Vol.2013-ARC-204 No.10
2013/3/27

ALU ローテーションによるスーパスカラプロセッサの性能向上

井上 聖等[1,a)]　三輪 忍[1]　中田 尚[1]　中村 宏[1]

概要：プロセッサの定格の電圧と動作周波数は，半導体素子が正常に動作することを保証するため，ジャンクション温度がしきい値を超えないように定められている．ジャンクション温度はホット・スポットのモジュールのアクティビティに依存するため，IPC を保ったままアクティビティを下げることができればより高い周波数を用いることができ，プロセッサ性能は向上する．ホット・スポットの 1 つである ALU は，通常の実装では ALU 間のアクティビティに大きな偏りが存在し，その結果，特定の ALU が高温になりやすい構造となっている．本稿では，多数の ALU を用意し，それらをラウンド・ロビンに利用することで，IPC を悪化させることなく個々の ALU のアクティビティを抑える手法を提案する．評価の結果，提案手法によって 10.4%性能が向上することがわかった．

キーワード：スーパスカラ・プロセッサ，動作周波数，ジャンクション温度，ALU，フォワーディング，バイパス

1. はじめに

LSI のピーク温度を意味する**ジャンクション温度**は，スーパスカラ・プロセッサのチップの動作周波数を決定する重要な要素である．LSI の消費電力は周波数と電圧の上昇にともなって増加するが，それにともなってジャンクション温度も上昇する．そして，ジャンクション温度があるしきい値[*1]を上回ると，半導体素子は壊れて正常に動作しなくなってしまう．そのため，市販されているプロセッサの定格の電圧と動作周波数は，ジャンクション温度がこのしきい値を決して上回ることのないように，ある程度のマージン[*2]とともに設定される．

プロセッサ・チップには**ホット・スポット**と**コールド・スポット**が存在することが知られている [1], [3], [8], [10]．また，LSI のダイナミック電力は**アクティビティ**に依存する．そのため，ジャンクション温度はホット・スポットに位置するモジュールのアクティビティに強く依存する．

プロセッサ性能は動作周波数と IPC の積によって与えられる．したがって，IPC を保ったままホット・スポットのモジュールのアクティビティを下げることができれば，ジャンクション温度が低下し，同等の安全性を確保しつつより高い周波数を利用することができ，プロセッサ性能は向上する．

ホット・スポットの 1 つである ALU は，後述するように，通常の実装では ALU 間のアクティビティに大きな偏りが生まれてしまう．結果として，特定の ALU のみが高温となりやすい構造になっている．このような構造はプロセッサをより高い周波数で動作させる上で都合が悪い．

そこで本稿では **ALU ローテーション**を提案する．提案手法では，多数の ALU を用意してそれらをラウンド・ロビンで使用することにより，ALU あたりのアクティビティを減らし，ジャンクション温度を低下させる．命令キューから ALU までのデータ・パスに追加するハードウェアはごくわずかであるため，発行レイテンシが悪化することはなく，したがって，提案手法による IPC の低下はない．そのため，ジャンクション温度の低下によってもたらされる周波数向上の恩恵をそのまま受けることができる．

以下本稿の構成は次のようになっている．まず次章において，従来の ALU 周辺のロジックを説明し，ALU のアクティビティが偏ることを示す．続く 3 章において，ALU ローテーションを詳しく説明する．評価は 4 章で行い，5 章で関連研究をまとめ，最後 6 章で本稿をまとめる．

