2004－ARC－159　(29)
2004／7／31

命令の振る舞いを利用した消費電力削減に関する検討

千代延 昭宏 †　　佐藤 寿倫 †
† 九州工業大学 情報工学部 知能情報工学科

近年の研究でプログラム中のクリティカルパス情報を利用して最適化を行うことで，プロセッサの処理性能向上や省電力化が可能であることがわかっている．これに対し我々は新たなクリティカルパス予測器を提案し評価を行ってきたが，満足のいく結果が得られていなかった．本稿では，処理性能を保ちつつ省電力化を行うために最適な命令スケジューリング方法と演算器の構成を示す．

Using Dynamic Information of Instruction Criticality for Low Power

Akihiro CHIYONOBU†　　Toshinori SATO†
† Department of Artificial Intelligence, Kyushu Institute of Technology

Recent studies show that microprocessor performance or its energy efficiency can be improved, if we utilize information regarding instruction criticality. This paper shows what instruction scheduling method and ALU configuration are the best for energy efficiency and high performance.

1　はじめに

近年携帯情報端末や組み込みシステムにおいても高い処理性能が求められるようになっており，高性能なプロセッサが搭載されている．しかしこれらのシステムには高い処理性能が求められる一方で，その消費する電力も少量であることが要求される．

この問題に対して我々はプログラム実行中のクリティカルパスに着目している．クリティカルパス上の命令を高速かつ消費電力の大きな演算器で実行し，それ以外の命令を低速ではあるが省電力な演算器で実行する省電力アーキテクチャを提案して研究を行っている [6] が，実行される命令がクリティカルパス上の命令かどうか特定する方法が不十分であるため，処理性能維持と省電力の両立に必ずしも成功していない [7]．

これまでの研究で，正確なクリティカルパス情報を得ることができれば処理性能維持と省電力の両立が可能であることがわかっている [8]．本稿では，検討中の省電力アーキテクチャ[7] における命令発行スケジューリングにクリティカルパス情報をどのように利用すべきか検討した結果と，クリティカルパス予測器の予測精度を改善するための手段について検討した結果について述べる．

2　クリティカルパスとその特定機構

クリティカルパスとは命令間の依存関係を結んだ鎖のうち最長のものを結んだ実行パスであり，プログラムの実行時間を決定する命令列である [5]．図 1 に命令列中に現れるクリティカルパスを表すデータ・フローグラフを示す．図 1 において矢印は命令間の依存関係を示す．つまり，依存している命令は矢印の始点にある依存先の命令実行が終了しない限り実行できない．全ての命令のレイテンシが 1 サイクルであるとすると，最も長いパスである命令 I:0 → I:3 → I:4 → I:6 → I:8 を結んだパスがクリティカルパスとなる．

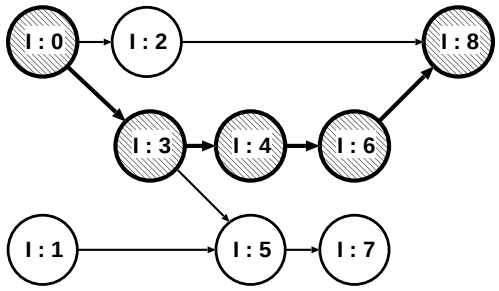

図 1: 命令間の依存関係とクリティカルパス

命令がクリティカルか否かを判断する予測器をクリティカルパス予測器という．クリティカルパス予測に関する研究は近年注目され始めている [1, 4]．Tune らのクリティカルパスヒストリテーブル (CPHT: Critical Path History Table) は各命令ごとのローカルな履歴のみを利用しているが，我々は命令がクリティカルになる要因に命令間の依存関係があることに着目し，命令間のグローバルな相関を予測の際に利用できればクリティカルパス予測の精度が向上するのではないかと考え，命令間の依存関係に着目した 2 レベルクリティカルパス予測器を考案した [6]．しかしそれらの予測器を評価した結果，これまでのところ満足するクリティカルパスの予測精度は得られていない [7]．

小林らはデータ・フローグラフ (DFG: Data Flow Graph) の最長パスを特定するためのパス情報テーブル (PIT: Path Info Table) を提案している [5]．PIT は命令ウインドウ内に存在する命令間の依存関係の情報を持つ．PIT の持つ依存関係の情報から DFG を作成し，その最長パスを特定する．そうすることで，命令ウインドウ内にある命令のクリティカルパスを特定できる．

