2005－ARC－161　(6)
2005／1／21

# SR11000 向け実行資源均等化命令スケジューリング

橋本 博幸　　本川 敬子　　久島 伊知郎

(株) 日立製作所システム開発研究所

スーパーテクニカルサーバ SR11000 モデル J1 に搭載されている POWER5 プロセッサでは、数命令ずつグループ化してグループ単位で命令の発行・完了などを行っている。POWER5 ではこのグループ内の位置(スロット)により利用する実行資源が固定されている。このため、ロード/ストアや浮動小数点演算などの実行資源は 2 個ずつ実装されているが、ある実行資源を利用する命令がグループ内の特定のスロットにのみ存在する場合、2 つのうち一方の実行資源だけを使うため、もう一方の実行資源が利用されず、最大性能が引き出せないという問題が生じる。そこで、グループ内の命令配置位置を考慮する実行資源均等化命令スケジューリングを開発した。実数型の基本演算プログラムに適用した結果、総和計算で約 21%、加算計算で約 16%の性能向上が得られた。

# Execution resource equal-ized instruction scheduling for SR11000

Hiroyuki Hashimoto,　Keiko Motokawa　and　Ichiro Kyushima
Systems Development Laboratory, Hitachi, Ltd.

In POWER5 processor mounted in super technical server SR11000 model J1, instruction issue, completion, etc. are performed per group which consists of some instructions. In POWER5, the execution resources used with the position(slot) in this group are fixed. For this reason, although two execution resources such as load/store and a floating point arithmetic, are mounted, when the instruction using a certain execution resource exists only in a specific slot in a group, another execution resources are not used, and the problem that the maximum performance cannot be pulled out arises. To solve this problem, the execution resources equal-ized instruction scheduling which considers the instruction arrangement position in a group was developed. As a result of applying to a series of basic floating-point arithmetic programs, about 21% of improvement in a performance was obtained for summation, and about 16% of improvement for addition calculation.

## 1. はじめに

近年の RISC プロセッサは 1 マシンサイクルで複数の命令実行が可能なスーパースカラ機能を搭載しており、コンパイラにおける命令スケジューリングでは並列実行可能な命令をなるべく同一サイクルにスケジュールするという手法が用いられてきた。この手法では、並列実行できる命令を見つけることが重要であって、並列実行できる命令間の静的な順序関係は特に考慮する必要がなかった。しかし、SR11000 モデル J1 に搭載している POWER5[1]プロセッサでは、1 マシンサイクルで実行する複数の命令をグループという単位で管理しており、グループ単位で命令の発行・完了を行っている [1)2)]。このとき、グループ内の位置(スロット)により利用できる実行資源が固定されているため、ある実行資源を使用する命令が常に特定のスロットに配置されていた場合、常に同じ実行資源だけを利用することになる。そのため、他の実行資源に空きができてしまい、プロセッサの持つ能力を使い切ることができないという問題が発生する。つまり、POWER5 プロセッサを搭載したシステムで実行資源を最大限に利用するためには、同時実行可能な命令を見つけることだけでなく、それらの命令をグループ内でどのように配置するかを考慮できるスケジューラが必要になる。

そこで、グループ内の命令配置を考慮できるスケジューリングとして、実行資源均等化命令スケジューリング手法の検討と開発を実施した。本稿では、今回開発した命令スケジューリング手法およびその評価結果について述べる。

[1] POWER5 は米国 IBM Corp.の商標です。

## 2. POWER5 の命令発行機構

### 2.1 命令発行機構

まず POWER5 での命令発行機構について説明する。

図 1 に POWER5 プロセッサの実行パイプラインを示す。

図中の小さな箱がパイプラインのステージを示している。命令はIF ステージによりフェッチされた後、D0~D3 ステージでグループの形成が行われる。分岐命令があるとそこでグループが区切られる。その結果、グループは 1~5 命令[2]で形成される[3]。グループ形成後 GD ステージでグループのディスパッチが行われ、MAP ステージで各種別(ロード・ストアユニット、固定小数点ユニット、浮動小数点ユニットなど)の実行資源の割り当てが行われる。実行資源割り当ての済んだ命令は ISS ステージで命令発行キューに入れられ、命令発行キューに入った命令は Out-of-order 実行される。自グループより古いすべてのグループが完了し、自グループ内の全命令の実行が終了した時点で CP ステージにより自グループの実行が完了される。

