卒業研究論文

パイプラインステージ統合手法における動作タイミング調整による改良

指導教員　　松尾 啓志 教授
　　津邑 公暁 准教授

名古屋工業大学 工学部 情報工学科
平成 19 年度入学　19115079 番

祖父江 宏祐

平成 23 年 2 月 8 日

パイプラインステージ統合手法における動作タイミング調整による改良

祖父江 宏祐

内容梗概

プロセッサの高性能化に伴い，消費電力は増大する傾向にある．これに対して現在，動的な電源電圧制御手法 (DVS) がモバイルプロセッサで採用されている．DVS は，プロセッサ負荷に応じて電源電圧とクロック周波数を変動させ省電力化する手法である．しかし，DVS の効果はプロセッサの標準電源電圧とトランジスタが動作する最低電圧 (閾値電圧) の差に依存し，今後標準電源電圧が低下する一方，閾値電圧の低下があまり見込めないことから，将来的に消費電力の削減効率の低下が予測されている．そこで，これらのことに依存しない DVS に変わる省電力化手法として，動的パイプラインステージ統合手法 (PSU) が提案されている．PSU はパイプラインステージを統合することで省電力化し，パイプライン負荷に応じて動的にパイプライン状態を変更することによって性能低下を防ぐ．しかし，PSU は負荷を検知するタイミングで負荷が変動しなければパイプライン状態を変更することができないという問題点をもつ．この PSU の問題点を解決した手法として負荷情報を用いた PSU がある．この手法は，既存の PSU に buffer を追加し，負荷情報を記憶する．これにより，PSU では対応できないようなパイプライン負荷の上昇に対応し，パイプライン状態を変更する．しかし PSU や負荷情報を用いた PSU は，詳細にプロセッサの状態を制御しているわけではない．そこで，パイプライン負荷の局所的な変動に対応することで，さらに省電力化を狙えると考え，負荷が低下する区間の調査を行った．調査結果から命令キャッシュミスによるメモリアクセス頻発区間では，数十サイクル程度の間隔でメモリアクセスが発生していることを確認した．

そこで本研究では，命令キャッシュミスが発生したときにプロセッサを一定期間低速状態に切り替える手法を提案する．評価には SimpleScalar-PISA 命令セットシミュレータを用い，実行は Out-of-Order 実行で行った．従来手法と比べ，ED 積削減率がメモリアクセスレイテンシ 32cycle で平均で 0.5%，最大で 1.0%，50cycle で平均で 0.6%，最大で 1.3%，100cycle で平均で 1.1%，最大で 2.4%，ED^2 積削減率が 32cycle で平均で 0.01%，最大で 0.02%，50cycle で平均で 0%，最大で 0.5%，100cycle で平均 0.06%，最大で 1.5% であることを確認した．

パイプラインステージ統合手法における動作タイミング調整による改良

目次

1 はじめに

プロセッサの高性能化に伴い，消費電力は増大する傾向にある．しかし，これまでのプロセッサ研究は，性能向上技術に着目したものが大部分であった．一方で，低消費電力が求められるモバイルコンピュータに高性能なプロセッサを搭載するために，プロセッサの省電力化への要求が強まっている．省電力化は，バッテリーや放熱ファンが小型化できるため，駆動時間，静音化の観点からも重要である．また，従来よりも少ない電源容量で従来と同程度の性能を確保できるということから，ハードウェアコスト削減の面でも有効である．この要求に対し現在のプロセッサでは，電源電圧とクロック周波数を動的に変更することで対処している．この手法は **DVS(Dynamic Voltage Scaling)** と呼ばれ，プロセッサの負荷が低い時に電源電圧とクロック周波数を低下させ省電力化する手法である．DVS は，消費電力を削減できる有効な手段であるが，半導体製作技術の進歩とともに削減できる電力が減少すると考えられる．それは，プロセッサの標準電源電圧が低下する一方，トランジスタが動作する最低電圧 (閾値電圧) が低下しないと予想されているためである．

このような背景から，DVS に代わる省電力化手法として，**PSU(Pipeline Stage Unification)** と呼ばれるパイプラインステージを動的に統合する手法が提案されている．PSU はプロセッサの負荷が低い時や分岐予測ミスが頻発する時などに，パイプラインステージ間のパイプラインレジスタを停止することで，パイプラインステージを統合する．これにより，実行速度低下を抑えつつパイプラインレジスタ部で使用される消費電力を削減する．

この PSU の動作を改良した手法として，**負荷情報を用いた PSU** がある．この手法は PSU の制御に加え，負荷上昇した区間開始地点のプログラムカウンタ (PC) を，新たに追加した buffer に記憶しておき，再びその PC を通過した時，強制的に通常状態に戻す．

しかし，現在提案されている PSU や負荷情報を用いた PSU には問題がある．これらのステージ統合手法は，定期的に IPC(Instruction Per Cycle) を調査することによりパイプライン状態を変更する．しかし，定期的な IPC 計測のみではパイプライン負荷の局所的な変動に対応できない可能性がある．そこで，本研究ではパイプラインの負荷が低下するメモリアクセスに着目し，PSU の制御に加え，メモリアクセスが頻発する区間での PSU の制御手法を提案する．

以下 2 章では研究背景として従来手法である DVS と PSU，負荷情報を用いた PSU

を説明し，これらの手法の問題点を挙げる．3 章では，本論文で提案する PSU の更なる省電力化を実現する命令キャッシュミス監視によるプロセッサ状態制御手法について述べ，4 章で提案手法の具体的な実装方法について述べる．5 章で汎用ベンチマークプログラムである SPEC CPU95 ベンチマークによるシミュレーションによる評価と考察を行い，最後に 6 章で本研究の結論を述べる．

2　研究背景

本章では，本研究の背景となる従来手法を述べ，その問題点を挙げる．

2.1　従来手法

2.1.1　DVS

モバイルプロセッサには，消費電力を抑えるために Intel の SpeedStep[6] や Geyserville[7]，AMD の PowerNow![8] といった DVS が導入されている．DVS とは，プロセッサの負荷に応じて動的に電源電圧を変化させ消費電力を削減する手法である．このときの電源電圧の変動幅は限られており，プロセッサの標準電源電圧と閾値電圧の間で変化させることができる．プロセッサの消費電力式を以下に示す．

$$P = pCV^2f \tag{1}$$

P：消費電力
p：スイッチングするトランジスタの割合
C：総キャパシタンス
V：電源電圧
f：クロック周波数

式 (1) より分かるように，消費電力はクロック周波数を低下させればそれに比例して低下し，電源電圧を低下させればその 2 乗に比例して低下する．電源電圧を低下させることにより消費電力を削減できる一方で，クロック周波数も低下するため，プログラムの実行速度が低下してしまう．実行時間低下を回避するため，DVS ではプロセッサにかかる負荷が低下した時にのみ電源電圧を低下させる．反対にプロセッサにかかる負荷が上昇すると，プログラム実行速度を上昇させるために電源電圧を上昇させる．このように，DVS はプロセッサにかかる負荷が低下したときにのみ電源電圧を低下さ

せることで，実行時間の増加を抑えつつ消費電力の削減を図る手法である．

2.1.2 PSU

DVS は消費電力を削減する有効な手法であるが，削減できる消費電力が将来的に減少するとの予想から，DVS に代わる省電力化手法として PSU が提案されている．PSU は，動的にパイプラインステージを統合することで消費電力を抑える手法である．

プロセッサの命令を処理するユニットは，複数のパイプラインステージに分割されている．例えば，Intel Pentium4 は高い処理性能を実現するために，パイプラインが 20 段のステージから構成されている．しかし，パイプラインの負荷はキャッシュミスやパイプラインフラッシュの発生により常に一定では無いため，高い処理性能が要求されない場面では高いクロック周波数を維持する必要はない．逆に，プロセッサにかかる負荷が低い場合に高いクロック周波数で CPU を動作させると，無駄に電力を消費してしまう．そこで，PSU はプロセッサにかかる負荷が低い時に，クロック周波数を低下させ，そしてパイプラインステージを統合することで実行時間の増加を抑えつつ消費電力削減を図る．

図 1，**図 2** に PSU に関連する信号線とパイプラインの各ステージ，およびパイプラインレジスタの様子を示す．パイプラインレジスタとはステージ間に存在するレジスタであり，各ステージ間のデータの受け渡しをする．図 1 はパイプラインステージを統合していない状態であり，図 2 はパイプラインステージを統合した状態である．また Stage は各ステージを表しており，実線部分は動作している部分を，破線部分は動作していない部分を表す．