クリティカルパス予測器と PIT とでは，PIT の方が正確なクリティカルパス情報を提供してくれることがわかっている．本稿では検討中の省電力アーキテクチャにどのように命令を発行すべきかを調査する．そのため，より正確なクリティカルパス情報を用いて命令スケジューリングを行うことが望ましい．よって，本稿で行う評価には PIT が提供するクリティカルパス情報を用いることにする．

3 クリティカルパス情報を用いた命令スケジューリング戦略の検討

クリティカルパス情報を命令スケジューリングにどのように活用すべきかについて検討する．検討中の省電力アーキテクチャでは，どの命令がどの演算器で実行されるかということがプログラムの実行時間，消費電力に大きな影響を与える．このため，どの命令から演算器に発行されるのかということや，命令がどの演算器で実行されるのかということは非常に重要である．

本節では我々の考える省電力アーキテクチャにおいて，発行される命令の選択や命令が発行される演算器の選択をどのように行うべきか検討する．我々は，命令が発行される優先度と命令が発行される演算器の選択方式の組み合わせの中から最も処理性能を低下させずに省電力化を達成できるものを調査した．以下，検討した各方式について説明する．

3.1 命令発行の優先度

命令のスケジュールリングには，演算器に発行できる状態になっている命令を検出する (wakeup) 動作と wakeup された命令の中からどの命令を実際に実行するのか選択する (select) 動作が必要である．このことを考慮して発行される命令の優先度について考えてみる．

発行される命令の優先度は，wakeup された命令がクリティカルパス上にあるか否かということや，wakeup された命令が命令ウインドウにディスパッチされた時間を用いて決めることができる．前者は，処理性能に与えるインパクトを抑えることが期待できる．プログラムの実行時間に影響を与えるクリティカルパス上の命令を優先的に発行するからである．しかし，wakeup されている命令の中からクリティカルパス上の命令を優先的に select するためのロジックが必要となる．後者は，特に付加的なロジックを必要としない．しかしクリティカルパス上の命令を優先的に発行しないため，前者を用いた場合よりも処理性能が低下する可能性がある．

3.2 演算器の選択方法

検討中の省電力アーキテクチャは不均質な演算器を持つ．これらの演算器は，主に命令がクリティカルパス上にあるかないかで使い分けられる．しかし要求された演算器が構造ハザードを起こしていた場合，どのように対処するかいくつかの選択肢が考えられる．この対処方法は処理性能・消費電力に影響を与えると考えられるため，詳しく調査する必要がある．以下，我々が検討する対処方法を説明する．

3.2.1 固定選択方式

検討中の省電力アーキテクチャでは，クリティカルパス上の命令のみが高速な演算器に発行され、それ以外の命令は低速な演算器に発行されるべきである．この方針に厳密に従い，当該命令が発行されるべき演算器にのみ発行されるようにスケジュールを行う．例えばクリティカルパス上の命令が wakeup されたとき，高速な演算器が構造ハザードを起こしていたとする．このとき，当該命令は低速な演算器に対して発行されない．高速な演算器の構造ハザードが解消されるまで，wakeup された状態で待ち続ける．利用可能になった高速な演算器に対してのみ発行される．当該命令がクリティカルパス上になかった場合も同様に低速な演算器に対してのみ発行される．この演算器の選択方式を固定選択方式 (FISS: Fixed Selection Strategy) と呼ぶことにする．

3.2.2 流動選択方式

クリティカルパス上にある命令はプログラムの実行時間を決定する．従って，クリティカルパス上の命令は，wakeup されたらすぐに実行されることが望ましい．この点を考慮して，クリティカルパス上の命令に優先度を与える．当該命令がクリティカルパス上にあった場合，その命令はまず高速な演算器に対して発行されようとする．高速な演算器が構造ハザードを起こしていて発行できない場合は，低速な演算器に発行をされようとする．高速・低速な演算器が同時に構造ハザードを起こしていなければ，次のサイクルを待たずに命令の実行を開始できる．高速・低速な演算器が同時に構造ハザードを起こしている場合は，どちらかの構造ハザードが解消されるのを待って発行される．なお，このような演算器の選択はクリティカルパス上の命令にしか許可しない．クリティカルパス上にない命令は，低速な演算器の構造ハザードが解消されるまで待ち続けることになる．この演算器の選択方式を流動選択方式 (FLSS: Flexible Selection Strategy) と呼ぶことにする．