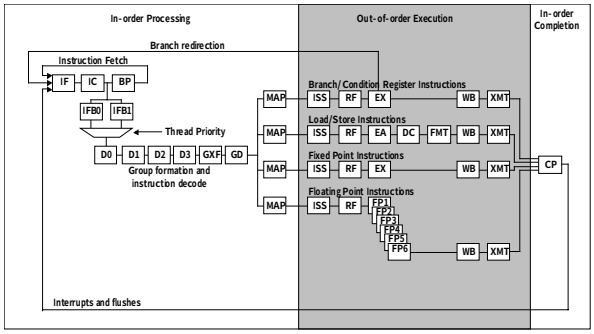


図1　POWER5 の実行パイプライン (文献2 より)

次に図2 を用いてグループのディスパッチから実行までの過程で、ディスパッチされた命令がどのように実行資源を使用し実行されるかについて説明する。

POWER5 プロセッサでは、ロード・ストアユニット(LSU)、固定小数点ユニット(FXU)、浮動小数点ユニット(FPU)に対しそれぞれ 2 個ずつの実行資源(ユニット=unit0/1)を持っており、それぞれのユニットは同時に実行可能である。また、FPU は 12 エントリの命令発行キューを 2 個、LSU/FXU は共有の 18 エントリの命令発行キューを 2 個持っている。それぞれの命令発行キューは利用するユニットが決まっている。どちらの命令発行キューに入るかは、ディスパッチされたグループ内のどの位置に命令が配置されているかによって決められている。このグループ内の位置のことをスロットと呼ぶ。スロットは 0~4 までがあり、通常の命令はスロット 0~3 に配置され、分岐命令は常にスロット 4 に配置される。スロット 0 とスロット 3 に配置された命令は命令発行キュー 0 と unit0 を、スロット 1 とスロット 2 に配置された命令は命令発行キュー1 と unit1 を利用する。このように、利用する実行資源(ユニット等)はディスパッチされるグループ内のスロット番号によって決まってしまうため、複数ある実行資源を有効利用するためには、命令発行キューに均等に命令を供給することが重要になる。この命令発行キューの使用に偏りがあると、実行資源の利用頻度に偏りが生じ、実行性能が低下するという現象が発生することがある。そのため、この性能低下を回避するために、各命令種別で命令発行キューに命令を均等に供給するための命令スケジューリングが必要になる。

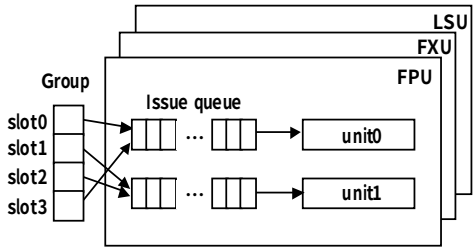


図2　実行資源の利用状態 (文献1,2 より)

## 2.2 予備調査

グループ内の命令配置状況により性能がどのように変化するかの予備調査を行った。測定プログラムの最内側ループの一部を図3 に示す。実際に測定を行ったループは、図3 に示した5 グループの命令列の後方に同様の命令例[4]で5 グループ、合計 10 グループからなるものである。測定はこのループを持つ関数を複数回呼び出すことによって行った。なお、実行資源の利用状況と実行性能の関係を調べるため、ループ内の命令列はレジスタ間の依存が極力少なくなるようにした。測定プログラムの説明を行う。コード(a)は、スロット 0,3 に FPU を利用する命令を、スロット 1,2 に FXU を利用する命令を並べたものである。この命令列では、FPU は unit0 だけを、FXU は unit1 だけを利用することになる。コード(b)は、スロット 0,1 に FPU を利用する命令を、スロット 2,3 に FXU を利用する命令を並べたものである。この命令列では、FPU と FXU が unit0 と unit1 を均等に利用することになる。ただし、FPU はスロット 0,1 だけを、FXU はスロット 2,3 だけを利用している。コード(c)は、unit0/1 の均等利用だけでなく、FPU,FXU が使用するスロットも均等になるように命令を並べたものである。