図 1 に示すように，パイプラインレジスタにはクロックドライバからの信号が入力されている．また，PSU を導入したプロセッサにはパイプラインステージの統合を制御するために，新たに信号線 (ステージ統合信号) が追加される．このステージ統合信号は，パイプラインレジスタをバイパスするために組み込んだマルチプレクサの入力として利用される．**図 3** に PSU のパイプラインレジスタ部を示す．これは StageA と StageB の間のパイプラインレジスタであり，図中の Mux はマルチプレクサを示す．このようにして使用回路を切り替えることでパイプラインレジスタのバイパスし，パイプラインステージの統合を実現する．

ステージを統合していない場合，ステージ間のパイプラインレジスタは動作しており，StageA の出力が一度格納される．そのため，StageA と StageB は異なったステージとして動作する．それに対して，ステージを統合している場合，StageA と StageB の間のマルチプレクサへステージ統合信号が入力されるため，パイプラインレジスタは

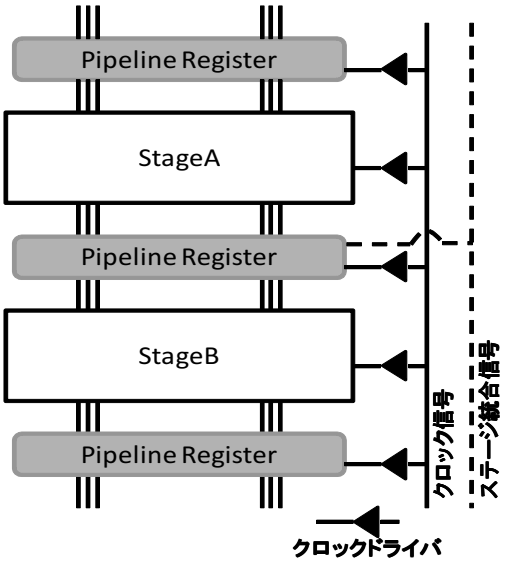

図 1: パイプラインステージ統合を行わない場合の回路

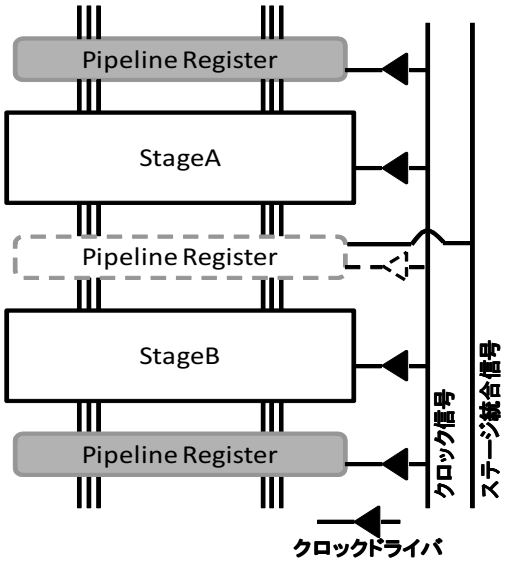

図 2: パイプラインステージ統合を行った場合の回路

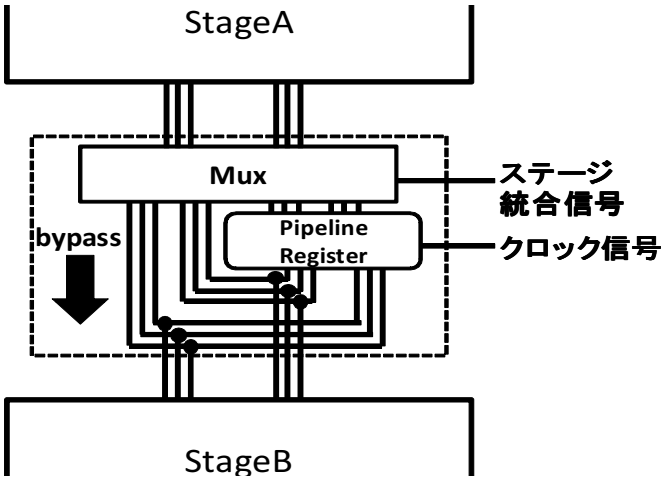

図 3: バイパス回路を持つパイプラインレジスタ

動作しない．パイプラインレジスタをバイパスすることで，図 3 で示すように StageA の出力が直接 StageB の入力として使用されるため，StageA と StageB は 1 つのステージとして動作する．

パイプラインステージを統合することで得られる効果は 2 つある．1 つ目は前述のような特徴を利用して，低負荷時にパイプラインレジスタを停止することで，その使

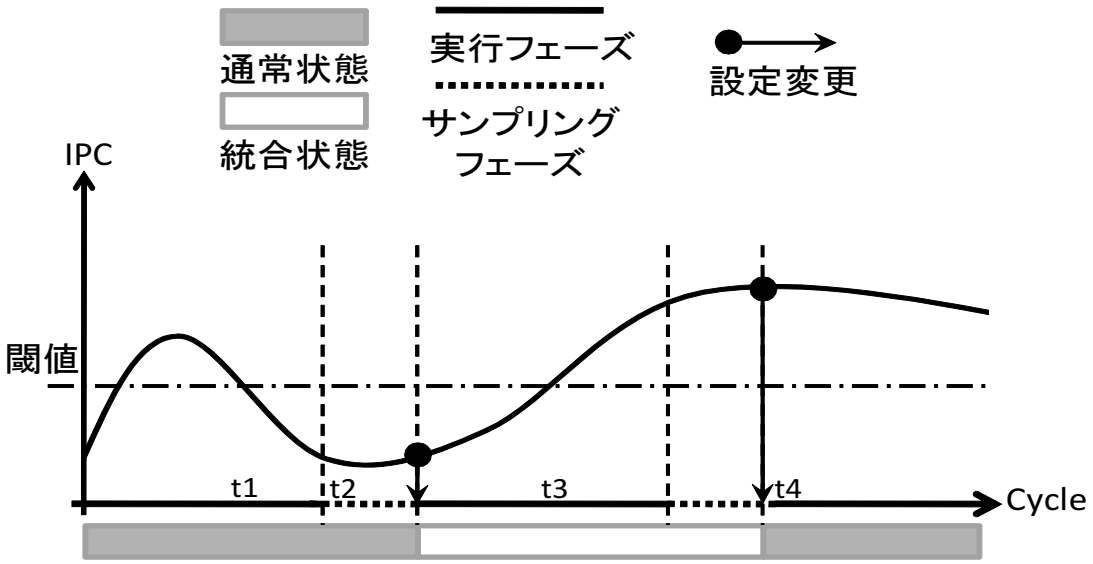

図 4: PSU の制御の概略

用による消費電力を削減できる．2 つ目は，クロック周波数を低下させることに伴い，パイプラインステージ段数を減らすことで命令レイテンシ向上も見込める．

PSU では，前述したパイプライン状態の変更動作を動的に行う．そのために，定期的に IPC をサンプリングする**サンプリングフェーズ**と，プログラムの実行のみを行う**実行フェーズ**をもつ．なお本論文でフェーズとは，一定サイクル数分の期間を指す．サンプリングフェーズでは，命令の実行と並行して IPC の測定を行う．さらに，測定した IPC に応じてクロック周波数とパイプラインステージ数を切り替える．実行フェーズでは，サンプリングフェーズで決定されたパイプライン状態でプログラムを実行する．実行フェーズではクロック周波数やパイプラインステージ数は変更できない．

以降，パイプラインステージがプロセッサの標準段数である状態を**通常状態**，パイプラインステージがプロセッサの標準段数よりも少ない状態を**統合状態**と呼ぶ．PSU の動作を**図 4** に示す．これはプログラムを実行したときの IPC 変化の例を示したものであり，横軸はサイクル，縦軸は IPC を表している．また最下段には，そのサイクル数における状態が通常状態であるか統合状態であるかを記している．サンプリングフェーズで計測した IPC が，図中段に示した閾値を上回った場合，通常状態で実行し，下回った場合，統合状態で実行する．この例ではまず，プロセッサは通常状態でプログラムを実行を開始するが，1 つ目の実行フェーズで IPC が閾値以下に低下してしまったとする (t1)．しかし，まだ実行フェーズ中であるため，この段階で IPC の低下を検知することはできない．ここで，実行フェーズからサンプリングフェーズに切り替わると

