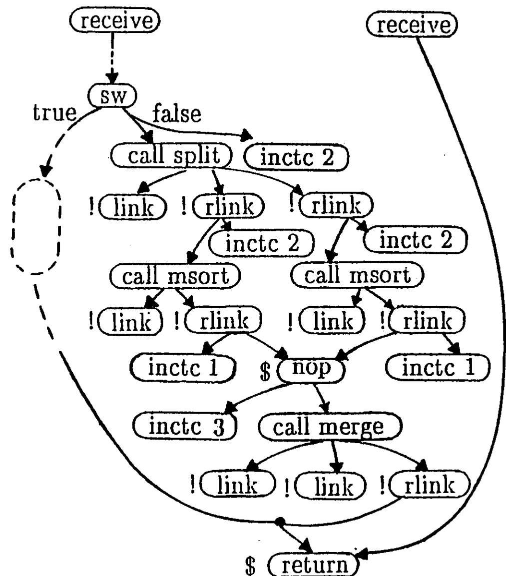

Fig. 8　Example of code generation to support resource management

ある命令としては

(a) link, rlink 命令

(b) rlink 命令を先祖に持つ（rlink 命令にデータ依存する）2オペランド命令

がある。(a)の命令については全てが終端命令となるが，(b)の命令についてはデータ依存関係の解析に基づいて，終端命令かどうか決定される．

Fig. 8 の例では全ての link, rlink 命令が終端命令であるので tc の1減らすための印！をつける．また return 命令は先祖に関数適用を持つ2オペランド命令で，この場合終端命令となりペアオペランドのマッチングに失敗した時に tc を1減らす印$をつける．

処理2では，インスタンスの割り付け時（call 命令実行時）に，設定すべき tc の初期値を決定する．その値は，終端命令のうち，親のどちらかの先祖に終端命令を持たないものの数である．

処理3では，実行時に必要な明示的カウンタ値の調整を行なう命令を挿入する．カウンタ値を調整する必要があるのは，次のような場合である．

- ある休止状態から復帰した後に，再び休止状態に陥る可能性がある場合．例えば **Fig. 8** の例で，関数 split の適用による休止状態から復帰した時点では tc の値は0であり，関数 msort に起因する休止状態を検出するために tc の値を調整しておく必要がある．
- 条件分岐命令の分岐先によって異なる個数の引数を持つ関数を適用する場合は，分岐先によって終端命令の数も異なるため tc の値も調整する必要がある．

4.3 インスタンス制御ユニット

休止状態インスタンスのデータも外部メモリへスワップアウトし，再開時に外部メモリからデータをスワップインする，といったオペランドメモリ管理の処理は，通常の演算命令と並列して実行させることが可能である．そのような管理を司るインスタンス制御ユニット（ICU）を，FU 内部において演算ユニット（ALU）と並列に配置することでスワップ処理と一般の演算を並行に実行し，スワップ処理のオーバヘッドが実行時間に影響を与えないようにする（**Fig. 9** 参照）．

ICU には，インスタンス名とその状態，tc 等の情報を登録しておく Register-File-Table (RFT)，空いているレジスタファイルを示す Free-Register-File-Table (FRFT)，サスペンドしたインスタンス名を入れておく Suspended-Instance-Queue (SIQ) を用意する．パイプライン上を流れるデータは，実際の物理的レジスタファイル名を識別子として持ち，一方パイプラインの外ではデータは論理的なインスタンス名を識別子として持つ．インスタ間でデータをやりとりする場合は，ICU で論理的なインスタンス名と物理的レジスタファイル名を対応づける．

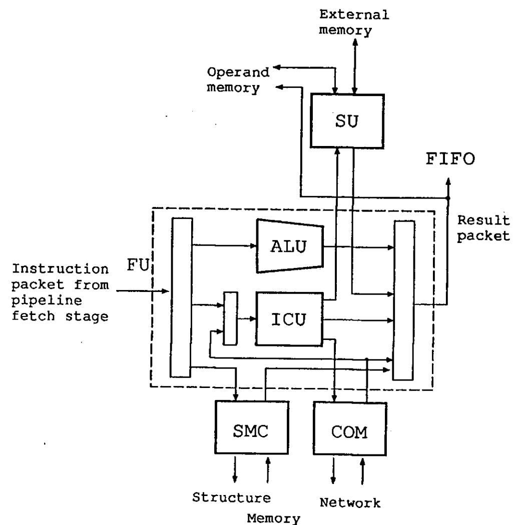

Fig. 9　ICU (FU organization)

