社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－170 (6)
2006／11／29

省電力に配慮したスケーラブルな命令ウインドウの提案と評価

渡辺 慎吾† 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
‡ 九州大学 システムＬＳＩ研究センター

現在のマイクロプロセッサは命令レベルの並列性を利用して高い処理性能を実現している．処理性能を向上させるためには，より多くの命令から並列性を抽出する必要があるが，これには大容量の命令ウィンドウが必要である．しかし，命令ウィンドウは連想メモリで構築されているため，消費電力と速度の観点から容易に容量を増加させることは困難である．そこで，本稿では連想メモリを用いない省電力に配慮した命令ウィンドウを提案する．これはデータ依存に基づき命令を明示的に WakeUp することで実現される．提案手法を評価した結果，従来の命令ウィンドウに対して，性能は平均 1.9%低下したが，動的消費電力は平均 84%削減となった．また，静的消費電力についても 28%削減できることが確認された．

An Energy-Efficient Scalable Instruction Window

SHINGO WATANABE† AKIHIRO CHIYONOBU† TOSHINORI SATO‡

†Departoment of Artificial Intelligence, Kyushu Institute of Technology
‡Sytem LSI Research Center, Kyushu University

Modern microprocessors attain high performance using ILP (Instruction Level Parallelism). In order to exploit ILP for improving processor performance, instruction window size should be increased. However, it is difficult to increase the size, since instruction window is implemented by CAM whose power and delay are much large. This paper introduces low power and scalable instruction window that replaces CAM with RAM. In this window, instructions are explicitly woken up. Our evaluation results show that our instruction window decreases performance only by 1.9% on average. Furthermore, dynamic power is reduced by 84% on average and static power is reduced by 28% on average.

1. はじめに

近年マイクロプロセッサの消費電力の急増が問題となっており，性能向上の制約の要因の１つとなっている．また，半導体の製造プロセスのディープサブミクロン化により，今後トランジスタの性能が大きく改善されることは期待できない．そのため，プロセッサの消費電力を抑制しつつ性能を改善するには，マイクロアーキテクチャレベルでの消費電力の低減が求められる．

現代のマイクロプロセッサでは命令レベルの並列性 (Instruction Level Parallelism:ILP) を利用することで高い性能を実現している．将来，さらにプロセッサの性能を改善するにはより ILP を活用する必要がある．そのために，大容量の命令ウィンドウが求められる．しかし命令ウィンドウは消費電力の大きな機構の１つであり，容易に大容量化することはできない．命令ウィンドウでは，データ依存が解決したことを後続の命令に伝えるために連想検索を用いている．連想検索は連想メモリ (Content Addressable Memory:CAM) で実現されているが，このことが命令ウィンドウの大きい消費電力の要因となる．命令ウィンドウから CAM を取り除くことで，省電力かつ大容量な命令ウィンドウの実現が可能である．

CAM は回路上の性質から動作速度が遅く，プロセッサの動作速度向上の制約となっており，従来から CAM を取り除いた命令ウィンドウ[1)5)6)7)] が提案されいる．しかし，それらは性能改善を目的としており消費電力に配慮していない．また，消費電力に関する評価を行っていない．

本稿では消費電力に着目し，省電力に配慮したスケーラブルな命令ウィンドウの提案と評価を行う．２節で提案する命令ウィンドウについて述べる．３節で評価手法について述べ，４節で評価の結果について述べる．５節でまとめとする．

2. ITS 命令ウィンドウ

本節では，省電力かつスケーラブルな命令ウィンドウである ITS 命令ウィンドウ (Indirect Tag Search Instruction Window) を提案する．ITS 命令ウィンドウでは，データ依存における先行命令とその演算結果を入力とする後続の命令を WakeUp リストと呼ばれる線形リストで管理する．WakeUp リストを用いるこ

とで，先行命令の実行完了を後続命令に明示的に通知することができる．本稿ではデータ依存において，オペランドを生成する先行命令を生産者，そのオペランドをソースオペランドとする後続命令を消費者と呼ぶ．また，生産者の実行完了を消費者に通知し，実行可能状態へ遷移させることを WakeUp と呼ぶ．