2. 従来の ALU 周辺の構成とその問題点

本章では，ALU の割り当て方法を中心に，通常の ALU

[1] 東京大学
The University of Tokyo
[a)] inoue@hal.ipc.i.u-tokyo.ac.jp

[*1] Si 半導体の場合は 150 ～ 170°C と言われている

[*2] Intel のチップの場合は，90°C を超えるとトグリングなどの温度抑制機構が動作し始める [5] ことから，90°C を目安としているようである．

Vol.2013-ARC-204 No.10
2013/3/27

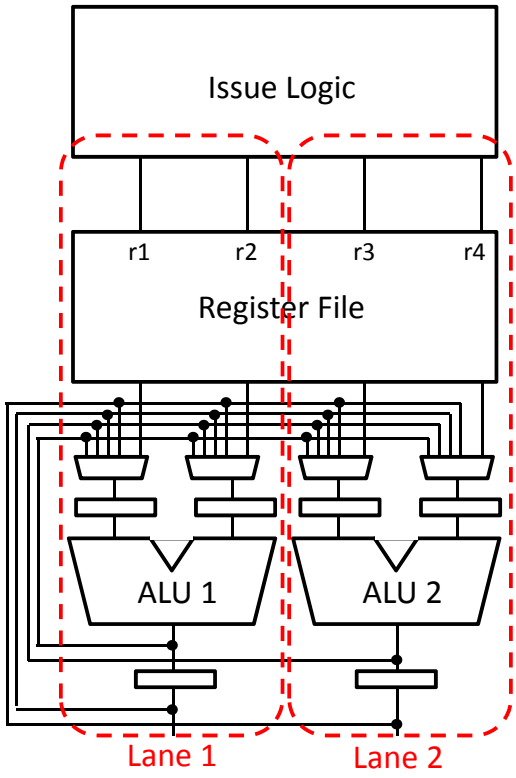

図 1 従来の ALU 周辺の構成

周辺のロジックをまず説明する．次いで，この実装では特定の ALU にアクティビティが集中することを予備評価の結果とともに説明する．

2.1 ALU 周辺の回路構成

ALU 周辺の回路構成を**図 1** に示す．図は 2 命令発行，かつ，レジスタ・アクセスに 1 サイクルを要するプロセッサを想定している．図はパイプラインを意識して描いており，命令発行ロジック，レジスタ・ファイル，ALU が上から下に向かって並んでいる．ただし，ALU からレジスタ・ファイルへライト・バックするデータ・パスは簡単のため省略してある．

図に示すように，レジスタ・ファイルのポートと ALU は配線によって直接結ばれており，1 つのレーンを形成する．そして，発行された命令はこのレーンに沿って処理される．例えば，レーン 1 に発行された命令は，$r1, r2$ の 2 つのリード・ポートを使ってレジスタ読み出しを行う．読み出された値は ALU1 へと送られ，実行される．

したがって，通常は，使用する ALU が命令発行時に決定される．命令発行ロジックが命令を発行するレーンを決定することにより，使用される ALU も自動的に決まる．

複数のレーンが利用可能な場合は，命令発行ロジックは，ある静的な優先度にしたがって利用するレーンを選ぶことが多い．例えば，あるサイクルにおいてレーン 1 と 2 のどちらも利用可能な場合は，まずレーン 1 に対して命令を発行する．このような単純な資源割り当てが行われているのは，命令発行ロジックは最もクリティカルなロジックの 1 つであり，資源割り当てのアルゴリズムとして複雑なものは採用しにくいためである．

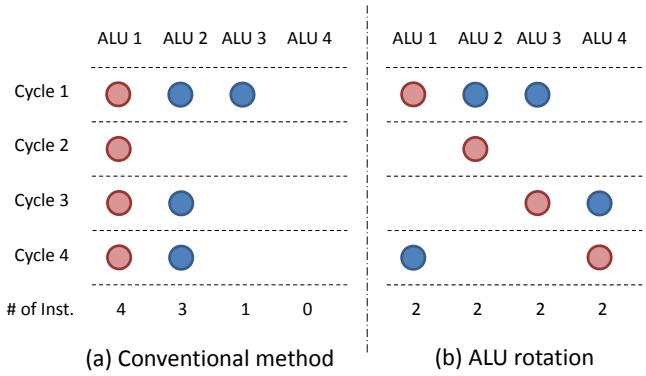

図 2 従来の資源割り当てと ALU ローテーションのコンセプト

2.2 ALU 間のアクティビティの偏り

命令発行ロジックは常に発行幅分の命令を発行するわけではないため，前節で述べた資源割り当てアルゴリズムのもとでは，ALU の利用率に大きな偏りが生じてしまう．

図 2(a) に例を示す．図は 4 つの ALU（ALU1~4）を備えた 4 命令発行のプロセッサに対し，前述の資源割り当てアルゴリズムにしたがって命令を発行する様子を表している．図の各列は ALU を表し，各行はサイクルを表す．丸印はそのサイクルにその ALU に発行される命令を表している．なお，4 つの ALU は ALU1, 2, 3, 4 の順に優先順位が付与されているものとする．

サイクル 1 において発行可能な命令が 3 命令あったとしよう．その場合，従来の資源割り当てアルゴリズムにしたがえば，最も優先度の高い ALU1 がまず選択される．図では最も優先度の高い資源に発行された命令を赤丸で表している．続いて，まだ発行可能な命令が 2 つ残っているので，ALU1 の次に優先度の高い ALU2 が選択され，さらに ALU3 も選択される．

続くサイクル 2 は発行可能な命令が 1 つのみとする．この場合，最も優先度の高い ALU1 が空いているので，ALU1 に対して命令発行が行われる．

以下同様にして，サイクル 3，および，サイクル 4 で 2 命令ずつ発行されたとする．そうすると，各 ALU に発行された命令数は図の最下段のようになる．すなわち，最も優先度の高い ALU1 には 4 命令発行される一方，最も優先度の低い ALU4 には 1 つも命令が発行されない．このように従来の資源割り当てアルゴリズムは ALU 間の利用率に大きな偏りを生んでしまう．

ALU を 4 つ有する 4 命令発行のプロセッサにおいて SPEC CINT2006 を実行した際の各 ALU のアクティビティを**図 3** に示す．シミュレーション環境とプロセッサ構成は 4 章で詳しく述べる．グラフの横軸はベンチマーク・