表1 にそれぞれのコードの SR11000 モデル J1 上での実行性能比較結果を示す。表1 より、コード(a)の実行性能を 1 とした場合にコード(b)で 1.52 倍、コード(c)で 1.53 倍速く実行できることが分かる。この結果より、実行性能を向上させるためには、実行資源を均等に利用するコードを生成することが重要である。

[2] 通常は 4 命令で 1 グループを形成する。5 命令目に分岐命令がある場合は、5 命令で 1 グループとなる。
[3] このとき、複雑な命令を 2 つ以上の単純な命令に分割するクラッキング処理も実施される。
[4] コード(c)の後半 5 グループの命令配置は、前半 5 グループの fadd/add の位置が反転したものとなる。

| Group | slot | コード(a) | FPU 0 | FPU 1 | FXU 0 | FXU 1 | コード(b) | FPU 0 | FPU 1 | FXU 0 | FXU 1 | コード(c) | FPU 0 | FPU 1 | FXU 0 | FXU 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | fadd fr0,fr1,fr2 | ■ | | | | fadd fr0,fr1,fr2 | ■ | | | | fadd fr0,fr1,fr2 | ■ | | | |
| | 1 | add r3,r4,r5 | | | | ■ | fadd fr3,fr4,fr5 | | ■ | | | fadd fr3,fr4,fr5 | | ■ | | |
| | 2 | add r6,r7,r8 | | | | ■ | add r3,r4,r5 | | | | ■ | add r3,r4,r5 | | | | ■ |
| | 3 | fadd fr3,fr4,fr5 | ■ | | | | add r6,r7,r8 | | | ■ | | add r6,r7,r8 | | | ■ | |
| 2 | 0 | fadd fr6,fr7,fr8 | ■ | | | | fadd fr6,fr7,fr8 | ■ | | | | add r9,r10,r11 | | | ■ | |
| | 1 | add r9,r10,r11 | | | | ■ | fadd fr9,fr10,fr11 | | ■ | | | add r12,r13,r14 | | | | ■ |
| | 2 | add r12,r13,r14 | | | | ■ | add r9,r10,r11 | | | | ■ | fadd fr6,fr7,fr8 | | ■ | | |
| | 3 | fadd fr9,fr10,fr11 | ■ | | | | add r12,r13,r14 | | | ■ | | fadd fr9,fr10,fr11 | ■ | | | |
| 3 | 0 | fadd fr12,fr13,fr14 | ■ | | | | fadd fr12,fr13,fr14 | ■ | | | | fadd fr12,fr13,fr14 | ■ | | | |
| | 1 | add r15,r16,r17 | | | | ■ | fadd fr15,fr16,fr17 | | ■ | | | fadd fr15,fr16,fr17 | | ■ | | |
| | 2 | add r18,r19,r20 | | | | ■ | add r15,r16,r17 | | | | ■ | add r15,r16,r17 | | | | ■ |
| | 3 | fadd fr15,fr16,fr17 | ■ | | | | add r18,r19,r20 | | | ■ | | add r18,r19,r20 | | | ■ | |
| 4 | 0 | fadd fr18,fr19,fr20 | ■ | | | | fadd fr18,fr19,fr20 | ■ | | | | add r21,r22,r23 | | | ■ | |
| | 1 | add r21,r22,r23 | | | | ■ | fadd fr21,fr22,fr23 | | ■ | | | add r24,r25,r26 | | | | ■ |
| | 2 | add r24,r25,r26 | | | | ■ | add r21,r22,r23 | | | | ■ | fadd fr18,fr19,fr20 | | ■ | | |
| | 3 | fadd fr21,fr22,fr23 | ■ | | | | add r24,r25,r26 | | | ■ | | fadd fr21,fr22,fr23 | ■ | | | |
| 5 | 0 | fadd fr24,fr25,fr26 | ■ | | | | fadd fr24,fr25,fr26 | ■ | | | | fadd fr24,fr25,fr26 | ■ | | | |
| | 1 | add r27,r28,r29 | | | | ■ | fadd fr27,fr28,fr29 | | ■ | | | fadd fr27,fr28,fr29 | | ■ | | |
| | 2 | add r30,r31,r0 | | | | ■ | add r27,r28,r29 | | | | ■ | add r27,r28,r29 | | | | ■ |
| | 3 | fadd fr27,fr28,fr29 | ■ | | | | add r30,r31,r0 | | | ■ | | add r30,r31,r0 | | | ■ | |