2.1 WakeUp リスト

従来の命令ウィンドウでは生産者の実行完了をブロードキャストすることで消費者を WakeUp している．そのために命令ウィンドウには CAM が必要である．これに対し ITS 命令ウィンドウでは，生産者の実行完了後に消費者を明示的に WakeUp する．これにより，CAM を取り除くことができる．消費者を明示的に WakeUp するには，命令間のデータ依存の情報を持つことが不可欠である．ITS 命令ウィンドウは従来の命令ウィンドウが管理している情報に加えデータ依存の情報を管理することで，消費者の明示的 WakeUp を可能とする．データ依存の情報は図 1 のようにリストで管理する．このリストを辿ることで消費者を WakeUp する．データ依存の関係を表すものとして，図 2 に示す Dataflow グラフがある．Dataflow グラフは生産者から消費者への枝を持つ．対して提案するリストは消費者から消費者へ枝を持つ．消費者は生産者から始まる 1 本のリストより管理される．本稿ではこのリストを WakeUp リストと呼ぶ．

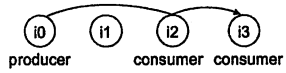

図 1 WakeUp リスト

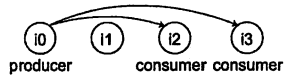

図 2 Dataflow グラフ

図 3 に ITS 命令ウィンドウの構成を示す．ITS 命令ウィンドウは，オペコードやタグを保存する従来の命令ウィンドウが持つフィールドと，WakeUp リストを管理するためのフィールドとで構成される．新たに追加されるフィールドにノードポインタを保存することで，WakeUp リストの情報を管理する．ノードとは命令ウィンドウの各エントリのことを示す．ノードポインタとはノードの識別子である．

WakeUp リストではノードは必ずリストの末尾に追加される．そのため，末尾のノードがどのエントリであるか知る必要がある．TNT(Tail Node Table) は末尾ノードを管理するためのテーブルである．TNT はそれぞれの生産者に対応する WakeUp リストの末尾ノードポインタを保存する．末尾ノードポインタは生産者のディスティネーションレジスタと対応づけられる．消費者は自身のソースレジスタ番号を用い TNT を参照することで生産者の末尾のノードポインタを得ることができる．TNT は生産者のディスティネーションレジスタと WakeUp リストの末尾ノードポインタを対応づけることができればよいので，マップテーブルやレジスタファイルを拡張することでも実現できる．

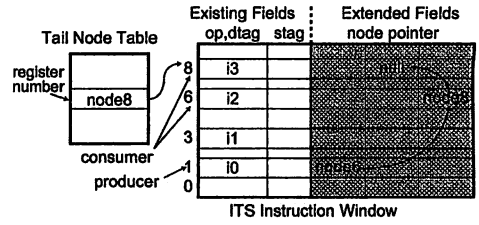

図 3 ITS 命令ウィンドウ

続いて WakeUp リストの作成手続きを述べる．この手続きは命令のディスパッチ時に行われ，大きく 2 つの操作に分けることができる．命令が挿入された命令ウィンドウのノードポインタを TNT に書き込む操作と，命令を WakeUp リストに追加する操作である．前者では，命令のディスティネーションレジスタ番号に対応する TNT のエントリに，当該命令が割り当てられた命令ウィンドウのノードポインタが書き込まれる．後者では，命令のソースレジスタ番号で TNT が参照され，得られたノードポインタが指すノードに当該命令のノードポインタが書き込まれる．

消費者の 2 つソースオペランドがともに利用できないときは，2 つの生産者に依存していることになる．つまり，消費者は 2 本の WakeUp リストで管理されることになる．また，命令は同時に生産者にも消費者にもなり得る．したがって，ノードポインタを保存するフィールドは 3 つ必要になる．ITS 命令ウィンドウでは生産者としてのノードポインタを保持するフィールドと，右ソースオペランドと左ソースオペランドのためのフィールドをそれぞれ用意する．