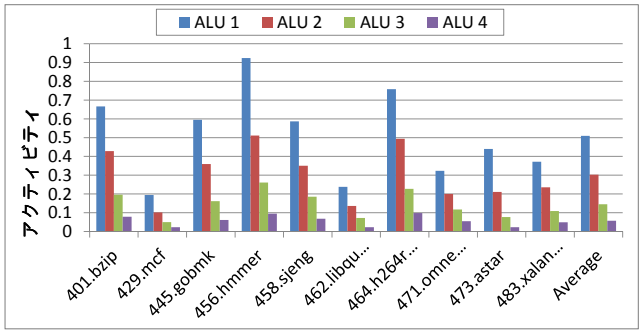

図 3 従来の資源割り当てにおける各 ALU のアクティビティ

プログラムを表し，縦軸はアクティビティ，すなわち，全実行サイクルに対してその資源が利用されたサイクルの割合を表す．プログラムごとの 4 つの棒グラフは左から順に ALU1, 2, 3, 4 に対応している．

グラフより，従来の資源割り当てアルゴリズムのもとでは，ALU のアクティビティに大きな差があることがわかる．例えば hmmer では，最も優先度の高い ALU1 のアクティビティは 9 割を超えるのに対し，最も優先度の低い ALU4 のアクティビティは 1 割以下である．平均でも ALU1 のアクティビティと ALU4 のそれとの差は 45 ポイントを超える．

このように従来の資源割り当てアルゴリズムでは特定の ALU にアクティビティが集中する．

3. ALU ローテーション

特定の ALU にアクティビティが集中することによるジャンクション温度の上昇を抑えるため，我々は ALU ローテーションを提案する．以下ではまず提案手法のコンセプトについて説明し，続く 3.2 節ではその実装を詳しく説明する．提案手法では多数の ALU を利用するが，単純に ALU 数を増加させるとフォワーディング (あるいはバイパス)・ネットワークが複雑化してしまう [11]．この問題に対して，我々は局所フォワーディング・ネットワークを提案する．最後 3.3 節ではそれについて詳しく述べる．

3.1 コンセプト

従来の資源割り当てアルゴリズムで特定の ALU にアクティビティが集中してしまうのは，複数 ALU が利用可能な時の発行の優先順位が静的に決まっているためである．図 2(a) の例では，ALU1 の優先度が常に高いため，ALU1 に対して命令が最も多く発行されてしまう．そこで，毎サイクル，優先順位をローテーションすることを考える．

提案手法のコンセプトを図 2(b) に示す．まず，サイクル 1 では，従来と同様，ALU1,2,3,4 の順に命令を発行する．このサイクルに発行可能な命令は 3 つなので，ALU1,2,3 に対して命令が発行される．

続くサイクル 2 では，ALU の優先順位を 1 つシフトし，

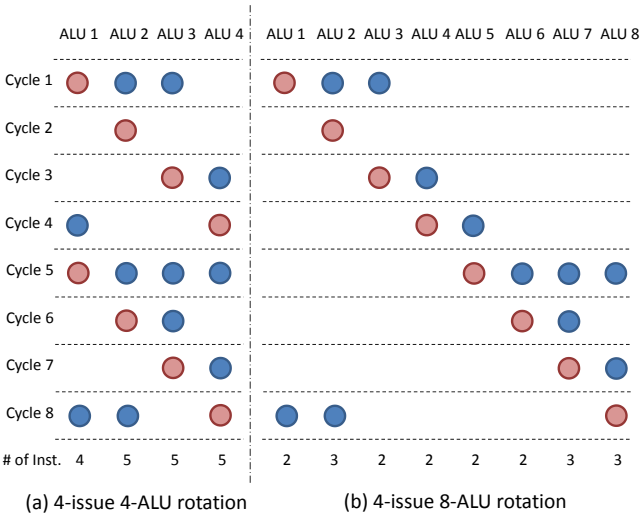

図 4 ALU 数を増やした時の ALU ローテーションの効果

ALU2,3,4,1 の順に命令を発行するものとする．このサイクルで発行可能な命令は 1 つだけなので，結局，ALU2 にのみ命令が発行されることになる．

同様に，続くサイクル 3 は ALU3,4,1,2 の順で，サイクル 4 は ALU4,1,2,3 の順で命令を発行する．このようにすると，各 ALU に発行された命令数は図 2 最下段のようになり，すべての ALU で等しくなる．従来方式と比較すると，ALU1 のアクティビティは半分に減っている．

この方式の利点は，ALU を増やせば増やすほど ALU あたりのアクティビティを減らすことができる点にある．例として，4 命令発行，8ALU のプロセッサで ALU ローテーションを行う様子を**図 4**(b) に示す．同じ発行幅の 4ALU のプロセッサで ALU ローテーションを行う場合 (同図 (a)) と比べ，ALU2 のアクティビティは 5/8 から 3/8 にまで低下している．

このように資源割り当ての優先順位を毎サイクル変えることにより，個々の ALU のアクティビティを抑えることができる．

3.2 実装

ALU ローテーションを行う場合の ALU 周辺の回路構成を**図 5** に示す．図は，図 1 のプロセッサに 2 つの ALU を追加し，計 4 つの ALU を用いてローテーションを行う場合を想定している．