図3　測定プログラム(最内側ループの一部)

表1 実行性能

| コード | 相対性能 |
|---|---|
| (a) | 1.00 |
| (b) | 1.52 |
| (c) | 1.53 |

## 3　実行資源均等化命令スケジューリング

### 3.1 処理手順

本節では実行資源均等化命令スケジューリングの処理手順について説明する。スケジューリングでは命令の配置順が問題になるため、基本ブロックの先頭に配置できる命令から順に探す、いわゆる cycle-based のリストスケジューリング[3]を行うものとする。つまり、既に配置した命令間に別の命令を配置することはしないものとする。また、命令間の定義-使用に関するレイテンシも必ず考慮する。つまり、資源のバランスを取るために、レイテンシの解消されてない命令を持ってきて配置することはしない。

以下図4を用いてアルゴリズムの詳細説明を行う。

まず始めに、[1]でスケジューリング前の各実行資源利用数をカウントしておく。次に[2]では、スケジューリング対象としている基本ブロック内の命令の依存関係を調べDAG(Directed Acyclic Graphs)と、スケジューリング結果を格納するスケジュール表を作成する。[3]では、現在のスロット番号を保持する変数(slot_no)の初期化を行う。グループの先頭は必ずスロット番号0なので、初期値は0になる。[4]-[23]で DAG 内の全命令に対し、スケジューリングを行う。[5]では、DAG 中の未スケジューリング命令のうち、レイテンシが解消され現在のスロットに配置可能な命令の集合を求める。これを、配置可能命令集合と呼ぶ。この集合は、基本ブロック内のすべての命令から候補を探す方法と、スケジュール前にグループ分けを行い[5]、グループ間の移動が発生しない命令から候補を探す方法の2つを今回試した。

[6]-[23]で配置可能命令集合が空になるまで[7]以下の処理を行う。[7]では、現在のスロットに命令を配置した場合に、各命令種別が利用する実行資源のバランスがどうなるかを

```
 1: 実行資源利用数のカウント
 2: DAG，スケジュール表の作成
 3: slot_no=0
 4: while (DAG に印の付いてない命令がある) {
 5:   配置可能命令集合を求める
 6:   while (配置可能命令集合が空になるまで) {
 7:     命令種別ごとに優先順位(高/低)を決定する
 8:     優先順位ごとに命令種別のソート
 9:     for (優先順位高命令種別の数) {
10:       if (配置可能命令集合内に同一種別の命令が存在する) {
11:         スケジュール表に登録
12:         DAG に印を付ける
13:         goto next_slot
14:       } }
15:     for (優先順位低命令種別の数) {
16:       if (配置可能命令集合内に同一種別の命令が存在する) {
17:         スケジュール表に登録
18:         DAG に印を付ける
19:         goto next_slot
20:       } }
21: next_slot:
22:     スロット番号(slot_no)の更新
23:   } }
24: スケジュール表から実行資源利用数をカウント
25: if (実行資源バランスが改善された) {
26:   命令の並び換え
27: }
```

図4　スケジューリングアルゴリズム

調べ、命令種別毎に優先順位を決定する。ここで命令種別とは、LSU/FXU/FPU のいずれかである。ここでは、ある実行資源と他方の実行資源との利用数の差を計算し、差が2以上になる場合は、その命令配置により資源利用数バランスが崩れると判断し、その命令種別の優先順位を「低」にセットする。バランスが崩れないと判断した場合は、優先順位を「高」にセットする。スロットに配置可能な命令が複数存在する場合、この命令種別の優先順位に従って配置する命令を選択することになる。バランスを調べる実行資源としては、ユニットとスロットの2つが考えられる。

[5] 事前にスロットを意識しない(従来の)命令スケジューリングを行っておく。

[8]では、優先順位が「高」または「低」として登録された命令種別が複数ある場合、さらにその中での優先順位を決定する。優先順位は、命令種別ごとに現在のユニット利用最少数を求め、利用数の少ない命令種別の順位が高くなるようにセットする。これは、これ以前に配置された命令列で使用した命令種別のうち、一番少ないものを優先的に割り付けることにより、命令種別ごとのバランスを整えるための処理である。異なる種別で利用数が同じ場合は、あらかじめ決められた優先順に従って並び変える。