2.2 分岐予測ミス時の WakeUp リストの振る舞い

分岐予測ミスが発生すると WakeUp リストは誤った情報を持つことがある．その状態が発生するのは分岐命令を越えて WakeUp リストが作成されているときである．図 3 を例に説明する．i0 が最も古い命令で i3 が最も新しい命令であり，i0 が生産者で i2 と i3 が消費者である．WakeUp リストは正しく作成されている．そして i1 が分岐命令であり，分岐予測ミスが判明する前の状態である．分岐予測ミスが判明すると i2 と i3 が破棄される．この状態では TNT の末尾ノードのポインタと i0 のノード（node1）の保持するノードポインタが誤った状態となる．次に正しい分岐パス上の命令がディスパッチされると 2 つの問題が生じる．1 つ目は新たに i0 の消費者が現れた場合である．消費

者は正しい末尾ノードポインタを得ることができず，WakeUp リストが正しく作成されない．2 つ目は i0 の消費者が存在しない場合である．node1 の保持するポインタが node6 を指したままであり，node1 が末尾ノードであることを識別することができない．この状態のリストを用いると本来の消費者でない命令まで WakeUp してしまう．本稿では本来の末尾ノード以降の誤った消費者を持つノードを違反ノードと呼ぶ．

前者の問題は TNT の状態を投機実行直前の状態に戻すことで解決する．これはマップ表などと同様にチェックポイントを作成することで実現できる．後者の問題はリストに属する消費者が正しい消費者か否かを識別する方法を新たに追加することで解決する．生産者のディスティネーションオペランドのタグと消費者のソースオペランドのタグは必ず一致することを利用する．タグが一致しない場合はその消費者は本来の消費者ではないことが識別できる．WakeUp リストでは消費者のノードを直接参照することができる．消費者のソースオペランドを読み出して，生産者のディスティネーションオペランドのタグと比較すればよい．この方法はブロードキャストを必要としない．

2.3 WakeUp 機構

図 4 に WakeUp 機構を示す．WakeUp 機構はリストの先頭ノードポインタとディスティネーションオペランドのタグを保持する WakeUp キュー，タグ比較する比較器，ITS 命令ウィンドウへのアクセスを制御する WakeUp ロジックからなる．実行が完了した生産者が持つノードポインタとタグは WakeUp キューに追加される．1 本のリストにおける WakeUp の手続きを説明する．WakeUp ロジックはキューからノードポインタを読み出す．次に，命令ウィンドウの該当ノードにアクセスし，消費者のソースオペランドのタグと次に WakeUp するノードのポインタを読み出す．最後に，生産者のディスティネーションオペランドのタグと消費者のソースオペランドのタグを比較し，一致すれば消費者を WakeUp する．この時点で 1 つのノードの WakeUp が終わる．さらに読み出された次のノードポインタに従って命令ウィンドウにアクセスし WakeUp の手続きを続行する．これを繰り返すことでリスト上のすべての消費者を WakeUp する．

ある WakeUp リストの手続きが終了する条件は 2 つある．1 つはタグが一致しない場合である．そのノードは違反ノードであるため WakeUp してはならない．さらに後続のノードが存在してもすべて違反ノードであるため手続きを終了してよい．もう 1 つは読み出されたノードポインタが null ポインタである場合である．この場合，WakeUp ロジックがアクセスしているノードが末尾ノードである．そのノードの WakeUp を行い，その後手続きを終了する．

WakeUp ロジックは 1 サイクルで 1 つのノードしか WakeUp できないとする．また，複数の WakeUp リストを同時に辿るには，複数の WakeUp ロジックと比較器を用いる．WakeUp キューから同時に複数のノードポインタとタグを読み出し，それぞれに WakeUp ロジックと比較器を割り当て個別にリストを辿る．

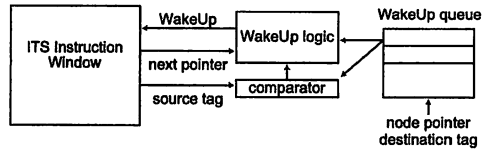

図 4 WakeUp 機構

3. 評価環境

本稿で用いる評価環境について述べる．アーキテクチャレベルの環境でプロセッサの性能を評価し，回路レベルの環境で消費電力に関する評価を行う．さらに，回路レベルの環境で得られた結果とアーキテクチャレベルで得られた結果を組み合わせプロセッサの動作時の消費電力を評価する．