(t2)，IPC を測定することでこれが閾値より下回っていることが判明するため，統合状態に移行する．そして，次の実行フェーズ中はパイプライン負荷が低いとプロセッサが予測し，統合状態で実行を続ける．しかし，例ではこの予測とは異なり，2 つ目の実行フェーズでパイプラインの負荷が上昇してしまっているのが分かる (t3)．しかし，まだ実行フェーズなのでプロセッサは IPC の上昇を検知できない．次のサンプリングフェーズに入り IPC の計測をしたところ，閾値よりも上回ったことが判明するため通常状態に切り替えて実行する (t4)．PSU では，このように実行フェーズとサンプリングフェーズを切り替えながら実行を進める．

また DVS と PSU は併用することが可能である．電源電圧を低下させる DVS に，パイプラインステージを統合する PSU を組み合わせることにより，電源電圧の低下による消費電力削減とパイプラインレジスタ部の消費電力削減が期待できる．

2.1.3 負荷情報を用いた PSU

PSU では，サンプリングフェーズでなければパイプライン状態の変更ができなかった．つまり，実行フェーズで負荷の変動が生じたとしても，次のサンプリングフェーズまでパイプライン状態の変更ができない．そのため，消費電力の削減率が抑えられてしまう可能性がある．この問題を解決する手法に，実行フェーズでもパイプライン状態を変更可能にする負荷情報を用いた PSU がある．この手法では，過去のサンプリングフェーズで IPC 上昇を検知したアドレスを記憶し，このアドレスを用いて，実行フェーズでもパイプライン状態を切り替えることを可能にする．このアドレスを負荷情報と呼ぶ．

負荷情報を用いた PSU では，DVS と PSU を組み合わせてプロセッサの電力を削減している．PSU のみの制御と区別するため，以降では電源電圧とクロック周波数を低下させ，さらにパイプライン統合をすることで消費電力を削減している状態を**低速状態**と呼ぶ．

この手法では，過去の負荷情報を記憶しておくため，プロセッサに記憶 buffer を追加する．記憶 buffer は過去のサンプリングフェーズで通常状態に切り替えた命令のアドレスを保持する．**図 5** にこの手法の動作を示す．図 5 の横軸はサイクルを縦軸は IPC を表している．また，横軸の下にそのサイクル数におけるプロセッサの実行状態および現在の PC を示している．例では実行フェーズ中にパイプラインの負荷が上昇し，それに伴い IPC が閾値よりも上回っている (t1)．しかし，PSU と同様にまだ実行フェーズ中なので，この段階で IPC の上昇を検知することはできない．サンプリングフェーズで IPC が閾値より上回ったことを検知し，再び通常状態に切り替わる (t2)．この手

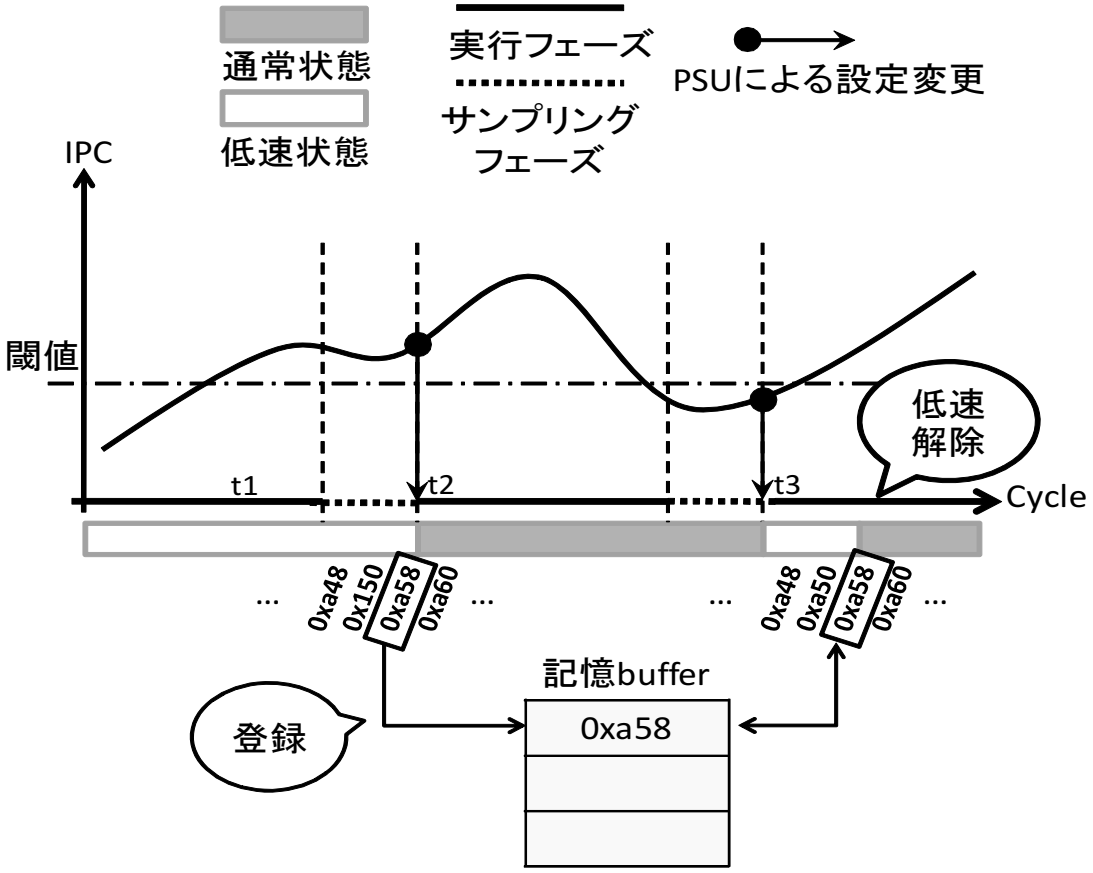

図 5: 負荷情報を用いた動的ステージ統合手法の制御概略

法では，それと同時に通常状態に切り替わった時の PC である 0xa58 を負荷情報として記憶 buffer に登録する．次のサンプリングフェーズでは，IPC が閾値より下回ったことを検知し，低速状態に切り替わる (t3)．その後，3 つ目の実行フェーズ中に再び PC が 0xa58 の値をとると，記憶 buffer に格納されている負荷情報と現在の PC が一致するので，今後負荷が上昇することを予測し強制的に低速状態から通常状態に復帰する．もし記憶 buffer 内の負荷情報と現在の PC が一致した時，すでに通常状態であった場合，プロセッサの実行状態を変更せず引き続き実行を続ける．

この手法が効果を発揮するのは，PSU 制御においてパイプライン負荷の予測が成功しないような場合である．例えば図 5 のように，IPC が閾値を下回ったため，今後パイプラインの負荷が低下すると予測する．しかし予測とは異なり，サンプリングフェーズ直後に負荷が上昇してしまっている．PSU の場合，実行フェーズでプロセッサの実行状態を変更することができないため，少なくとも次のサンプリングフェーズまではプロセッサが低速実行を続けなければならない．そして高負荷時に低速実行すること

は，プロセッサの実行速度を大きく犠牲にする．

2.2 従来手法の問題点

2.2.1 DVS の問題点

DVS では，電源電圧は標準の電圧とトランジスタは閾値電圧の間で変化させる．しかし，将来的にプロセッサの製造技術の進歩により標準の電源電圧は下がっていくと予測される一方で，半導体の微細化が進むと閾値電圧を低下させることができなくなると考えられる．その理由としては主に以下の 2 つが挙げられる．

1 つ目は，リーク電流の増大である．リーク電流とは電子回路上で，絶縁されていて電流が本来流れないはずの場所，経路で漏れ出す電流である．リーク電流が増大する原因は，回路を微細化することで薄くなった絶縁膜をトンネル効果によって電流が通り抜けてしまうことである．これは閾値電圧の低下によっても増大し，リーク電流はゲート電圧が 0.6V 下がるごとに 10 倍となる．この電流はトランジスタオフの状態でも流れ続けるため消費電力の 30%〜40%を占めるプロセッサも存在する．閾値電圧の低下は，デジタル半導体の消費電力低減に寄与してきた一方で，このようにリーク電流の増加につながる問題を抱えている．