[9]-[14]では、[8]でソートされた命令種別の順に、配置可能命令集合内にその種別の命令が存在するかを調べる。命令が存在する場合は、その命令を配置可能としてスケジュール表に登録し、DAG にスケジュール済みという印を付ける。配置可能命令が複数ある場合は、スケジューリング前の命令並びで先に出現するものを選択する。もし配置可能命令がない場合は、[15]-[20]で、優先順位低命令種別の中から配置する命令を探す。

[22]ではグループ内の 1 命令が配置されたため、次の命令配置のためにスロット番号を更新する。なお、スロット番号は最大 3 までなので、3 の次の更新値は 0 となる。

すべての命令の配置が終了した後[24]でスケジュール表に配置された命令列から実行資源の利用数をカウントする。そして[25]で、スケジューリングにより実行資源利用数のバランスが改善されたかどうかを調べる。これは、実行資源ごとに利用数の最多のものから最少のものの差を計算し、その差が小さくなったかどうかで判断する。この差が小さくなったか、変わらない場合、バランスが改善されたものとみなし、[26]でスケジュール表に従って命令の並べ変えを行う。なお、バランスが改善されなかった場合は、スケジューリング結果を破棄し、スケジューリング前の命令列を結果として返す。このバランスを調べる実行資源としては、ユニットとスロットが考えられる。

次に、実際のスケジュール手順を図 3 コード(a)の命令列を使って説明する。この例では、図 4[5]の配置可能命令集合を基本ブロック内から探し、図 4[7]の優先順位決定時の実行資源チェックはユニットとスロットに対して行うものとする。

初期状態(図 5(1))の後、基本ブロックの先頭(slot0)に配置できる命令を基本ブロック内から探す。ここでは、図中の 20 命令すべてが配置可能命令集合として登録される。次に[7]で、各命令種別のバランスを調べる。例えば、slot0 に FPU 種別の命令を配置した場合、FPU の unit0 と slot0 の利用数が 1 となり、unit0 と unit1 または slot0 と slot1/2/3 との差が 1 なので、バランスは崩れないと判定され、FPU は優先順位高として登録される。同様に FXU もバランスが崩れないと判定され、優先順位高として登録される。[8]で優先順位高命令種別のソートを行うが、ユニット利用数の最少数が共に 0 であるので、あらかじめ決めてある優先順にソートする。ここでは FPU の優先順が高いとし、FPU,FXU の順で並ぶものとする。[9]-[14]では、優先順の高い FPU 命令(fadd fr0, fr1, fr2)が選択され、スケジュール表に配置される。スケジュール後[22]でスロット番号の更新(slot_no=1)を行い、次に配置する命令を考える。

slot1 に FPU 種別の命令を配置した場合、FPU の unit1 と slot1 の利用数が 1 となる。他の実行資源との差を調べると、unit0 と unit1 の差は 0 で、slot1 と slot0/2/3 との差は 1 となる。そのため、利用数のバランスは崩れないと判定され、FPU が優先順位高として登録される。同様に FXU も優先順位高として登録される。命令の選択では、それぞれの種別の利用最少数が共に 0(まだ slot1 へは配置してないため)なので、優先順の高い FPU 命令(fadd fr3, fr4, fr5)が選択されスケジュール表へ配置される(図 5(2))。

次は slot2 に配置可能な命令を探す。ここで、FPU 種別の命令を配置した場合、FPU の unit1 の利用数が 2、slot2 の利用数が 1 となる。他の実行資源との差を調べると、unit0 と unit1 の差は 1 で、slot2 と slot0/1/3 との差は 1 となる。そのため、利用数のバランスは崩れないと判定され、FPU が優先順位高として登録される。同様に FXU も優先順位高として登録される。[8]の優先順位高命令種別のソートでは、ユニット利用数の最少数を調べるが、FPU が 1 で、FXU が 0 となるので、FXU,FPU の順で並ぶ。[9]-[14]では、優先順の高い FXU 命令(add r3, r4, r5)が選択され、スケジュール表に配置される。上記の手順を繰り返し、4 命令の配置が終了した状態が図 5(4)となる。