3.2.3 積極的流動選択方式

これまでの演算器の選択方法の場合，クリティカルパス上にない命令は必ず低速な演算器で実行させられていた．つまり低速な演算器が構造ハザードを起こしていると，必ずストールさせられる．このように演算器を選択させることで，当該サイクルではクリティカルパス上にない命令であっても，実際に実行されるサイクルではクリティカルパス上の命令になってしまう可能性がある．このことを回避する．当該命令がクリティカルパス上にある・ないに関わらず，本来発行されるべき演算器が構造ハザードを起こしている場合，もう一方の演算器に対して発行を試みる．こうすることで，クリティカルパス上にない命令が発行されるサイクルでクリティカルパス上の命令になってしまうことを防ぐことが可能だと考えられる．この演算器の選択方式を積極的流動選択方式 (AFLSS: Agressively Flexible Selection Strategy) と呼ぶことにする．

4 クリティカルパスの探索範囲

本節ではクリティカルパスの探索範囲について検討する．2 節で説明したように，PIT は命令ウインドウ内に存在する命令間の依存関係からクリティカルパスを特定する．つまり，命令ウインドウ内のクリティカルパスを空間的に探索していると言える．命令ウインドウを毎サイクル空間的に探索することで，クリティカルパスの時間的な変化も知ることができる．

我々は，クリティカルパスの時間的な探索範囲を拡大することを試みる．命令ウインドウ内に存在する命令はサイクル毎に異なる．これは，各サイクルで命令が演算器に対して発行されたり新たにディスパッチされたりするためである．このため，命令ウインドウ内に限定したクリティカルパスはサイクル毎に異なることが予想される．我々は命令ウインドウに収まりきらないプログラム全体のクリティカルパスを得るため，命令ウィンドウ内で一度クリティカルパス上の命令だと特定された命令は，以後のサイクルではその情報をずっと保持させておくことにした．こうすることで，より正確にクリティカルパス上にない命令を特定でき，より高い確率でクリティカルパス上にある命令を特定できるようになると考えられる．しかし，クリティカルパスの時間的な変化は追跡できなくなる．

5 評価方法

評価に用いたプロセッサ構成は参考文献 [7] と同様である．本稿では提案するアーキテクチャを整数 ALU に適用して評価する．プロセッサは ALU を 6 個持ち，高速な ALU と低速な ALU のレイテンシはそれぞれ 2 : 1 となるようにする．低速な ALU は，高速な ALU をパイプライン動作させるものに相当し，レイテンシは 2 に増加するが，スループットは 1 のままである．これらの ALU の電源電圧は，高速動作時に 1.1V，低速動作時に 0.7V とする [3].

ベンチマークプログラムは SPEC 2000 CINT から選んだ，bzip，gcc，gzip，parser，vortex，vpr の 6 本である．どのプログラムも，先頭の 1B 命令をスキップし，続く 100M 命令をシミュレーションした．

これまでの評価では処理性能とエネルギー遅延積 (EDP: Energy Delay Product) を用いてきた．しかし今回の評価からは，EDP の代わりにエネルギー遅延二乗積 (ED^2P: Energy Delay^2 Product) を用いる．ED^2P は EDP よりも処理性能に重点を置いた基準である．我々は，単に省電力化を目指して本研究を行っている訳ではない．処理性能の維持，もしくは処理性能の改善と省電力化の両立を目指している．したがって EDP よりも ED^2P を用いて評価を行う方が，我々の考える省電力アーキテクチャ，クリティカルパス情報を用いた命令スケジューリングの成果をより厳しく評価できる．

6 シミュレーション結果

シミュレーション結果を示す．まず，6.1 節で最適な命令発行の優先度と演算器の選択方法について調査した結果を示す．次に，6.2 節でクリティカルパスを探索する範囲の影響について調査した結果と最適な演算器構成について示す．