3.1 プロセッサ構成

アーキテクチャレベルでの評価は SimpleScalar ツールセット[2] に ITS 命令ウィンドウを実装して行う．評価に用いたプロセッサの構成を表 1 に示す．命令ウィンドウにはレジスタ更新ユニット (RUU:Register Update Unit) を用いる．RUU は命令ウィンドウとリオーダバッファを共有しているが，ITS 命令ウィンドウをモデリングできる．ITS 命令ウィンドウにおいて，WakeUp 機構が同時に辿ることができる WakeUp リストの本数は命令発効幅と同じ 4 本とし，リスト当たり 1 クロックサイクルにつき 1 ノード Wakeup 可能とする．

表 1 プロセッサの基本構成

ISA	PISA
issue width	4 instructions
commit width	4 instructions
branch pediction	bimodal and gshare
RUU size	128 entry
LSQ size	64 entry
L1 I cache	64 KB
L1 D cache	64 KB
L2 cache	2048 KB

評価には SPEC2000 及び MediaBench ベンチマークプログラムを使用する．SPEC2000 においては，はじめの 5 億命令をフォワードして，その後の 1 億命令をシミュレーションする．入力ファイルは ref を用いる．MediaBench ではフォワードせず，すべてシミュレーションする．

3.2 消費電力の見積もり

消費電力の見積もりは，RAM と CAM の回路を設計し，HSPICE を用いて行う．図 5 に設計した 1bit の SRAM セルを，図 6 に RAM アレイを示す．評価

は基本的な回路を対象とし，デコーダ，タイミング制御回路等は対象としない．RAM はリード用とライト用のポートをそれぞれ 4 ポート持ち，128 ラインとする．各ラインのビット数は命令ウィンドウのフィールドのビット数と同じとする．従来の命令ウィンドウでは，オペコードが 16 ビット，ディスティネーションオペランドのタグが 7 ビットであるので合計 23 ビットになる．ITS 命令ウィンドウではオペコード，ディスティネーションオペランドのフィールドに加えソースオペランドのタグのフィールドも RAM で構成される．ソースオペランドのタグは 7 ビットであるので合計 40 ビットになる．また，WakeUp リストのノードポインタのビット数は 9 ビットとし，3 つフィールドがあるので合計 27 ビットになる．

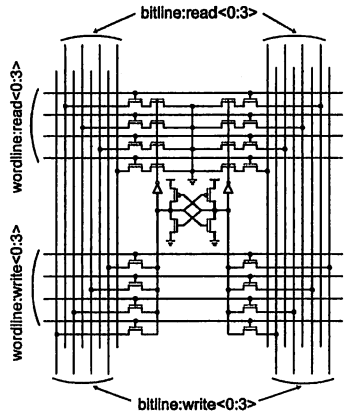

図 5 SRAM セル

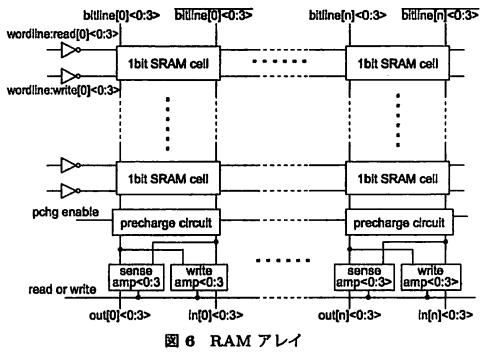

図 6 RAM アレイ

CAM は RAM を拡張して設計する．各ラインに比較器を 4 つ持たせ，各 SRAM セルに比較器用のリードポートを 4 つ追加する．また，各ラインまでタグ情報を送り込むためのライトアンプを付加する．CAM の構成はリード用とライト用，さらにブロードキャスト用のポートをそれぞれ 4 ポート持ち，128 ラインとする．各ラインのビット数はソースオペランドのタグのビット数と同じとし，右ソースオペランド用と左ソースオペランド用を 2 機並べる．