提案手法では命令発行のデータ・パス上に交換器を追加する．そして交換器内に位置するデマルチプレクサの制御信号を毎サイクル適切に切り替えることでローテーションを行う．例えば，r1, r3 から読みだされた値は，最初のサイクルではそれぞれを ALU1,ALU2 に送り，次のサイクルではそれぞれを ALU2, ALU3 に送り，といったように制御する．なお，図では省略してあるが，通常は，フォワーディングされた値を選択するためのセレクタの制御信号は発行ロジックで生成され，セレクタへと送られる．その

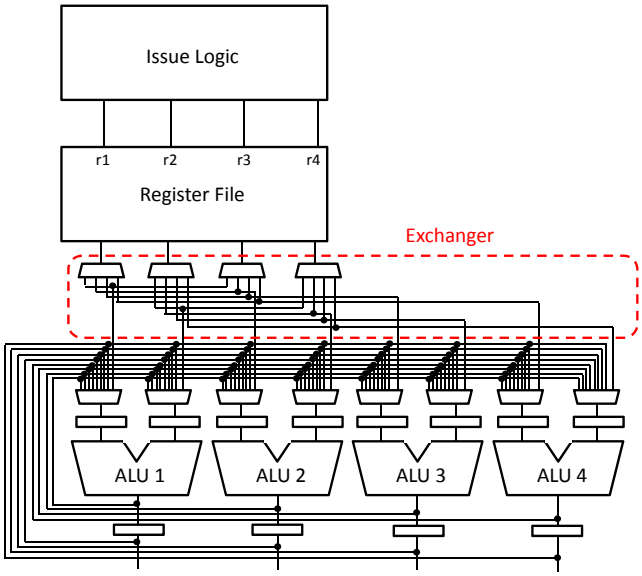

図 5 ALU ローテーションの実装

データ・パス上にも交換器は挿入される．

図に示すように，ALU ローテーションによって命令発行のデータ・パス上に追加されるハードウェアは，ALU 数を N とすると，1toN のデマルチプレクサのみである．したがって，発行レイテンシへの影響は軽微であると言える．次章では ALU ローテーションの効果を詳しく評価するが，その際はデマルチプレクサの挿入によって発行レイテンシは変わらないものとしている．

ALU 数を増やした場合に問題となるのはフォワーディング・ネットワークが複雑化してしまうことである．図 5 と図 1 を比べればわかるように，ALU 数が 2 倍になったことでフォワーディング・パスも倍増している．その結果，ソース・ラッチの手前に位置するセレクタの入力は 5 から 9 へと増えてしまっている．このセレクタの入力はフォワーディング・パス数に依存するため，ALU をさらに増やしていった場合にはさらに多入力のセレクタが必要となり，フォワーディング・パスの遅延を悪化させてしまう．フォワーディング・パスは一般にクリティカルであることから，その遅延の悪化は受け入れがたい．

3.3 局所フォワーディング・ネットワーク

前節で述べたフォワーディング・パスの複雑化の問題に対し，我々は**局所フォワーディング・ネットワーク**（Partial Forwarding Network．以後 **PFN** とする）を提案する．**図 6** に例を示す．

ここで，説明の前に優先順位のシフト幅という，優先順位をいくつシフトするかという値を導入する．前節まではサイクルの経過に従って，ALU の優先順位を 1 つずつシフトしてきたが，当然 2 以上のシフト幅となる実装をすることは可能である．例えばシフト幅 2 のものは，優先順位が ALU1,2,3,4 である次のサイクルには，優先順位は

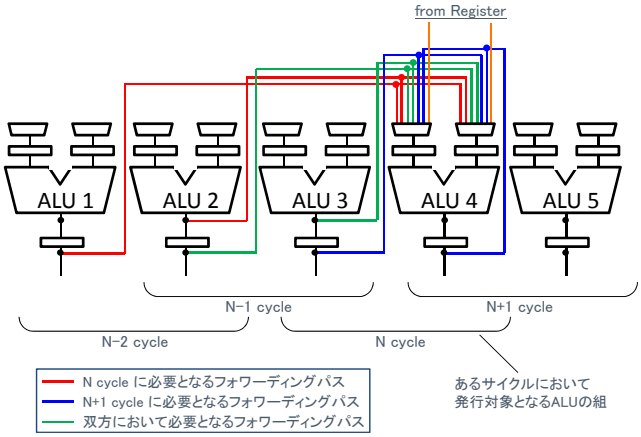

図 6 Partial Forwarding Network(シフト幅 1)

ALU3,4,1,2 という具合になる．また，(ALU 数/シフト幅) が割り切れる場合，常に特定の組，この例では ALU1,2 と ALU3,4 がセットで利用されることになる．

ALU を増加することは個々の ALU のアクティビティを低減するものの，前述のようにペナルティが大きい．そこで ALU を増加させるものの，発行幅は増加させないというやり方が考えられる．そのような実装ではすべての ALU が同時に利用される状況は発生しないため，本節で提案する PFN のように，完全なフォワーディング・ネットワークではなく，ALU の数や発行幅の構成に応じて適切に局所的なフォワーディング・ネットワークを形成すれば良い．なお，ここでは前節までに述べてきた ALU のローテーション利用を前提として話を進める．