2 つ目は，SRAM などにみられる閾値電圧のばらつきである．通常，1 つの SRAM には 6 個以上のトランジスタが使用されている．それぞれのトランジスタの閾値電圧に大きなばらつきがある場合，SRAM へのデータの読み書きが正常に出来なくなる可能性がある．半導体の微細化が進むにつれてこのばらつきは増大する傾向にある．

以上で述べた 2 つの理由から，閾値電圧が下げづらくなり，その反面標準電圧は下がっていく．この変動幅の狭まりから，DVS によって削減できる消費電力は将来的に低下することが予測される．

2.2.2 PSU の問題点

このような DVS の問題から，プロセッサの製造技術に依存しない手法として，PSU が提案されている．しかし，PSU にも消費電力の削減率が抑えられてしまう可能性が存在する．PSU はサンプリングフェーズと実行フェーズが必ず交互に繰り返されている．通常状態，統合状態を切り替えることができるのはサンプリングフェーズのみであり，実行フェーズはパイプライン状態を切り替えることができない．このことから，負荷のかかるタイミングによっては消費電力の削減率が抑えられてしまうことがある．

PSU の消費電力の削減率が抑えられてしまうのは，プロセッサがパイプラインの負荷の予測を誤る場合や，頻繁に IPC が変動する場合である．つまり，PSU では局所的

に負荷の増減が多数発生するプログラムに対応できない．パイプライン状態を負荷の増減に応じて変更できないことで，消費電力や実行時間の増加を招いてしまう可能性がある．

2.2.3 負荷情報を用いた PSU の問題点

前述の PSU の欠点を補うために提案された負荷情報を用いた PSU は，プロセッサの実行状態の変更を実行フェーズでも行えるように改良している．しかし，詳細にプロセッサの状態を制御しているわけではない．また，負荷情報を用いた PSU では状態を変更するための指標として IPC を使用しているが，過去に IPC が上昇した区間を再度実行した場合に再び IPC が上昇するとは限らない．例えば関数を実行し，関数の命令列がキャッシュに存在したため，パイプラインストールが発生せずパイプラインの負荷が上昇したとする．しかし再びその関数に到達した時に，命令キャッシュに当該関数の命令列が存在するとは限らない．このため, この手法も PSU と同様に消費電力の削減率が抑えられてしまう可能性がある．また，記憶 buffer を追加することによる消費電力の増大や，記憶 buffer へのアクセスコストによる実行速度低下も懸念される．

3 提案

本章では，まずプログラム中 IPC 低下に繋がる箇所の特徴を調査し，その情報を用いてプロセッサの実行状態を制御する手法を提案する．

3.1 プログラムの調査

実行速度の低下を抑えつつプロセッサの消費電力をさらに削減するために，通常状態と低速状態の切り替えタイミングを効率化する必要がある．そこで，切り替えタイミングを効率化するために，プログラムのどのような部分でパイプラインの負荷が上昇，低下するのか調査した．

まず，調査の方針を示す．負荷が上昇するのは，パイプライン内に効率良く命令が流れている場合である．逆に負荷が低下するのは何らかの影響によりパイプラインがストールしている場合やパイプラインフラッシュが発生する場合である．パイプライン負荷が低下する原因は様々であるが，現在の計算機で最も大きなペナルティはメモリアクセスである．そこで，プログラム中でメモリアクセスが頻繁に発生する箇所を SPEC CPU95 ベンチマークを用いて調査した．

まず命令キャッシュミスによるメモリアクセスの特徴は，数十サイクル程度の短い間隔で発生する点である．これは，命令キャッシュミスのほとんどが初期参照ミスのため

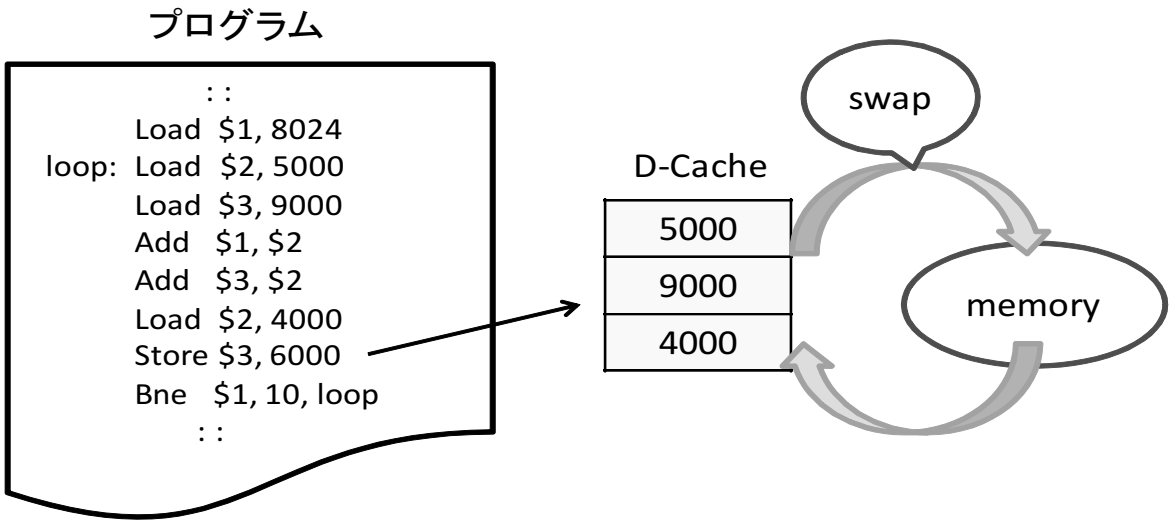

図 6: ループにおけるデータキャッシュミス発生例

である．初期参照である区間の命令は，まだキャッシュ上に乗っていないため，フェッチすると必ずメモリアクセスが発生する．また，プログラムには時間的局所性や空間的局所性が存在し，一度参照された命令は近い将来に参照される可能性が高い．この性質により，しばらく参照されない命令は LRU 等の入れ替え方式でキャッシュから追い出される．そのため例えば，ループ脱出時にキャッシュミスが発生する可能性は高い．

一方，データキャッシュミスによるメモリアクセスの特徴は，ループを実行するにあたり，使用するワーキングセットがデータキャッシュに乗り切らない場合に発生することが多い点である．この場合，ループを脱出するまでデータキャッシュミスが 1 イタレーションごとに発生する．この例を**図 6** に示す．簡略化のため，データキャッシュのサイズは 32Byte × 3 ラインとする．データ転送する Load 命令，Store 命令がループ中に 4 つあり，それぞれ異なるアドレスを使用する．データキャッシュは 3 ラインのみであるので，使用データがデータキャッシュに乗り切らない．そのため，このプログラムの場合，1 イタレーションごとに必ずデータキャッシュミスによるメモリアクセスが発生する．ループの 1 イタレーションのワーキングセットがキャッシュに乗り切らない原因の 1 つは，ループの大きさである．しかし，小規模のループでも連想方式によっては，交互にアクセスされる変数が同一キャッシュラインを共有している場合，頻繁なキャッシュミスが起こる．これは一般的にスラッシングと呼ばれる．スラッシングが発生すると負荷が低下するため，プロセッサを低速状態にし消費電力を削減することが望ましい．

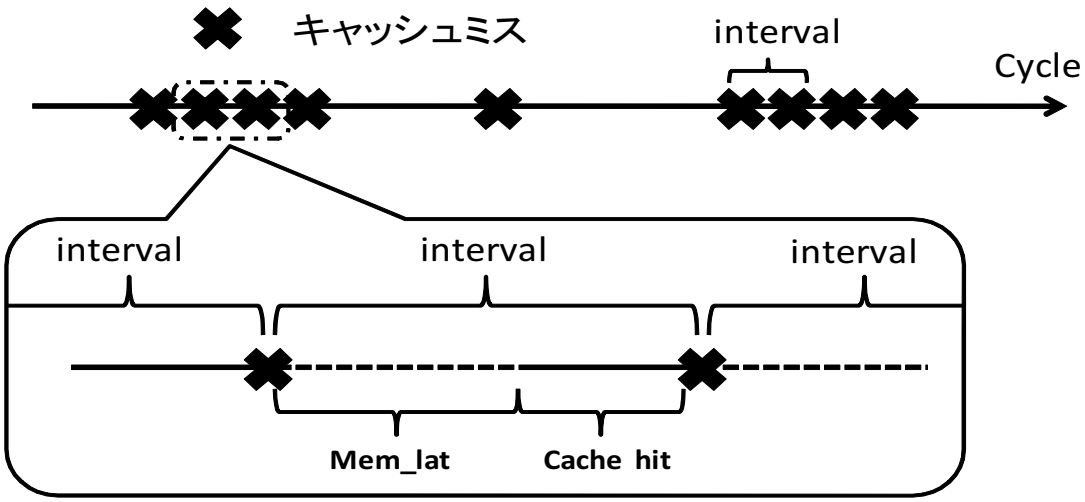

図 7: キャッシュミス発生概略

表 1: 命令キャッシュミス頻発区間における CacheHit

ベンチマーク	最大	最小	平均
129.compress	462cycle	1cycle	25.8cycle
130.li	448cycle	2cycle	14.4cycle