CMOS モデルは UC Berkeley の PTM (Predictive Technology Model)[3][4] を用いる．プロセスルールは 65nm 世代のものを使用する．配線幅と配線ピッチも同様に PTM を参考にする．表 2 に詳細をまとめる．CMOS のサイズは SRAM セルで用いたものである．また，一般に多ポート RAM は配線の面積で決定される[6] ことから，1bit の SRAM の面積は配線幅と配線ピッチから計算し求める．SRAM セルやセンスアンプ，比較器などの回路は文献 6) を参考にした．

表 2 CMOS 及び配線のパラメータ

プロセスルール	65nm
電源電圧	1.1v
トランジスタサイズ	nMOS L:65nm, W:0.1μm pMOS L:65nm, W:0.2μm
配線幅	0.1μm
配線ピッチ	0.1μm

本稿では TNT については評価をしていない．TNT は末尾ノードを記憶するためのものであるから，論理レジスタ数と同じエントリ数を持てばよい．各エントリはノードポインタと同じビット数である．また，チェックポイントを作成するため，チェックポイントの回数分の TNT を記憶できる容量の RAM が必要となる．

4. 評価結果

本節では，ITS 命令ウィンドウが性能に与える影響と消費電力の削減効果について述べる．

4.1 性能に与える影響

図 7 に従来の命令ウィンドウを用いたプロセッサと ITS 命令ウィンドウを用いたプロセッサの性能を比較した結果を示す．縦軸に IPC (Instructions Per Cycle) を示す．従来手法を用いた場合の IPC で正規化されている．横軸はそれぞれベンチマークプログラムを示している．

提案手法は従来手法に対して平均で 1.9%IPC が低下している．しかし，最も IPC が低下した mcf でも 6.3%の低下にすぎない．4%以上低下したベンチマークプログラムは 4 つのみである．以上の結果から，提案手法がプロセッサの性能に与える影響は小さいと言える．この理由は，約 9 割の命令において，1 つの生産者に対して消費者が高々 1 つであり，WakeUp で遅延が発生する消費者がほとんど存在しないためである[8]．

次に，2.2 節で述べた分岐予測ミスによる違反ノードについて述べる．提案手法では分岐ミス時に違反ノードが発生する．違反ノードが存在すると WakeUp ロジックを 1 サイクル多く占有してしまい，後続の WakeUp リストの WakeUp の開始が遅れる．これは性能に悪影響を与える可能性がある．図 8 に全 WakeUp リスト中，

違反ノードを含むものの割合を示す．違反ノードを含むWakeUpリストは平均で2.2%存在する．adpcmD, adpcmE, mpeg2Eの3つのベンチマークプログラムはその他のプログラムと比べて割合が高くなっている．しかし，図7からこの3つのベンチマークプログラムで特に性能低下が起きていることを読みとることはできない．このことから，違反ノード数と性能低下には強い相関関係は無いと言える．