PFN は発行幅やローテーション利用のシフト幅，およびフォワーディング回路がデータを保持すべきサイクル数等により，種々の取りうる構成が考えられる．例として ALU 数が 6 以上，命令発行幅が 2 という構成における，シフト幅が 1 の場合について図 6 で，シフト幅が 2 の場合について**図 7** を用いて説明する．ただしレジスタのリード/ライトはそれぞれ 1 サイクルを想定する．

図 6 では，図が煩雑になるのを避けるために ALU5 へのフォワーディングに必要なパスのみを示しているが，すべての ALU に対して図に示したものと同等のパスが実装されている必要がある．ここでは，ALU5 を例に説明を進めることとする．シフト幅が 1 のローテーションでは図に示した，ALU5 の優先度が 2 番目である N cycle 時と，ALU5 の優先度が 1 である N+1 cycle 時の，2 つの場合を考える必要がある．前者の場合には 1 cycle 前，すなわち N-1 cycle の実行結果である ALU2，3 の直接の出力と 2 cycle 前，すなわち N-2 cycle の実行結果である ALU1，2 のラッチを挟んだ出力がセレクタへの入力として必要となる．後者の場合には 1 cycle 前，すなわち N cycle の実行結果である ALU3，4 の直接の出力と，2 cycle 前である N-1 cycle の実行結果である ALU2，3 のラッチを挟んだ出力がセレ

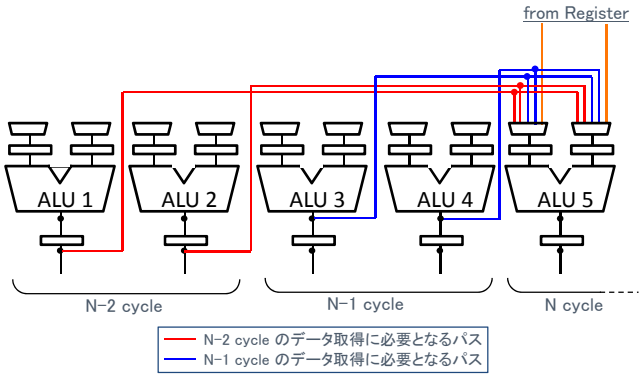

図 7 Partial Forwarding Network(シフト幅 2)

クタへの入力として必要となる．図では前者と後者の場合に共通して利用可されるパスを緑色の線で示した．また，これに加えてレジスタからの入力も必要であり，これらのことからシフト幅 1 のローテーションではオペランド 1 つあたりに 7to1 のセレクタが必要となっている．これに対して，完全フォワーディングネットワークを実装するためには，ALU 数 8 を想定するならば ALU1 個につき 2 入力，さらにレジスタからの入力を加味して 17to1 のセレクタが必要となる．

次にローテーション幅 2 の場合について図 7 に示しているが，こちらは 2 サイクル前の実行結果のフォワーディングに必要なパスを赤線で，1 サイクル前のモノについては青線で記した．こちらもシフト幅 1 の場合と同様に，ALU5 に着目した図である．発行幅 2 に対してシフト幅も 2 であるときには，ALU1,2，ALU3,4，ALU5,6 という組み合わせでしか ALU が同時に利用されることはないので，現在 N cycle に ALU5 が利用されていると思えば，N-2 cycle には ALU1,2，そして N-1 cycle には ALU3,4 のみが利用されている．また，N+1 cycle には ALU5 は利用されないため，シフト幅 1 の場合のように ALU5 の出力を自身への入力とする必要はない．レジスタからの入力も考慮しこれらをまとめると，オペランドあたりに求められるセレクタの要件は 5 入力 1 出力であり，シフト幅 1 の場合と比べてさらに 2 入力分不要となっている．

このように，ALU 数に対し発行幅が小さい場合には，フォワーディング・ネットワークで結果を保持すべきサイクル数や ALU 数，発行幅に依るものの，完全なネットワークが無くともフォワーディング機能の実現が可能あり，適切な構成をとることで回路の複雑化を緩和することができる．

4. 評価

シミュレータを用いて ALU ローテーションによるアクティビティ分散の効果と，それによる温度低下および達成可能な周波数について評価を行った．また，ALU ローテーションにより発行レイテンシや，配線遅延が増加した場合を想定した IPC についても評価し，それらを考慮した総スループットの向上率を求めた．本章ではシミュレーション環境について説明した後，その結果を示す．

4.1 評価環境

ALU ローテーションの効果確認のため，サイクルアキュレートなシミュレータである鬼斬弐を用い，提案手法を実装した．基本的なプロセッサ構成は**表 1** に示すとおりである．

また，各手法による温度低下への影響と，それによる周波数のゲインについてもシミュレーションにより評価を行う．こちらの評価では温度シミュレータとして, 主要なカンファレンスにおいても幾つもの論文で用いられている [7], Hotspot を [3] 用いる．Hotspot への入力はプロセッサのフロアプラン，及びフロアプランに対応した各モジュールの電力トレースである．