レジスタファイルのスワップ管理をするため，関数適用時には次の動作を行なう．

関数起動時　空いているレジスタファイルがない場合には SIQ からサスペンド状態のインスタンス名を取り出し，そのインスタンスのデータを外部メモリにスワップアウトし，空いたレジスタファイルを新たなインスタンスに割り当てる．

結果返答時　値の返し先インスタンスがスワップアウトされている場合は，関数適用時と同様にしてレジスタファイルを確保して，そこへ返し先インスタンスのデータをスワップした後，値を返す．

5 資源割付管理方式の評価

レジスタファイルのスワップ管理が実行時間に与える影響を，レジスタファイル数，レジスタファイル1枚当たりのレジスタセル数をパラメータとしてソフトウェアシミュレータで実験し検討を行なう．シミュ

/Number of PEs	1	2	4	8	16	32	64
64 registers×64	1.96	2.14	2.22	1.95	1.68	1.45	1.35
64 registers×32	4.55	4.03	3.98	3.17	3.42	3.08	2.65
64 registers×16	―	6.93	6.84	6.51	6.03	5.40	4.82
32 registers×64	1.24	1.31	1.34	1.27	1.22	1.15	1.10
32 registers×32	2.15	2.05	2.10	2.01	1.90	1.80	1.64
32 registers×16	―	3.60	3.59	3.48	3.24	2.96	2.65
16 registers×64	1.06	1.07	1.08	1.08	1.06	1.06	1.02
16 registers×32	1.16	1.26	1.32	1.32	1.28	1.25	1.22
16 registers×16	1.79	1.93	1.96	1.96	1.87	1.79	1.63

レーションには，不規則なプロセス木を生成する例題として，N-queen の全解問題を解くプログラム（N＝8）を用いる．

Table 1 にスワップ処理を行なわない（並列展開の制御は行なうがレジスタファイルは必要なだけ用意される）ときの実行時間を1とし，オペランドメモリのスワップ管理を行なったときの実行時間比を示す．

この表より，PE 内のレジスタファイル数が少なく，レジスタファイル1枚あたりのレジスタ数が多い時ほどオーバヘッドは無視できなくなることがわかる．スワップ処理と他の演算は並行して実行可能であるが，レジスタファイル数が少ない場合物理的な資源数に比べ論理的に割り付けるレジスタファイル数が増加し，スワップ処理の回数も多くなるため，通常の演算実行では隠せないほどのオーバヘッドが出ている．

レジスタファイル1枚あたりのレジスタ数が多い場合，1回のスワップ処理に要する時間が長いため次のようなことが考えられる．現方式ではインスタンス活性化の処理を開始した時点で Nal の値をインクリメントしているが，実際にそのインスタンスで計算が開始されるのはスワップ処理が終了してからである．このため Nal と実際に計算が行なわれているインスタンスの数に差が生じ並列展開制御がうまく機能せず FU 稼働率が低下する可能性がある．

レジスタファイル1枚当たりのレジスタセルが少なければ，論理的に割り付けるレジスタファイルと物理的なレジスタファイルの比が大きくとも，レジスタファイル管理のオーバヘッドは目立たなくなっている．

6　本方式の問題点とその解決法

本節では，前節での評価をもとに本資源管理方式の問題点とその解決法について考察する．

オペランドメモリの割付管理のオーバヘッドを削減するためには，次の3つ問題点：

1．スワップ回数の削減
2．スワップ処理時間の短縮
3．スワップ処理方式の改善

の解決が有効であると考えられる．以下にそれぞれの解決法を考察する．

3.1　スワップ回数の削減

現在，関数適用に際してコンパイラが出力するコードでは，関数本体内で，先行評価によって可能な限り部分計算を進める狙いで，関数起動命令の call 命令を（引数データの存在に関係なく）無条件に発火させるようになっている．そのため，実行時には活性化されたものの，引数が到着するまで無駄にレジスタファイルを割り付けられているインスタンスが存在する場合がある．このような無駄な割付けは，利用可能なレジスタファイルが減少しスワップ処理の回数が増加する原因と考えられる．