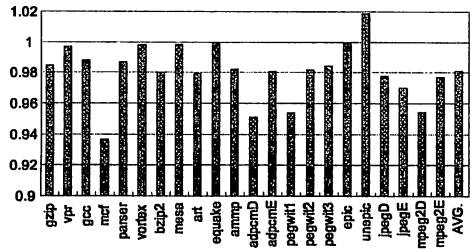

図7 プロセッサ性能

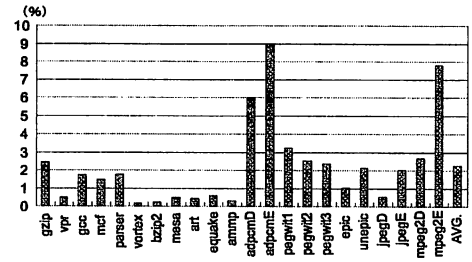

図8 違反ノードを含むWakeUpリストの割合

4.2 ITS命令ウィンドウの消費電力に関する評価

ITS命令ウィンドウにおける消費電力について述べる．はじめに静的消費電力について評価し，次に動的消費電力について評価する．

4.2.1 静的消費電力

図9に従来の命令ウィンドウとITS命令ウィンドウの静的消費電流を示す．縦軸は単位時間当たりの電流量である．baseが従来の命令ウィンドウを示す．図9で最も特徴的であるのは，従来の命令ウィンドウのCAM領域の消費電力である．RAM領域よりも容量が小さいにも関わらずCAM領域の電力がRAM領域の電力を上回っている．この理由は次の2つある．1つはCAMセルのポート数がRAMセルより多い点である．もう1つは各ラインにつき4つの比較器を持っている点である．このため，トランジスタ数の合計がRAM領域よりCAM領域の方で多くなっている．続いて，従来の命令ウィンドウとITS命令ウィンドウを比較する．ITS命令ウィンドウはすべてRAMで構成されているためCAM領域が無い．RAM領域を比較すると，従来の命令ウィンドウに対してITS命令ウィンドウが1.6倍の消費電力となる．これはRAM領域に新たにWakeUpリストの管理フィールドが加わるためである．しかし合計の消費電力では，CAM領域の影響が大きいためITS命令ウィンドウの方が28%減少している．静的消費電力ではITS命令ウィンドウは有効であると言える．

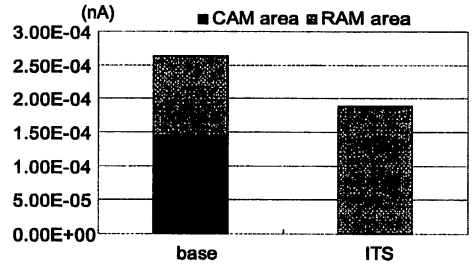

図9 静的消費電流

4.2.2 動的消費電力

回路レベルでの動的消費電力について評価する．図10は1命令が1エントリに対して操作するときに発生する消費電流である．縦軸は操作で消費された電流量である．横軸はそれぞれの操作を示している．dispatch writeとissue readは，それぞれ命令ウィンドウに対するディスパッチ時の書き込みとイシュー時の読み出しである．broadcastは1エントリが1命令分のブロードキャストを受け取りタグ比較するときの電流である．list node writeはWakeUpリストを作成するためにノードポインタを書き込むときの電流である．wake upはWakeUpリストを辿りWakeUpをするときの電流である．それぞれについて評価する．

同図で最も特徴があるのはdispatch writeである．従来の命令ウィンドウに対して約2倍近く電力を消費している．ITS命令ウィンドウではノードの初期化を含め，エントリのすべての領域に書き込みをしている．そのため，書き込みのビット数が約2倍になる．書き込みビット数の増加が電力の増加につながっている．issue readについては同じビット数を読み出しているため，消費電力は同じである．list node writeはリストを作成するためのものであり，ITS命令ウィンドウでのみ消費される．また，書き込みのビット数が小さいにも関わらずissue readより消費電力が大きいのは，評価に用いた回路モデルが読み出しより書き込みの方が多く電力を消費するためである．最後に，broadcastとwake upを比較する．どちらも1エントリに対して1命令が行った操作の結果であるが，broadcastの方が消費電力が大きくなっている．これは操作手順の違いのためである．broadcastは生産者のタグ情報を全エントリに放送し，タグ比較を行っている．対してwake upでは消費者のタグ情報を読み出して比較している．タグ情報を放送することは回路の動作上，書き込みと同じことである．つまり，list node writeと同