想定するフロアプランを**図 8** に示す．フロアプランは McPAT[6] を用いて各モジュールの面積を見積もり，それを元に Intel Nehalem アーキテクチャのフロアプラン [2] を参考に設計した．ただし，図中最上部に配置した alu_4,5,6,7, および_space は ALU 数が 8 の時のみ存在するものとし，_space はダイナミック，リークのいずれの電力も消費しない．また，ALU の面積は 4 から 8 にする際に単純に 2 倍の面積とし，面積と同様にリーク電力も増加している．

各モジュールの電力トレースは，McPAT および前述の鬼斬弐を用いて取得した．McPAT に想定するプロセッサ構成を入力として与え，モジュールの最大消費電力を評価する．その電力の値と，鬼斬弐を用いて取得した各モジュールのアクティビティのトレースの各区間の値を掛けることで，各モジュールの電力トレースデータを取得する．区間の長さは 1M サイクルとした．

温度シミュレーションに関する種々のパラメータについては**表 2** に示す．プロセスは 45nm を想定する．チップの厚みは，HotSpot の論文に記載されていた値（130nm の時で 0.5mm）をもとに，単純にスケーリングして求めた．周辺温度は 40°C とした．その他いくつかの値については [4] を参照した．また，想定する上限温度は商用プロセッサが温度制御回路を作動させる閾値温度を参考に 90 ℃とした [5].

ベンチマークには SPEC2006 の INT，FP を用い，ALU ローテーションによるアクティビティの変化の評価では 1G 命令をスキップした後，1G 命令を実行し評価した．それを受けての温度評価では 1G 命令をスキップした後，11G サイクル実行した．Mesa-Martinez 等によると積極的な冷却システムでは 31 ミリ秒，より消極的な冷却システムでは 2000 ミリ秒のシミュレーション時間は確保すべきであるとしており [7]，今回の 11G サイクルのシミュレーションは，動作周波数 4GHz を想定した時には 2.75 秒に相当

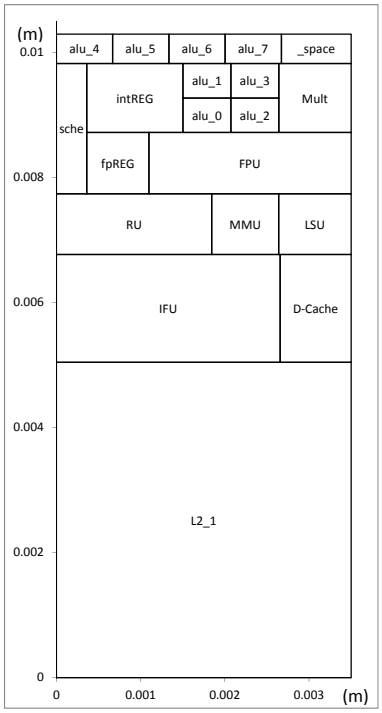

図 8 想定するフロアプラン

し，過度に高い周波数でなければ十分な時間の温度シミュレーションとなっている．なお，周波数は 100MH z 刻みで評価を行った．

4.2 評価結果

4.2.1 アクティビティの偏りの緩和

ALU ローテーションによる ALU 間のアクティビティの偏りの変化を**図 9**，および**図 10** に示す．図 9 は，図 3 と同様に，縦軸が各 ALU のアクティビティ，横軸が SPEC CINT2006 の各ベンチマークを表している．図より，ベンチマークによらず各 ALU のアクティビティは等しくなっている．また，アクティビティの絶対値に着目すると図 3 と比較して各ベンチマークのアクティビティが平均化されていることが確認できる．

図 10 のシフト幅 2 の場合では機械的に ALU の優先順位を 2 ずつ増えるようなシミュレーションを行った．そのため同時に利用されうる ALU の組みにおいての優先順位は静的なものとなっており，シフト幅 2 では ALU の番号が奇数のものと偶数のものとでアクティビティにやや差が生まれる結果となっている．しかし，これはローテーションが 1 順するごとに，同時に利用される ALU の組の中で更にローテーションを行う，すなわち階層的なローテーションを行うことでこの差をほぼ解消できるものと推測する．

表 1 ベース・プロセッサの構成

Parameters	Remarks
Frequency	3.5GHz
Way	4
Issue Width	INT:4 / FP:1 / Mem:2
Issue Latency	4
Instruction Queue	INT:32, FP:16
Load/Store Queue	32 (in each)
Register File	INT:128, FP:128
L1I-Cache	64KB, 16B/line, 2 way, 2 cycles
L1D-Cache	64KB, 16B/line, 2 way, 2 cycles
LLC (per core)	2MB, 16B/line, 8 way, 32 cycles