コンパイラで必須引数の解析を行ない，必須引数到着まで関数起動を行なわないようなコードを生成すれば無駄なインスタンスの活性化を削減でき，資源割り付けのオーバヘッドも削減可能であると考えられる．先行評価による利点と，レジスタファイル管理のオーバヘッドとの関係を明らかにし，適切な関数起動のスケジューリングを行えば，効率の良い資源管理を行な

うことができる.

6.2 スワップ処理時間の削減

関数型プログラムの実行では，プロセスの粒度は低いため，レジスタファイル1枚あたりのレジスタ数を削減し，スワップ処理に要する時間を短縮することが可能である．しかし，問題によっては，1つの関数を実行するために必要なレジスタ数がレジスタファイル1枚あたりのレジスタ数をオーバーフローすることが考えられる．このとき，コンパイラによって複数のインスタンスに自動的に分割することで対応するが，実行時にはインスタンスを1つ（場合によってはそれ以上）余分に生成することになる．レジスタ数削減と，オーバーフロー時の余分なインスタンス処理のオーバヘッドの関係はトレードオフの関係にあり，最適なレジスタ数の決定は今後の課題である.

6.3 スワップ処理方式の改善

レジスタファイルの取得に要する時間を短縮するための方法として，以下に述べるようなスワッピングの先行処理がある．現方式では休止状態のインスタンスが存在しても，レジスタファイルが不足するまでスワップ処理は行なっていない．しかし，サスペンド状態のインスタンスが検出された時点でスワップアウトを行っておけば，レジスタファイルが不足した時即座にレジスタファイルに新しいインスタンスのデータを上書きでき，スワップ処理のオーバヘッドを隠すことができる.

今後は，上記の手法をコンパイラ，シミュレータに取り入れ，評価を行なっていく予定である.

7 結　論

関数型プログラムを高速に超多重並行処理する Datarol マシンの計算機資源管理方式と，それを支援するコンパイラのコード生成について述べた.

Datarol アーキテクチャはメモリを導入することにより，従来のデータフローアーキテクチャの欠点を改善する．そのためオペランドメモリの効率の良い割り付け方式が必要不可欠である．本資源管理方式では，全ての命令でトークンの増減を監視することなく，静的に検出不可能なインスタンスの休止状態を実行時に検出し，休止状態のインスタンスのデータをスワップアウトすることで，物理的に限られた数のレジスタファイルを，物理的にそれ以上の数のインスタンスに割り付け，効率の良い多重処理環境を実現する.

今後は，資源管理により生じるオーバヘッドの削減をはかり，本方式の有効性を示す.

参 考 文 献

1) 雨宮 “超並列多重処理のためのプロセッサアーキテクチャ”，情報処理学会「コンピュータアーキテクチャ」シンポジウム，(1988).

2) 武末 “データフロー型計算機の負荷制御方式”，電子情報通信学会論文誌，Vol. J70-D No. 10 p.p.1878-p.p.1889 (1987).

3) 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”，情報処理学会論文誌，Vol. 30 No. 12, p.p.1628-p.p.1638, (1989).

4) 星出，薗田，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”，電子情報通信学会技術研究報告，CPSY91-7, p.p.25-p.p.32, (1991).

5) Amamiya, M. and Taniguchi, R., “Datarol: A Massively Parallel Architecture for Functional Language”, Proc. SPDP, p.p.726-p.p.735, (1990).

6) Ruggiero, C. A. and Sargeant, J., “Control of Parallelism in the Manchester Dataflow Machine”, Proc. Conf. Functional Programming Languages Computer. Architecture, Portland, OR, LNCS 274, (1987).