結果を表すグラフは全て ED^2P を表す．またそれぞれのグラフは，低いほど省電力化が達成できていることを表す．それぞれの結果は，全て高速な演算器の結果によって正規化されている．

6.1 命令スケジューリング戦略

図 2 に最適な命令発行の優先度と演算器の選択方法についての評価結果を示す．図 2 において，`Slow` は全ての演算器が低速な場合を示す．その他の結果において，ラベルのハイフン “–” の左側は 3.1 節で説明された発行される命令の優先度を表す．`CPA` はクリティカルパス上の命令を優先的に select した場合の結果を示している．また，`WO` は wakeup された順番に select れた場合の結果を示している．一方ハイフン “–” の右側は，3.2 節で説明された命令が実行される演算器の選択方法を表している．`FISS` は固定選択方式を用いた場合の結果を，`FLSS` は流動選択方式を用いた場合の結果をそれぞれ示している．`AFLSS` は積極的流動選択方式を用いた場合の結果を示している．

本評価では，高速・低速な演算器の組み合わせは全ての組み合わせを評価した．図 2 に示す結果は，それ

ぞれの命令発行の優先度と演算器の選択方法の組み合わせで最良な結果を示している．

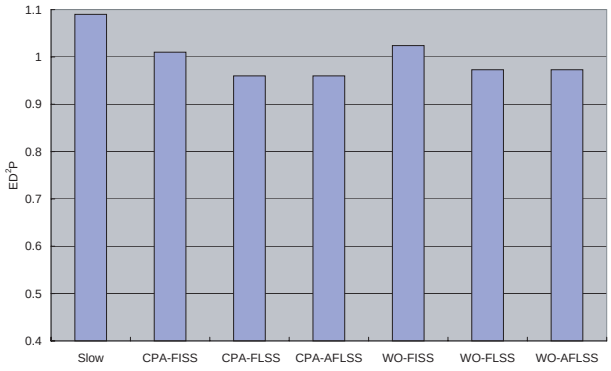

図 2: 発行される命令の優先度と演算器の選択方法

6.1.1 最適な命令発行の優先度

発行される命令の優先度の観点から結果を見てみる．クリティカルパス上の命令から演算器に発行した場合の結果の方が，若干ではあるが wakeup された順番に演算器に発行した場合の結果よりも良い結果を示している．しかし，両者の差は非常に小さい．

命令の発行は，本来ならば wakeup された最古の命令から順番に発行される．このためクリティカルパス上の命令から発行するためには，wakeup された命令の中から優先的にクリティカルパス上の命令を select する付加的なロジックが必要となる．そのようなロジックを新たに付加するということは，ロジックの分だけプロセッサの消費電力が増加するということを意味する．現段階では，クリティカルパス上の命令から優先的に演算器に発行させるロジックの消費電力の見積りはできていない．しかし結果に顕著な差が見られないことから，新たなロジックを追加してまでクリティカルパス上の命令を優先的に発行する必要は低いと言える．よって，命令発行の優先度は wakeup された順番を優先することにする．なお，このロジックの消費電力の見積もりは今後の課題とする．

6.1.2 最適な演算器の選択方法

次に演算器の選択方法の観点から結果を見てみる．3.2.1 節で説明された固定選択方式を用いた場合，全て高速な演算器の場合と比較して約 2%の電力利用効率の悪化が見られる．このため，固定選択方式は適切ではないということがわかる．3.2.2 節で説明された流動選択方式を用いた場合と 3.2.3 節で説明された積極的流動選択方式を用いた場合の結果に差は見られない．両方ともクリティカルパス上にない命令が高速な演算器に発行されていないからだと考えられる．従って，クリティカルパス上にない命令を高速な演算器に発行する必要はないと言える．しかし演算器を選択するロジックの作成を考慮すると，クリティカルパス上の命令のみ両方の演算器に発行可能という具合に限定した設計を行うよりも，全ての命令を両方の演算器に発行可能という具合に冗長性を持たせた設計を行う方がロジックの構成が簡素になると考えられる．このため，演算器の選択方法には積極的流動選択方式を採用することにする．