次にキャッシュミスの発生間隔の特徴について述べる．キャッシュミスの発生間隔 (interval) は**図 7** のように，メモリアクセスによるストール時間 (Mem_lat) とキャッシュヒットが続く時間 (CacheHit) で構成されている．Mem_lat はメモリ構成から決定する静的パラメータであるため，interval は CacheHit に依存する．そこで，命令キャッシュミスとデータキャッシュミスの CacheHit の特徴を掴むためにプログラムの調査を行った．

プログラム調査の結果，CacheHit が 500cycle 以上になると近い将来，命令キャッシュミスの発生頻度が低下する傾向が見られた．反対に，CacheHit が 500cycle 以下である区間はキャッシュの発生が固まって存在することを確認した．また，データキャッシュミスは前述のように，比較的ワーキングセットの大きいループの実行中に発生していることが多く，そのため平均 CacheHit が数千サイクルと大きいことを確認した．このことから，命令キャッシュミスの方が発生する間隔が狭いため，低速実行することで省電力化できると考えた．

さらに詳しく調査を行うために，命令キャッシュミスが頻発する期間とデータキャッシュミスが発生しているループ中の CacheHit を調べた．結果を**表 1**，**表 2** に示す． 表 1，表 2 は 129.compress と 130.li で調査したキャッシュミスが頻発する全ての区間にお

表 2: データキャッシュミスが発生するループ区間における CacheHit

ベンチマーク	最大	最小	平均
129.compress	14714cycle	17cycle	3230.4cycle
130.li	27230cycle	2cycle	2698.9cycle

ける最大 CacheHit，最小 CacheHit，平均 CacheHit を表している．表 1 から分かるように，命令キャッシュミスが頻発する区間における平均 CacheHit は各プログラムとも 10 から 25 サイクル程度である．つまり，interval が短くパイプライン負荷が低下する．そのため，この区間に対して低速実行することが望ましい．反対にループ実行中のデータキャッシュミスにおける平均 CacheHit は，表 2 からわかるように平均 2000 サイクルから 3000 サイクルと大きいため，パイプライン負荷が上昇する可能性が高い．そのため通常実行することが良いと考えられる．

以上より，本研究では，命令キャッシュミスによるメモリアクセスに着目し，メモリアクセス発生箇所でのパイプライン状態の制御を行うことで省電力化する手法を提案する．

3.2 命令キャッシュミス監視によるプロセッサ状態の制御

メモリアクセスによるパイプラインストールが発生すると，命令がパイプライン上を流れなくなり，通常状態ではその間の電力が無駄に消費される．そこで，前節の調査結果から命令キャッシュミスによるメモリアクセスに着目し，消費電力を削減する手法を提案する．

具体的な動作を**図 8** を用いて説明する．提案手法もサンプリングフェーズで IPC を計測し，それに応じてプロセッサの実行状態を切り替えるという基本動作は PSU と同様である．しかし，図 8 に示すように提案手法では局所的なパイプライン負荷の変動にも対応する．例えば，2 つ目の実行フェーズで局所的に IPC が低下している．この低下の理由がアドレス 0xa50 の命令から始まる命令キャッシュミス頻発区間だとする．この頻発区間は実行フェーズにおいて現れたため，従来手法ではプロセッサが IPC の低下を検知することはできない．しかし，提案手法ではアドレス 0xa50 で命令キャッシュミスが発生した時点で，今後も命令キャッシュミスが連続して発生することをプロセッサが予測し，一定期間だけ低速状態に切り替える．この期間のことを，**低速実行期間**と呼ぶ．

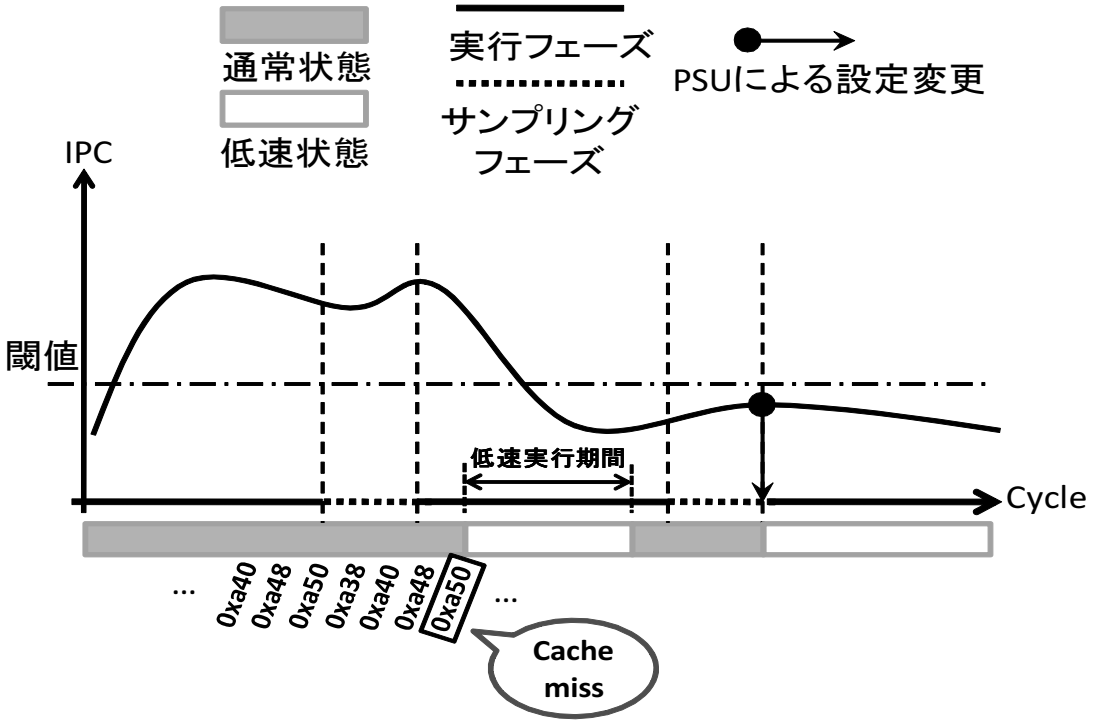

図 8: 提案手法の制御概略

表 3: MissCount の平均値

	$\overline{MissCount}$
129.compress	25.0 回
130.li	27.2 回

本研究では低速実行期間を決定するために以下の式を定義する．

$$S = (\overline{CacheHit} + Mem\_lat) * \overline{MissCount} \tag{2}$$

S：低速実行期間
$\overline{CacheHit}$：CacheHit の平均値
$\overline{MissCount}$：キャッシュミス頻発区間内における命令キャッシュミス回数の平均値

式 (2) に示すように低速実行期間は，平均 interval 長と頻発区間における平均キャッシュミス回数の積と定めた．この式を使用して計算するにあたり，プログラム調査を再度行い，$\overline{MissCount}$ を算出した．調査結果を**表 3** に示す．表 3 に示すように，頻発区間の平均キャッシュミス回数は 129.compress，130.li ともに平均 26 回程度であるこ