次のグループでは、再び slot0 に命令を配置するが、FPU の命令を配置した場合、slot0 の利用数が 2 となり、slot1/2/3 との差が最大 2(slot2/3 の利用数が 0 のため)となるため、利用数のバランスが崩れると判定され FPU は優先順位低として登録され、FXU だけが優先順位高として登録される。そのため、slot0 には FXU 命令(add r6, r7, r8)がスケジュールされる。

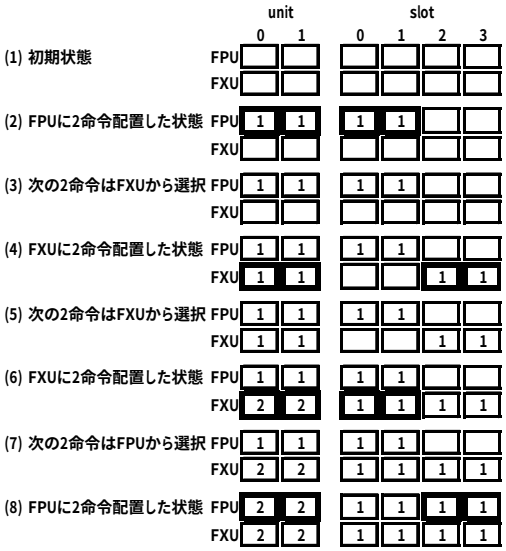


図 5　実行資源利用数遷移図

この処理の結果、図3コード(c)の命令列を作成することができる。ここで、8命令の配置を終えた時点(図5(8))でユニット・スロットの各資源の利用数の差が0となり、初期状態と同じになるため、これ以降の繰り返しでも同様の命令並びが得られることとなる。

### 3.2 コンパイラの構成

以上の実験結果を元に実行資源均等化命令スケジューリングをSR11000用のコンパイラ[4][5]に実装した。コンパイラ最適化部の構成を図6に示す。実行資源均等化命令スケジューリングでは、基本ブロック内の命令を1~5命令の単位でグループ化していくため、本スケジューリング適用後に命令の増減があると、スケジューリング時に想定したグループと実際に実行されるグループとの間に違いが発生する。そのため、命令スケジューリングを適用する位置は、命令レベルの最適化がすべて終了したところで行うものとする。具体的には、図6に示すようにレジスタ割り付け、大域命令スケジューリングやピープホール最適化後に実行する。

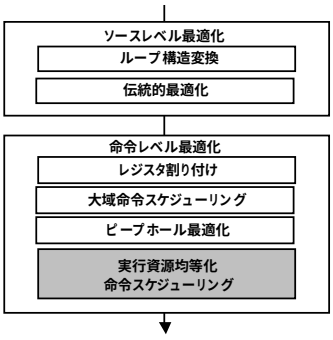


図6　コンパイラ最適化部の構成

次に、本最適化の適用範囲について説明する。今回対象としたPOWER5プロセッサでは、分岐命令で必ずグループが切れるため、分岐命令の飛び先である基本ブロック(途中で分岐や飛び込みのない命令列)の先頭をグループ化の基準点にする。また、適用対象とする基本ブロックは最内側ループに属するものだけとした。ループ本体は複数の基本ブロックから構成されることもあり、その場合は各基本ブロックごとに本最適化を適用する。

グループ化の基準点を最内側ループの基本ブロックの先頭とすることの妥当性について図7を用いて説明する。
通常のループ構造を示したものを図 7(a)に示す。ループは、ループプリヘッダ(1)、ループ本体(2)、ループ出口(3)と呼ばれる基本ブロックから構成され、命令はこの順に配置される。ループプリヘッダブロックの最後には分岐命令が存在しないため、ループプリヘッダブロックの命令数によっては、ループ本体の命令と結合されたグループが形成される場合がある(図7(b))。しかし、2回目以降の繰り返しでは図7(c)のように、ループ本体の最後の命令が分岐命令であるため、必ずループ本体の先頭からグループが形成される。つまり、ループの最初の1回だけは、グループ形成が想定したものと異なる可能性があるが、2 回目以降は最内側ループの基本ブロックの先頭がグループの切れ目となり、ここをグループ化の基準点とすることが妥当である。