しかし積極的流動選択方式を採用したとしても，クリティカルパス上の命令を高速な演算器にのみ発行，クリティカルパス上にない命令を低速な演算器にのみ発行するというロジックと比較すると，非常に複雑なロジックになってしまうことが予想される．命令が実行される演算器を選択するロジックについての複雑度，消費電力量などについても見積もりはできていない．よって，この見積もりやロジックの複雑度を低減する検討も今後行う必要がある．

6.2 クリティカルパスの探索範囲の影響

6.2.1 時間的探索範囲の影響

図 3 にクリティカルパスの時間的探索範囲を広げた場合の結果を示す．図中の `Init` は従来の探索範囲を用いた結果を表し，`Non-Init` は時間的探索範囲を広げた場合の結果を表す．また，図 4 に高速な演算器の使用率を示す．図 4 において，縦軸は高速な演算器の使用率を示す．

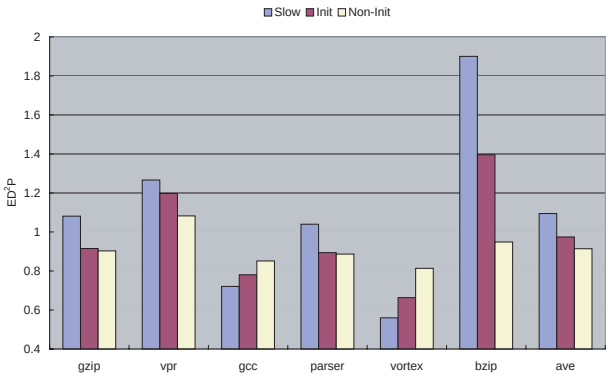

図 3: 時間的探索範囲の影響

ベンチマークプログラムによってやや傾向が異なることがわかる．一部のプログラムでは，電力利用効率が悪化するという結果が得られた．図 4 からわかるように，時間的探索範囲を広げる前は ALU で実行される命令の約 30%が高速な演算器で実行されている．時間的探索範囲の拡大後は，約 65%が高速な演算器で実行されるようになっている．また，全てのプログラムで同様の結果が得られている．それにも関わらず，時間的探索範囲拡大の影響は同一ではない．

我々は，この原因がクリティカルパス上にない命令

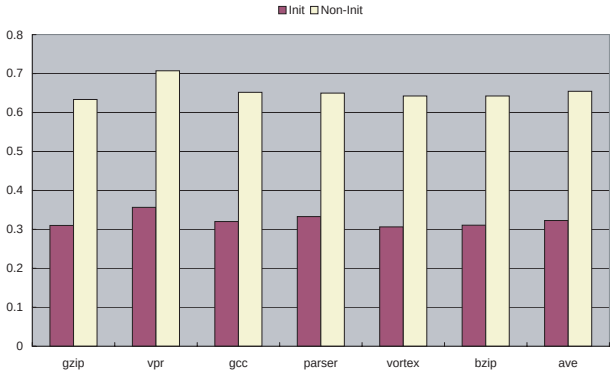

図 4: 高速な演算器の使用率

が高速な演算器で実行されたためと考える．時間的探索範囲を拡大するために，クリティカルパス上の命令になったという情報を命令が発行されるまで保持し続けている．このため，ベンチマークプログラムによってはプログラム全体ではクリティカルパス上の命令ではないが，命令ウインドウ内という限定された範囲ではクリティカルパス上の命令になる命令が多数存在しているのではないかということが推測される．このような命令が多く存在する場合，時間的探索範囲拡大はマイナスに影響すると考えられる．しかし時間的探索範囲を拡大すると，平均では電力利用効率の改善が進んでいる．このため，クリティカルパスの時間的な変化を追跡するよりも時間的な探索範囲を広げた方が良いと言える．