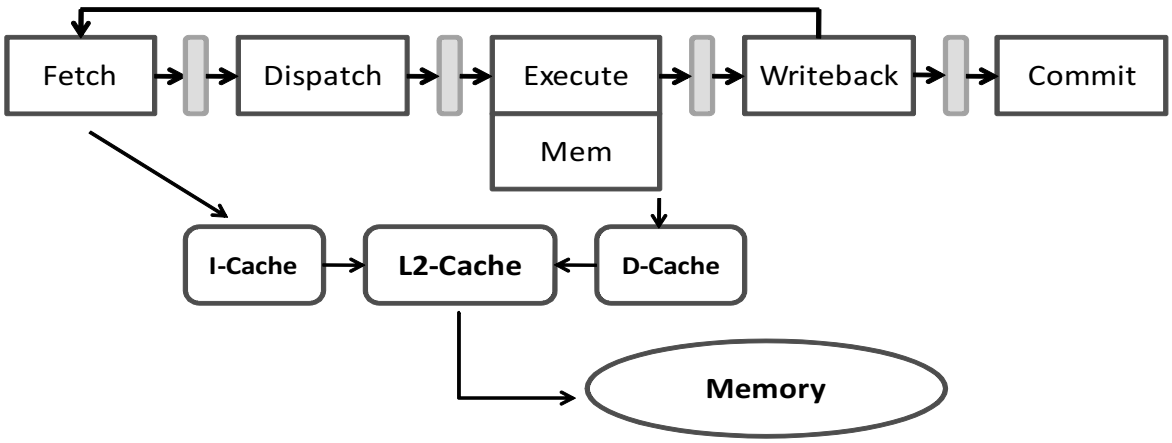

図 9: sim-outorder のパイプライン構成

とが分かった．この結果と Mem_lat を用いて低速実行期間を定めた．

なお，この手法が有効である理由として，メモリアクセスによるストール時間は，プロセッサの実行状態に関わらず一定時間命令がフェッチされないので，低速状態による実行時間の増加を抑えることができる点が挙げられる．本来，PSU による制御ではサンプリングフェーズ内でのみ負荷の計測を行うため，実行フェーズでメモリアクセス等の局所的な IPC の変化に対応することができない．このため，提案手法は PSU の制御のみに比べ実行時間の増加を抑えつつ消費電力の削減を図ることが可能である．

この手法は新たにハードウェアを追加する必要がない．それは, DVS と PSU の制御を命令キャッシュミスが発生した時に行うのみだからである．よって提案手法では，追加ハードウェアによる消費電力の増大はない．

4　実装

命令キャッシュミスによるメモリアクセスを監視することで，消費電力を削減する手法を SimpleScalar シミュレータに実装した．本章では提案する機構の実装について述べる．

4.1　命令パイプラインの仕様

4.1.1　パイプライン構成

DVS と PSU，および本提案手法を，スーパースカラ型プロセッサのシミュレータである SimpleScalar[1] の sim-outorder に実装した．sim-outorder はスーパースカラの特徴である Out-of-Order 実行の機構を備えている．**図 9** にパイプライン構成を示す．

sim-outorder では，図 9 のように 5 つのパイプラインステージをもっている．矢印はデータの流れを示している．また，SimpleScalar は 1 次命令キャッシュと 1 次データキャッシュ，および命令とデータ共通の 2 次キャッシュをもつ．メモリの容量は無制限でありプログラムのアドレスは全てメモリ上に存在している．また，各ステージ間にはパイプラインレジスタが存在する．

各パイプラインステージの役割を以下に述べる．

Fetch ステージ　命令をキャッシュから読み出すステージである．具体的には，パイプライン本数分の命令をフェッチするために I-Cache(Instruction-Cache) にアクセスする．I-Cache に命令が存在しない場合，2 次キャッシュにアクセスする．2 次キャッシュミスした場合，メモリアクセスが発生する．I-Cache は 1cycle でアクセス可能であるが，2 次キャッシュやメモリへのアクセスは 1cycle 以上かかるため，パイプラインはストールする．命令を記憶媒体から Fetch 後，IFQ(Instruction Fetch Queue) に格納する．IFQ は Fetch ステージと Dispatch ステージ間のパイプラインレジスタである．また，分岐予測によりプログラムカウンタを更新し次にフェッチする命令の決定する．

Dispatch ステージ　命令のデコードと issue を行う．IFQ から命令を In-Order で取り出し Reservation Station と Reorder Buffer の機能を持つ RUU(Register Update Unit) に格納する．そして，格納した命令を順次デコードする．その後，RUU に存在する命令のメモリアクセスアドレス，レジスタの依存関係を調べ，依存関係が無ければ readyqueue へ格納する．readyqueue は Dispatch ステージと Execute ステージ間のパイプラインレジスタである．

Execute ステージ　readyqueue の先頭から命令を取り出し，その命令が使用するポートやレジスタ等が確保できた時に，命令を実行する．また，データアクセス命令である場合は D-Cache(Data-Cache) にアクセスする．D-Cache ミスした場合は，2 次キャッシュにアクセスする．2 次キャッシュミスした場合，メモリアクセスが発生する．D-Cache は 1cycle でアクセス可能であるが，2 次キャッシュやメモリへのアクセスは 1cycle 以上かかるため，パイプラインはストールする．また命令を実行した後，実行結果およびメモリから読み出したデータを eventqueue へ格納する．eventqueue は Execute ステージと Writeback ステージ間のパイプラインレジスタである．

Writeback ステージ　Execute ステージで格納されたデータを eventqueue より取り出し，分岐予測の成功，失敗を判定する．分岐予測が失敗と判断された場合，分岐

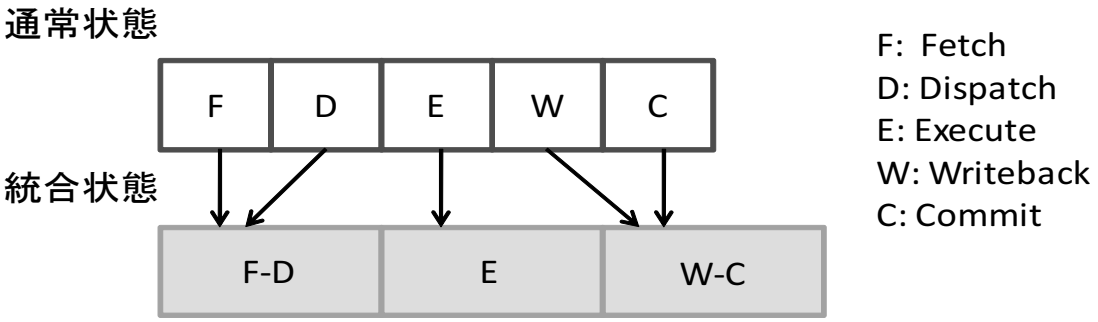

図 10: 通常状態と統合状態

予測ミスをした命令以降に実行された命令を全て破棄し，分岐予測ミス前の状態に戻す．また，分岐予測が正しい場合は実行が終了した命令を RUU に格納する．命令の実行が終了したことによって，issue 可能となった命令を readyqueue に格納する．Out-of-Order 実行により命令実行順序とプログラム記述順序が異なる場合，前の命令の実行が終了するまでこのステージで待機する．

Commit ステージ　RUU の先頭から命令を取り出し，その命令の実行が終わっていた場合，終了処理を行う．終了処理とは，実行結果でレジスタを更新し，ストア命令の場合には，実行結果をメモリに書き込むことである．RUU から命令を取り出し，その命令が writeback イベントを終えていた場合，その命令の終了処理を行う．また，Store 命令の場合，メモリアクセスポートを確保する．

4.1.2　パイプラインのステージ統合手法

sim-outorder は，前節の通り 5 つのパイプラインステージを有しており，これを**図 10** のように 3 つのステージに統合する．統合するステージは Fetch ステージと Dispatch ステージ，Writeback ステージと Commit ステージとする．本研究の実装では Execute ステージは他のステージと統合を行わないが，電源電圧低下に伴いクロック周波数も低下するため，Execute ステージの動作時間も通常状態より長くなる．

パイプラインステージを統合する際，統合するステージ間のパイプラインレジスタを停止する必要があるため，ステージ統合タイミングを調整しなければならない．もしパイプラインレジスタに命令が存在すると，パイプラインレジスタを停止することにより不具合が生じるからである．そのため，本研究では統合するステージの間に存在するパイプラインレジスタに命令が存在しないタイミングで統合する．統合するステージは Fetch ステージと Dispatch ステージ,Writeback ステージと Commit ステージであるため，このステージ間のパイプラインレジスタを監視する必要がある．具体的

には，パイプライン状態を変更する信号が流れた時に IFQ と RUU 内の命令の有無を確認する．命令が存在した場合，命令のフェッチを止め，IFQ と RUU に存在する全ての命令が破棄またはコミットされるのを待つ．そして，命令が存在しないことを検知した時にパイプラインステージを統合する．