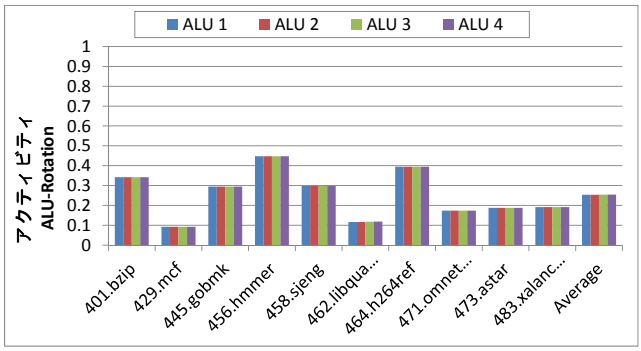

図 9 ローテーション利用時のアクティビティ
(4alu - 4issue - シフト幅 1)

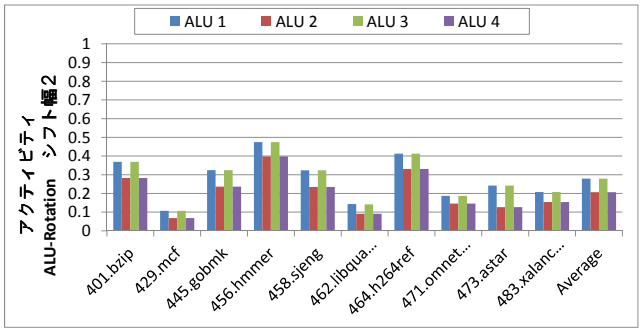

図 10 ローテーション利用時のアクティビティ
(4alu - 2issue - シフト幅 2)

4.2.2 温度低下とそれによる周波数ゲイン

従来方式と比較して ALU ローテーションによる温度低

表 2 温度シミュレーションに用いたその他のパラメタ

Parameters	Remarks
Chip Conditions	
Process technology	45nm
Chip Thickness	0.2mm
Silicon Thermal Conductivity	100W/(m·K)
Heat Sink Conditions	
Side	60mm
Thickness	6.9mm
Thermal Conductivity	400W/(m·K)
Heat Spreader Conditions	
Side	30mm
Thickness	1.87mm
Thermal Conductivity	400W/(m·K)
MISC. Conditions	
Ambient Temperature	40 ℃
peak Temperature limit	90 ℃

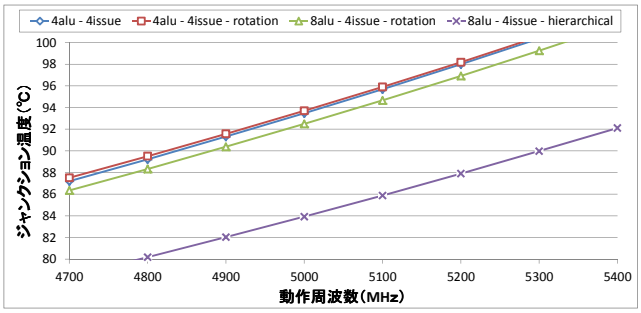

図 11 提案方式の温度および周波数評価

下と，それによって利用可能となる周波数についての評価を**図 11** に示す．図の横軸はプロセッサの動作周波数を，縦軸はそれに対応したジャンクション温度を表している．凡例名末尾の rotation は ALU ローテーションを適用した手法，hierarchical は今回階層的なローテーションをシミュレーションしていないため，仮に行ったとしてアクティビティを ALU 間で均等に分散できた場合を想定した評価であることを示している．また，ALU 数より発行幅が小さいものは，シフト幅を発行幅と等しい値とした．

まず温度に関して図 11 より，従来の 4alu-4issue とそのローテーションを比較すると，フロアプランの影響もあり，ベンチマークや周波数により温度が低下したり上昇したりしており，トータルではジャンクション温度が極わずかに上昇してしまう結果となった．8alu-4issue-rotation ではフロアプランや階層的にローテーションを行なっていないことが影響し，ALU を増やした効果があまり見られていない．一方で 8alu-4issue-hierarchical について見ると，アクティビティが適切に分散されたことで，4alu-4issue と比較して 10 ℃前後の温度低下を実現している．

周波数に関して図中 90 ℃のラインに着目すると，従来構成である 4alu-4issue では 4800MHz，最も温度低下を実現できた 8alu-4issue-hierarchical では 5300MHz が 90 ℃を上回らない最も高い周波数であり，このことより hierarchical な構成による周波数向上率は 10.4%に及ぶ．ただし，8alu の構成では ALU の増加に伴い，全体で約 4.8%面積が増加している．

4.2.3 IPC の低下を考慮したスループット

ALU ローテーションにより発行レイテンシや，配線遅延が増加した場合を想定し，すべての命令発行にかかるレイテンシが 1 サイクル伸びた場合の IPC 低下を**図 12** に示す．縦軸は，発行レイテンシ増加前の値で正規化した相対 IPC を，横軸はベンチマーク・プログラムを表している．図より，IPC は最大で約 3.4%，平均で約 2.1%の低下となった．

最後に，これまでの評価を踏まえて，各構成の総スループット (IPC ×動作周波数) を**図 13** に示す．縦軸は従来方式の 4alu-4issue の値を 1 とした相対スループットを，横軸は各構成および発行レイテンシ増加の有無を示してお