6.2.2　空間的探索範囲の影響と最適な演算器構成

本節ではクリティカルパスの探索範囲，特に空間的探索範囲について調査した結果と最適な演算器の組み合わせについて調査した結果について示す．図 5 に結果を示す．

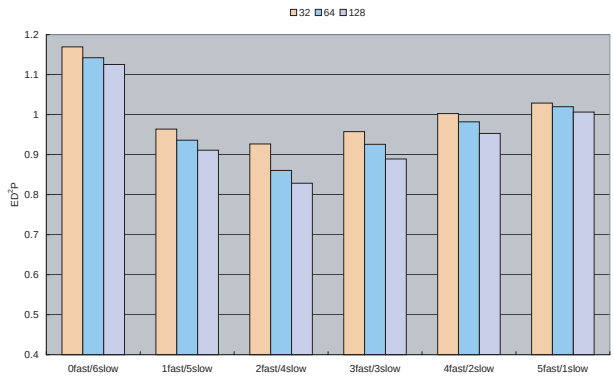

図 5: 空間的探索範囲の影響と最適な演算器構成

図において `0fast/6slow` は，クリティカルパス情報を用いた最適化を適用しない場合を示す．また，この場合の演算器は全て低速な演算器である．`1fast/5slow`, `2fast/4slow`, `3fast/3slow`, `4fast/2slow`, `5fast/1slow` はそれぞれ高速，低速な演算器が，1 : 5，2 : 4，3 : 3，4 : 2，5 : 1 の場合を示している．また `32`，`64`，`128` はそれぞれ命令ウインドウの大きさが，32，64，128 エントリの場合を示している．

まず空間的探索範囲の影響について考えてみる．命令ウインドウが大きくなり，クリティカルパス探索の範囲が広がっていくにつれ，どの演算器の組み合わせにおいても少しずつ消費電力が改善していくことがわかる．このことから，クリティカルパス探索をする範囲を空間的に広げることで，より正確なクリティカルパスが特定できるようになることがわかる．

次に最適な演算器の組み合わせについて考えてみる．消費電力は `2fast/4slow` で最良となり，以後高速な演算器を増やしても改善されない．これは，クリティカルパス上にない命令までもが高速な演算器で実行されているためだと考えられる．この結果から，最も省電力化が達成できている `2fast/4slow` が最適な演算器の組み合わせであると言える．

7　検討中の課題

7.1　クリティカルパス予測の精度改善

6.2 節の評価結果から，クリティカルパスを探索する際の範囲は時間的にも空間的にもできるだけ広くとった方が良いことがわかった．これは，時間的情報と空間的情報を沢山利用した方がより正確なクリティカルパスを特定できることを意味する．

これまでクリティカルパスを予測する際に，命令間の相関関係を示す情報として過去のクリティカルパスの履歴を用いてきた．しかし，クリティカルパス履歴を利用した場合と利用しなかった場合の予測精度に大きな差は見られていない [7, 8]．クリティカルパスの履歴は時間的情報とも空間的情報ともとれるが，PIT の探索範囲がもたらす情報とは性格が異なっていると言える．時間的情報にしろ空間的情報にしろ，PIT がもたらす情報はこれから実行されようとする命令の情報である．一方クリティカルパス予測器が利用する情報は，全て過去に実行された命令の情報である．我々は，このような両者の利用する情報の質の違いがクリティカルパスを特定・予測する際の精度の違いとして現れているのではないかと考えた．そこで我々は，これまでクリティカルパス予測器が利用していなかった，PIT の提供する時間的・空間的情報に近い情報をクリティカルパス予測に利用することを考えている．具体的にどのような情報をどのように予測に用いるかということは，現在検討中である．

7.2 省電力キャッシュメモリ

我々はさらなる省電力化を進めるため，メモリアクセス命令の重要度を利用する省電力キャッシュを提案している [9, 10]．このキャッシュは，クリティカルパス上の命令が参照するデータを重要であるとみなす．データの重要度によって，データの格納場所と記憶階層間でのデータ移動に制約を設ける．この制約により，レベル 1 キャッシュだけでなく記憶階層全体の省電力化を図る．初期評価として，実際にどのくらいのロード命令がクリティカルパス上に存在するかを調査した．

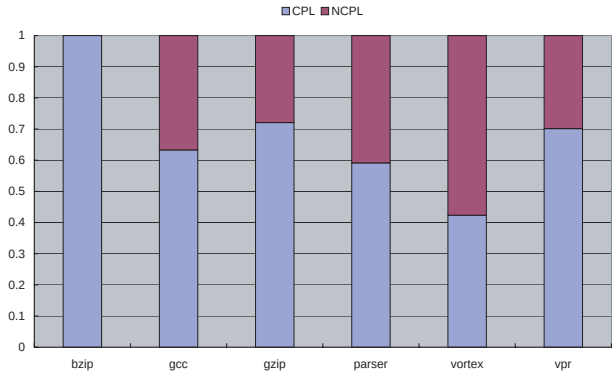

図 6: ロード命令のクリティカリティ

図 6 において，CPL はクリティカルパス上のロード命令の割合を NCPL はクリティカルパス上にないロード命令の割合を示している．bzip 以外のベンチマークプログラムで，CPL と NCPL が混在していることがわかる．クリティカルパス上に無いロード命令の参照するデータは省電力キャッシュに保持できる可能性があり，図 6 は初期評価としては良い結果を示していると言える．