4.2 消費エネルギーの算出

本研究ではプロセッサの消費エネルギーを算出するため Wattch[3] を使用する．Wattch は，シミュレータレベルで消費エネルギーを算出するための SimpleScalar に対する拡張であり，パッチとして提供されている．

Wattch は，ALU やキャッシュ等で毎サイクル同じだけのエネルギーを使用することを前提に作られている．そのため，電源電圧やクロック周波数の変化に対応していない．本研究では DVS と PSU を併用して使用するため，Wattch を改良する必要がある．DVS と PSU を使用した場合のエネルギーを算出するため，動的使用エネルギー変更機構を Wattch に追加した．DVS，PSU のエネルギー計算は [10] を参考にした．

4.3 命令キャッシュミス監視によるプロセッサ制御手法

命令キャッシュミスによるメモリアクセスが発生した場合に，低速実行期間だけ低速状態で動作するように PSU に改良を加えた．命令キャッシュミスはキャッシュから命令をフェッチできない場合に発生するため，Fetch ステージを監視すれば良い．Fetch ステージでメモリアクセスを検知した場合，DVS により電源電圧を低下させ，さらに，Fetch ステージからステージ統合信号に命令を送り，パイプラインレジスタを停止させることで消費電力を削減する．そして，低速実行期間経過した後，低速状態から通常状態に復帰させる．

しかし，その低速状態でサンプリングフェーズから実行フェーズに移行した場合，消費電力の削減率が抑えられてしまう可能性がある．その例を**図 11** に示す．図 11 は命令キャッシュミス頻発区間が，サンプリングフェーズ内に存在した場合を示している．図 11 のような場合，(t1) から提案手法によりプロセッサは低速実行期間だけ，低速実行に移行する．しかし，サンプリングフェーズで IPC を計測した結果，閾値を上回っているため PSU の制御により低速実行期間中にも関わらず，プロセッサが通常状態へ移行してしまう．しかし，この期間中は命令キャッシュミスが続く可能性が高いため，通常状態で実行すると消費電力の削減率が抑えられてしまう．そのためこのような場合には，低速実行期間中における PSU の制御を停止させることで対処する．

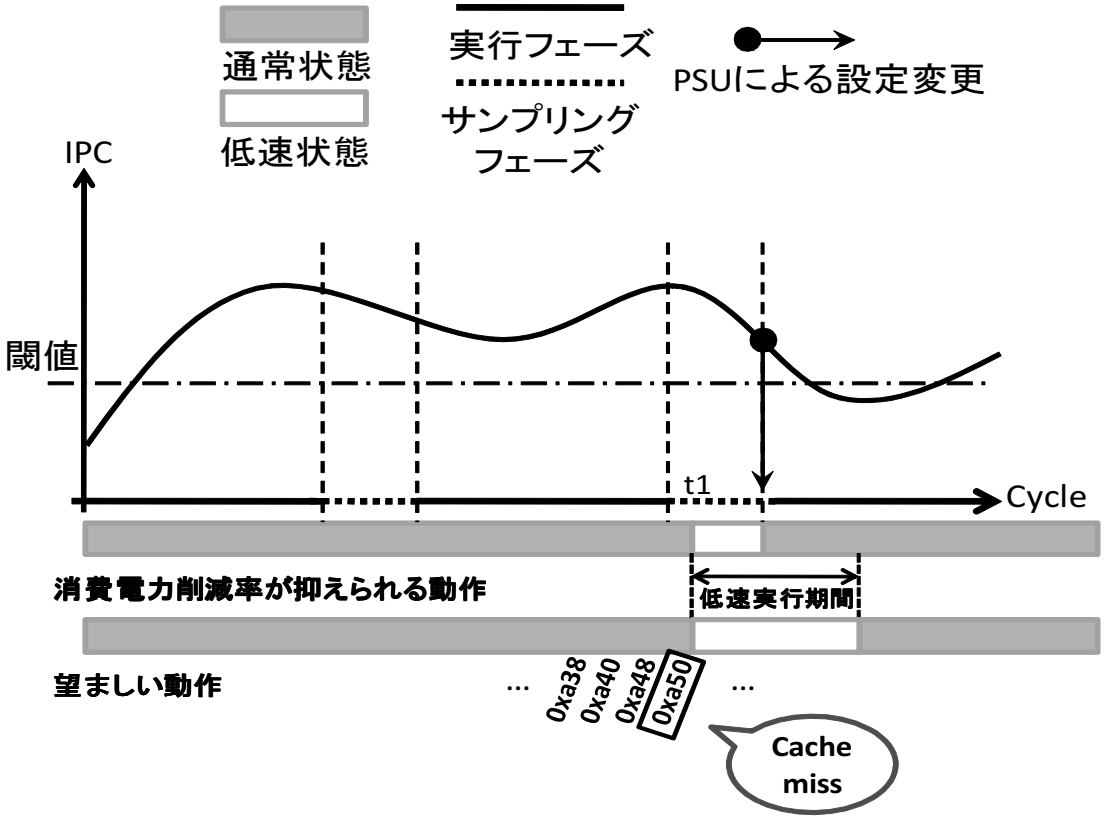

図 11: 提案手法の消費電力削減率が抑えられてしまう場合

5 評価

本章では，提案手法の有効性を示すためにベンチマークを用いて評価し，その結果をもとに考察する．

5.1 評価環境

評価には一般的に用いられる SimpleScalar Toolset 中の Out-of-Order 実行シミュレータを用いた．評価に用いたパラメータを**表 4** に示す．また，**表 5** に通常状態と統合状態における分岐予測ミスペナルティ，キャッシュヒットレイテンシ，パイプライン状態の変更にかかるレイテンシを示す．統合状態におけるこれら各レイテンシは，クロック速度が半分になるという仮定から通常状態の 0.5 倍と設定した．また，PSU の制御に使用する閾値は，プロセッサを終始通常状態で実行した場合の IPC とした．予めプロファイルにより調べた各プログラムの IPC を**表 6** に示す．

本研究ではエネルギー遅延積 (ED 積) と ED^2 積を使用して評価した．ED 積とはプログラムの実行を完了するまでに必要なエネルギーとプログラム実行時間の積である．

表 4: プロセッサ構成

命令セット		PISA
発行命令幅		4 命令
RUU		16 エントリ
LSQ		8 エントリ
メモリポート		2 ポート
機能ユニット	整数	4 個
	整数乗除算	1 個
	浮動小数点	4 個
	浮動小数点乗除算	1 個
TLB	命令	16 エントリ/4way
	データ	32 エントリ/4way
分岐予測	予測手法	gshare
	BTB	512 エントリ/4way
	RAS	8 エントリ
キャッシュ	I-Cache	16KB/32B ライン/1way
	D-Cache	16KB/32B ライン/1way
	L2	256KB/64B ライン/4way
実行フェーズ		500000cycle
サンプリングフェーズ		10000cycle

シミュレート対象のプロセッサが使用したエネルギーを算出するために，Wattch を使用した．本研究では Wattch によりリネームロジック，Load/StoreQueue，BTB(branch target buffer)，命令ウィンドウ，レジスタファイル，キャッシュ，ALU，リザルトバス，クロックで消費されるエネルギーの合計を算出した．

5.2 結果

メモリアクセスコストの増加に伴い，提案手法がどの程度効果が見込めるのか調査するために，メモリアクセスコストを 32cycle，50cycle，100cycle と変化させ評価を行った．評価結果のグラフは，左から順に

(DP) DVS+PSU

表 5: ペナルティと各種レイテンシ

	通常状態	統合状態
分岐予測ミスペナルティ	4cycle	2cycle
L1 キャッシュヒットレイテンシ	1cycle	1cycle
L2 キャッシュヒットレイテンシ	6cycle	3cycle
パイプライン状態変更にかかるレイテンシ	1cycle	1cycle

表 6: 通常状態でのメモリアクセスレイテンシと IPC

ベンチマーク	メモリアクセスレイテンシ	IPC
129.compress	32cycle	1.62
	50cycle	1.6
	100cycle	1.54
130.li	32cycle	1.36
	50cycle	1.36
	100cycle	1.36
132.ijpeg	32cycle	1.87
	50cycle	1.85
	100cycle	1.8
134.perl	32cycle	0.98
	50cycle	0.98
	100cycle	0.97

(M32) 提案手法・メモリアクセス 32cycle
(M50) 提案手法・メモリアクセス 50cycle
(M100) 提案手法・メモリアクセス 100cycle