様に回路の特性の影響である．

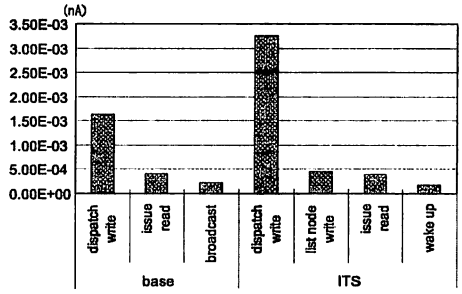

図 10　動的消費電流

最後にアーキテクチャレベルで命令ウィンドウの動的消費電力について評価する．評価はプログラムを実行したときの命令ウィンドウに対するそれぞれのアクセス数とアクセス当たりの動的消費電力を乗して行う．図 11 に従来の命令ウィンドウを，図 12 に ITS 命令ウィンドウの結果を示す．縦軸は消費電力量であり，どちらも従来の命令ウィンドウの全電力量で正規化されている．まず，合計の消費電力は従来の命令ウィンドウに対して ITS 命令ウインドウは平均で 84%減少している．最も良い値を示す vortex は 88%の，最も悪い値を示す adpcmE でも 78%の減少である．ITS 命令ウィンドウは大幅に消費電力を削減することに成功している．最も消費電力の削減に影響したものは WakeUp に関する電力である．従来の命令ウィンドウでは WakeUp のためのブロードキャストによる消費電力が全体の消費電力の 9 割を占めている．ITS 命令ウィンドウはブロードキャストを用いないため WakeUp に関する電力が非常に小さくなっている．また，ITS 命令ウィンドウでは書き込みビット数が増えることによってディスパッチ時の書き込みの電力が従来の命令ウィンドウに対して増加している．さらに，wake up list を作成するための node の書き込みの電力が追加される．しかしながら，それらの増加分は非常に小さく問題にならない．したがって，ITS 命令ウィンドウは消費電力の大幅な削減を実現している．

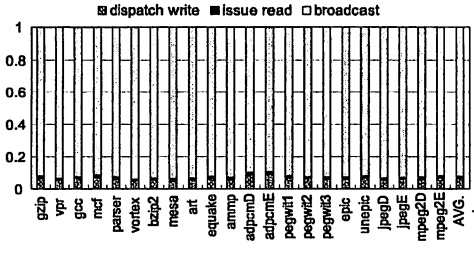

図 11　消費電力量：従来の命令ウィンドウ

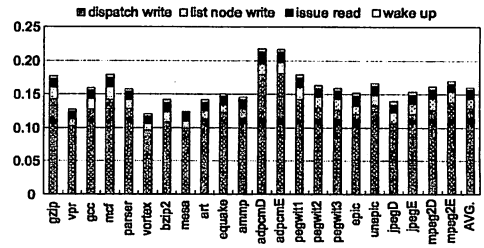

図 12　消費電力量：ITS 命令ウィンドウ

5. ま と め

近年，プロセッサの消費電力が問題となっており，消費電力に配慮したマイクロアーキテクチャの要求が高まっている．本稿では省電力かつスケーラブルな ITS 命令ウィンドウを提案し，評価を行った．評価の結果，ITS 命令ウィンドウは従来の命令ウィンドウに対して平均で 1.9%の性能低下となった．また，動的消費電力は従来手法に対して，平均で 84%の削減となった．静的消費電力に関しても 28%削減できることが確認できた．したがって，提案手法は性能低下を抑制しつつ，高い消費電力削減効果を発揮することが確かめられた．

謝　辞

本研究の一部は，文部科学省科学研究費補助金 (No. 16300019, No.176549) の援助によるものです．

参 考 文 献

1) E.Brekelbaum, J.Rupley, C.Wilkerson, B.Black, "Hierarchical Scheduling Windows", 35th Annual International Symposium on Microarchitecture, 2002.
2) D.Burger, T.M.Austin, "The SimpleScalar tool set version 2.0", ACM SIGARCH Computer Architecture News, vol.25, no.3, 1997.
3) Y.Cao, T.Sato, D.Sylvester, M.Orshansky, C.Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design", IEEE Custom Integrated Circuits Conference, 2000.
4) W.Zhao, Y.Cao, "New generation of Predictive Technology Model for sub-45nm design exploration", Leading Design for Quality and Manufacturbaility, 2006.
5) 石井 康雄, 平木 敬, "命令ウィンドウ拡張による命令レベル並列性の利用", 並列/分散/協調処理に関する『青森』サマー・ワークショップ, 2004.
6) 五島 正裕, "Out-of-Order ILP プロセッサにおける命令スケジューリングの高速化の研究", 京都大学 博士論文, 2004.
7) 佐藤 寿倫, 有田 五次郎, "連想検索を取り除いたスーパースカラプロセッサ向け命令発火機構", 並列処理シンポジウム, 2001.
8) 渡辺 慎吾, 千代延 昭宏, 佐藤 寿倫, "スーパースカラプロセッサにおける命令依存関係の特徴調査", 電子情報通信学会九州支部学生講演会, 2005.