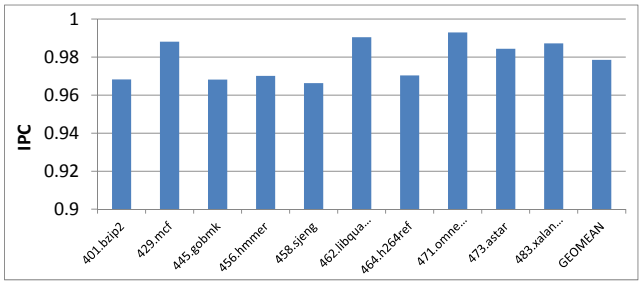

図 12 発行レイテンシ 5 の場合の IPC 低下
（発行レイテンシ 4 の値で正規化）

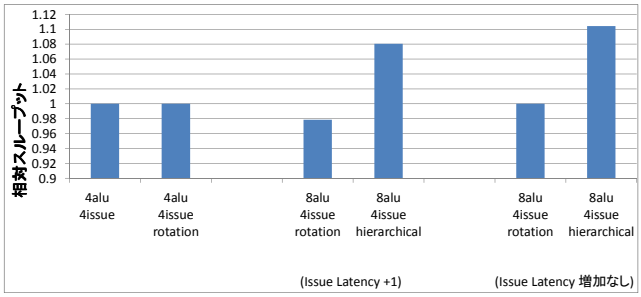

図 13 総スループット

り，図の真ん中 2 本の棒グラフは ALU 増加に伴いレイテンシが増加した場合を，右側の 2 本はレイテンシの増加がなかった場合の値を示している．hierarchical な構成ではレイテンシが増加した場合で約 8.1%の，増加しない場合で周波数向上率そのままの 10.4%のスループット向上を達成している．

5. 命令発行方式の関連研究

機能ユニットへの命令発行に関する研究として，Aviral Shrivastava 等は leakage-aware な発行方式 [9] を提案している．リーク電力は温度に対して指数関数的に上昇する特性があるため，リークセンサによってリーク電流をモニタリングし，その大きいところが温度も高いというように判断することができる．その判断に従い温度の低い，すなわちリークの最も小さい機能ユニットへの命令発行を優先的に行うように制御するものである．センサを用いて測定する実際の値を制御に利用するため，微細化が進むにつれ顕著になるプロセスばらつきにも対応できる点がセンサを用いる手法の特長としてあげられる．

6. まとめと今後の課題

本稿では，ホットなモジュールの 1 つである ALU を対象に，各々の ALU 間ではアクティビティに偏りがあることを確認し，その解消のために ALU のローテーション利用を提案した．また，更にアクティビティを低下するために ALU を増やす場合でも，発行幅を固定したままローテーション利用を行うことで，完全フォワーディングネットワークを必要とせずに回路の複雑化の緩和，およびフォワーディン

グ機能の実現が可能な Partial Forwarding Network を提案した．

シミュレーションによる評価の結果，ALU ローテーションによってアクティビティの均一化が可能であることを確認し，提案方式の内 8alu-4issue-hierarchical の構成では従来方式と比較して，命令発行レイテンシが増加しない場合で 10.4%，1 サイクル増加した場合でも 8.1%のスループット向上が達成された．

今後は，コア内に複数の温度センサを搭載するチップを用いて，実機でのコア内の温度差等を評価して行きたいと考えている．

謝辞 本研究の一部は科研費（課題番号 24700044）による．

参考文献

[1] Black, B. et al.: Die Stacking (3D) Microarchitecture, *MICRO-39*, pp. 469–479 (2006).
[2] Hennessy, J. L. and Patterson, D. A.: *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 5th edition (2011).
[3] Huang, W. et al.: Hotspot: A compact thermal modeling method for CMOS VLSI systems, *IEEE Transactions on Component Packaging and Manufacturing Technology*, Vol. 14, pp. 501–513 (2006).
[4] Intel: Intel Core i7-900 Desktop Processor Extreme Edition Series and Intel Core i7-900 Desktop Processor Series on 32-nm Process (Datasheet, Volume 1).
[5] Intel: CPU monitoring with DTS/PECI (whitepaper) (2010).
[6] Li, S. et al.: McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures, *MICRO-42*, pp. 469–480 (2009).
[7] Mesa-Martinez, F. J. et al.: Characterizing processor thermal behavior, *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS XV, New York, NY, USA, ACM, pp. 193–204 (online), DOI: 10.1145/1736020.1736043 (2010).
[8] Poirier, C. et al.: Power and Temperature Control on a 90nm Itanium-Family Processor, *ISSCC*, pp. 304–305 (2005).
[9] Shrivastava, A. et al.: Reducing Functional Unit Power Consumption and its Variation Using Leakage Sensors, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 18, No. 6, pp. 988 –997 (online), DOI: 10.1109/TVLSI.2009.2019082 (2010).
[10] Skadron, K. et al.: Temperature-aware microarchitecture, *ISCA-30*, pp. 2–13 (2003).
[11] 三輪 忍ほか：小容量 RAM を用いたオペランド・バイパスの複雑さの低減手法，*ACS19*, Vol. 48, No. SIG.13, pp. 58–69 (2007).