7.3 スラック情報などの利用

クリティカルパス情報以外に命令のクリティカリティを示す情報として，命令のスラック (slack)[2, 11] がある．我々は．省電力アーキテクチャの命令スケジューリングにスラックを用いることを検討している．具体的には，クリティカルパス予測器とスラック情報をハイブリッドに利用して命令をスケジューリングする．クリティカルパスとスラックという異なる性格の情報を同時に利用することで，より正確な命令スケジューリングが実現できることを期待している．

また，クリティカルパス予測にロード命令のキャッシュヒット・ミスの履歴や分岐予測のヒット・ミスの履歴を用いることも検討している．

8 まとめ

命令ウインドウ中のクリティカルパスを特定する PIT を用いて，我々の考える省電力アーキテクチャにおける最適な命令発行の優先度，最適な演算器の選択方法，最適な演算器の組み合わせついて調査を行った．その結果，wakeup された順序を優先度とし，命令発行を遅らせないよう空いている演算器に命令を発行する演算器の選択方法，高速な演算器を 2 つ低速な演算器を 4 つという演算器の組み合わせが最も良いということがわかった．またこれらの調査結果を分析していく過程で，クリティカルパス予測器の予測精度を改善するために検討すべき方向を明らかにすることができた．

今後はクリティカルパス予測器の予測精度を改善するため，具体的にどのような時間的・空間的情報を予測に用いればよいか検討し，評価を行っていく予定である．

謝辞

本研究の一部は，科学研究費補助金 基盤研究 B(2) (#16300019)，および北田奨学会記念財団 (#03–003) の援助によるものです．

参考文献

[1] B.Fileds, S.Rubin, R.Blodik: “Foucusing Processor Policies via Critical-Path Prediction”, the 28th International Symposium on Computer Architecture, July 2001.

[2] J.Casmira and D.Grunwald, “Dynamic Instruction Scheduling Slack”, Kool Chips Workshop, December 2000.

[3] M.Levy: “SAMSUNG Twists ARM Past 1GHz”, Information Quarterly, vol.1, no.1, 2002.

[4] E.Tune, D.Liang, D.M.Tullsen, B.Calder: “Dynamic Prediction of Critical Path Instructions”, the 7th International Symposium on High Perdormance Computer Arcitecture, January 2001.

[5] 小林良太郎, 安藤秀樹, 島田俊夫: “データフロー・グラフの最長パスに着目したクラスタ化スーパースカラ・プロセッサにおける命令発行機構”, 2001 年並列処理シンポジウム JSPP2001, 2001 年 6 月.

[6] 千代延昭宏, 佐藤寿倫, 有田五次郎: “低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の提案”, 情処研報 2002-ARC-149 , 2002 年 8 月.

[7] 千代延昭宏, 佐藤寿倫, 有田五次郎: “低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の評価”, 電子情報通信学会論文誌　和文論文誌 C, Vol.J86-C, No.8, 2003 年 8 月.

[8] 千代延昭宏, 佐藤寿倫: “プログラムの実行時における命令の重要度決定に関する検討”, 情処研報 2003-ARC-154, 2003 年 8 月.

[9] 千代延昭宏, 佐藤寿倫: “メモリアクセス命令の重要度を利用したキャッシュメモリの省電力化”, 情処九州支部 火の国シンポジウム, 2004 年 3 月.

[10] 藤井誠一郎, 千代延昭宏, 佐藤寿倫: “データの重要度を利用した省電力キャッシュ”, 先進的計算基盤システムシンポジウム SACSIS2004, 2004 年 5 月.

[11] 劉小路, 小西将人, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: “クリティカリティ予測のためのスラック予測”, 先進的計算基盤システムシンポジウム SACSIS2004, 2004 年 5 月.