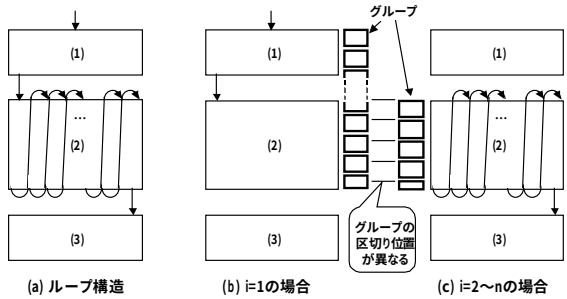


図7　ループ構造とグループ区切りについて

## 4. 評価

本報告で述べた実行資源均等化命令スケジューリングの効果を実数型基本演算を対象に評価した。図8に測定に用いたプログラムの一部を示す。

```
(a) 加算
  do i=1,n
    c(i) = a(i) + b(i)
  enddo

(b) 内積
  do i=1,n
    sum = sum + a(i) * b(i)
  enddo

(c) リカレンス
  do i=1,n-1
    c(i+1) = const * c(i) + const2
  enddo

(d) 総和
  do i=1,n
    sum = sum + a(i)
  enddo
```

図8 測定プログラムの一部

測定プログラムは図8の4種類(加算、内積、リカレンス、総和)で、それぞれn回繰り返すループ構造になっており、さらにこのループをm回呼び出して性能測定を行った。なお、測定はSR11000モデルJ1上で実施した。

図9が図8の4種類のプログラムに本最適化を適用し、スケジューリング対象ループを 16 並列で実行した場合の測定結果である。グラフでは、本最適化適用前の実行性能を1とした場合の、適用後の実行性能の相対比を示している。測定は図4[5]で配置可能命令を基本ブロック内の全命令から探すパターン(A)と、グループ間の命令移動が生じない範囲から探すパターン(G)の 2 種類と、図 4[7]の優先順位決定時の実行資源としてユニットとスロットの両方を調べるパターン(us)と、ユニットのみを調べるパターン(u)の2種類の組み合わせで合計4種類について実施した。

さらに、表2に本最適化適用前後での最内側ループにおけるそれぞれの実行資源利用数を示す。なお、今回は実行資源均等化命令スケジューリングの評価を行うのが目的であるため、本来はユニット・スロットの利用バランスが改善された場合だけスケジューリング結果を採用するのだが、改善さ

れなかった場合もスケジューリング結果を採用するようようにした。

図 9 の結果より、配置可能命令を同一グループ内で探し、かつ、優先順位決定時の実行資源としてユニットとスロットの両方を調べるパターン(Gus)において、総和計算で約 21%、加算計算で約 16%の性能向上が見られる。総和については表 2 より、LSU/FPU 共にユニットの利用バランスは変わらないが、スロットの利用バランスが改善されていることが分かる。具体的にはスケジューリング適用前は、LSU でスロット 0,1 だけしか利用していなかったが、スケジューリング適用後はスロット 0,1,2,3 すべてを均等に利用するコードになっている。また、FPU も同様のことが言える。2.2 の予備調査ではスロット均等化の効果は見られなかったが、総和については均等化の効果が見られた。加算についても表 2 より、LSU/FPU のユニット利用バランスが改善されていることが分かる。逆に内積では約 5%の性能劣化が見られる。これは、スケジューリング適用前は LSU のユニット利用バランスが均等であったが、スケジューリング適用後にユニット利用バランスが崩れたためと考えられる。但し、先程も述べたとおり、我々のコンパイラでは、ユニットやスロットの利用バランスが崩れた場合は、スケジューリング結果を破棄して元のコードをそのまま出力するため、実際のところ性能劣化は発生しない。

これらの結果から、SR11000 モデル J1 上で、実行資源を均等化する命令スケジューリングが有効であることが示された。

## 5. 関連研究

文献 6)では、命令間に存在するレイテンシを無視し、1 サイクルに利用可能な実行資源に注目したスケジューリング方式を提案している。この方式では、実行資源のうち各命令種別で 1 サイクルに完了可能な命令数に注目し、各サイクルに配置する命令を決定している。そして、このスケジューリング方式が Out-of-order 機構を持つプロセッサ(PA-8000)に対し、有効であることを示している。

## 6. まとめ

本稿では、SR11000 モデル J1 に搭載している POWER5 向けに実行資源均等化命令スケジューリング手法の説明と効果を評価した。POWER5 では、複数の命令をグループという単位で管理し、グループ単位で命令の発行・完了を行っている。さらに、グループ内の命令位置により、利用する実行資源が予め決められているため、実行資源の利用バランスによって実行性能に影響が出ることが示された。そこで、本稿で説明した命令スケジューリングを実数型の基本演算プログラムに適用することで、総和演算と加算演算で実行資源の利用バランスが改善し、実行性能がそれぞれ約 21%、約 16%向上することが示された。

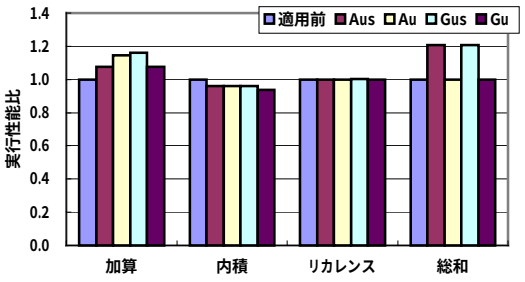


図 9　実行性能比(16 並列実行)

表 2　実行資源利用数

| LSU | ユニット(0/1) スロット(0/1/2/3) 適用前 | Aus | Au | Gus | Gu |
|---|---|---|---|---|---|
| 加算 | 11/13<br>6/7/6/5 | 13/11<br>8/6/5/5 | 12/12<br>8/8/4/4 | 12/12<br>6/6/6/6 | 11/13<br>3/6/7/8 |
| 内積 | 20/20<br>10/10/10/10 | 19/21<br>10/11/10/9 | 19/21<br>11/12/9/8 | 21/19<br>10/10/9/11 | 20/20<br>5/7/13/15 |
| リカレンス | 11/11<br>6/5/6/5 | 11/11<br>5/5/6/6 | 11/11<br>5/5/6/6 | 11/11<br>6/5/6/5 | 11/11<br>7/6/5/4 |
| 総和 | 10/10<br>10/10/0/0 | 10/10<br>6/6/4/4 | 10/10<br>10/10/0/0 | 10/10<br>5/5/5/5 | 10/10<br>10/10/0/0 |

| FPU | ユニット(0/1) スロット(0/1/2/3) 適用前 | Aus | Au | Gus | Gu |
|---|---|---|---|---|---|
| 加算 | 5/11<br>3/5/6/2 | 7/9<br>5/5/4/2 | 7/9<br>4/4/5/3 | 8/8<br>5/4/4/3 | 8/8<br>6/4/4/2 |
| 内積 | 9/11<br>5/6/5/4 | 10/10<br>5/5/5/5 | 10/10<br>4/4/6/6 | 9/11<br>5/6/5/4 | 10/10<br>10/9/1/0 |
| リカレンス | 19/21<br>11/10/11/8 | 20/20<br>11/9/11/9 | 20/20<br>11/9/11/9 | 19/21<br>11/10/11/8 | 19/21<br>11/10/11/8 |
| 総和 | 10/10<br>0/0/10/10 | 11/9<br>5/4/5/6 | 11/9<br>1/0/9/10 | 10/10<br>5/5/5/5 | 10/10<br>0/0/10/10 |

## 7. 参考文献


1) "The POWER4 Processor Introduction and Tuning Guide", IBM RedBooks(2001), http://www.redbooks.ibm.com/redbooks/pdfs/sg247041.pdf
2) " Advanced POWER Virtualization on IBM eServer p5 Servers Architecture and Performance Considerations", IBM RedBooks Draft(2004).
3) Gang Chen, Michael D.Smith : "Reorganizing Global Schedules for Register Allocation", ICS-99, pp.408-416, 1999.
4) 青木秀貴, 處雅尋, 本川敬子, 五百木伸洋, 石原 修 : "SR11000 モデル H1 のソフトウェアプリフェッチ手法"、電子情報通信学会 2004 年総合大会、D-6-11、2004。
5) 本川敬子, 伊藤信一, 橋本博幸, 久島伊知郎 : "SR11000 コンパイラにおけるデータキャッシュ向け最適化"、FIT2004、C-028、2004。
6) David A Dunn and Wei-Chung Hsu : "Instruction Scheduling for the HP PA-8000", Micro-29, pp.298-307, 1996.