となる．グラフの各モデルは ED 積と ED^2 積を表しており，それぞれ DVS+PSU(DP) を 1 として正規化した．SPEC CPU95 のベンチマークの結果を**図 12**，**図 13** に示す．図 12 に示すように，全体的にメモリアクセスコストを増加させるにしたがって，ED 積が (DP) に比べ削減できた．その中でも，129.compress，132.ijpeg は顕著に削減できていることが分かる．これらの (M32) と (M100) を比較すると，ともに ED 積削減率が 2 倍となっている．逆に 130.li は，メモリアクセスコストを増加させても提案手

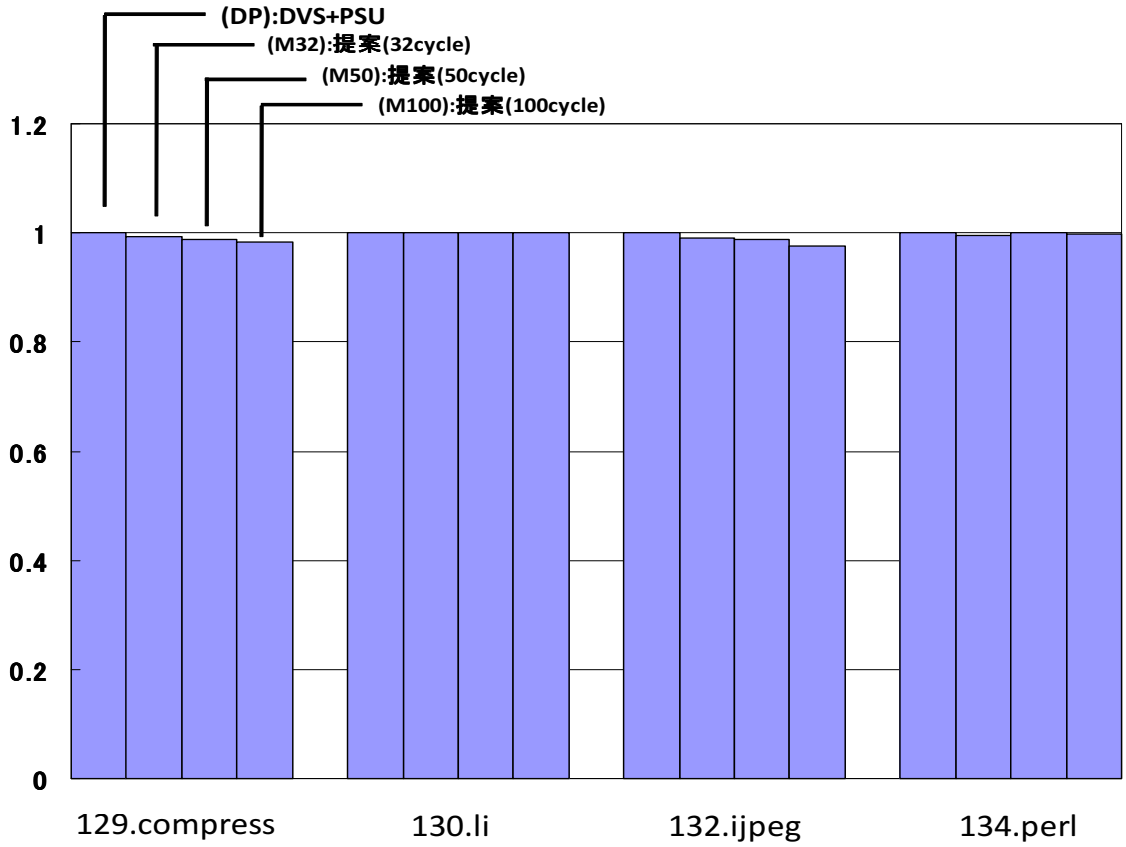

図 12: ED 積による評価結果

法の効果をほとんど得られなかった．129.perl は (M32)，(M50)，(M100) 全てにおいて (DP) より ED 積を削減しているが，(M50) の ED 積削減率が (M32)，(M100) に比べ悪化した．

また，図 13 に示すように 132.ijpeg，134.perl はメモリアクセスコストを増加させるにしたがって，ED^2 積を削減できていることが分かる．しかし，反対に ED 積を削減していた 129.compress の (M32)，(M50) の結果が (DP) に比べ悪化してしまった．130.li は ED 積の時と同様にメモリアクセスコストを増加させても提案手法の効果をほとんど得られなかったため，グラフの変化がほぼ見られない．結果をまとめると，SPEC CPU95 ベンチマークで DVS+PSU(DP) に比べ，ED 積削減率が (M32) で平均で 0.5%，最大で 1.0%，(M50) で平均で 0.6%，最大で 1.3%，(M100) で平均で 1.1%，最大で 2.4%，ED^2 積削減率が (M32) で平均で 0.01%，最大で 0.02%，(M50) で平均で 0%，最大で 0.5%，(M100) で平均 0.06%，最大で 1.5% であることを確認した．

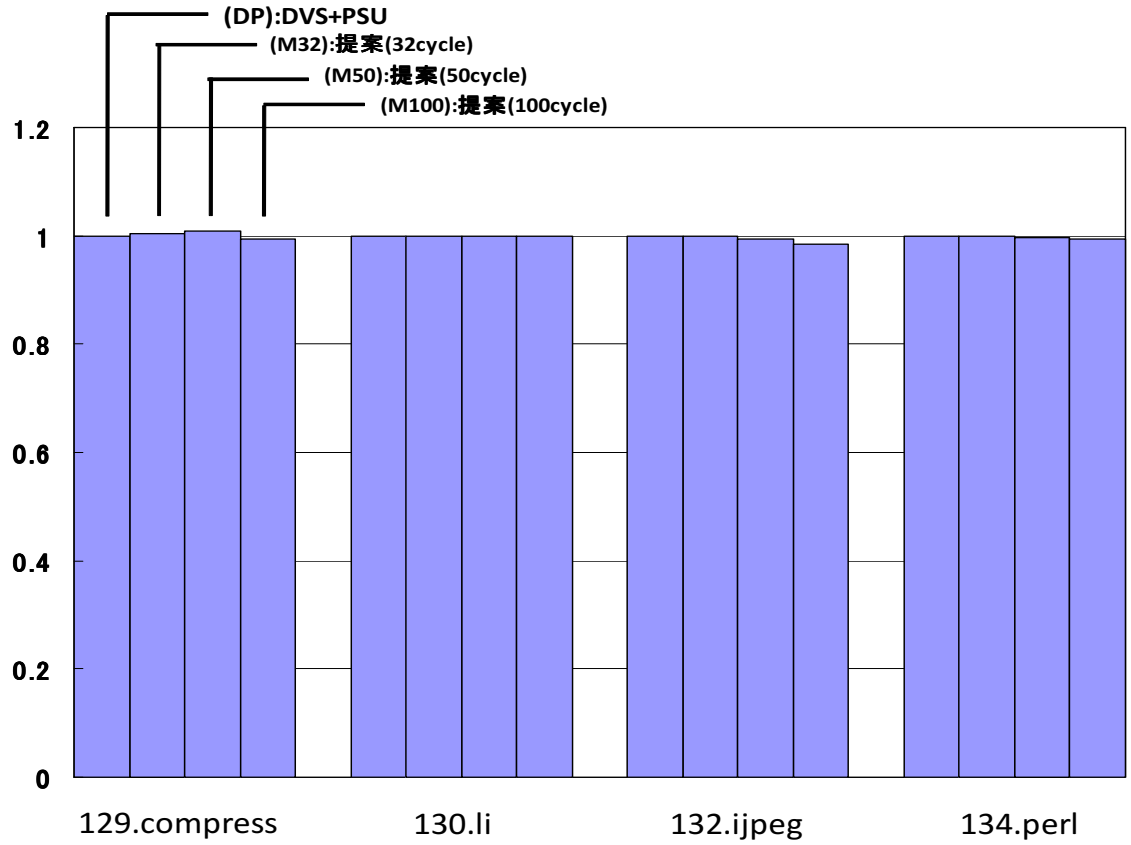

図 13: ED^2 積による評価結果

5.3 考察

ED 積において 129.compress と 132.ijpeg の (M32) と (M100) を比較すると，メモリアクセスコストが 3 倍になる一方，ED 積削減率は 2 倍となった．メモリアクセスコストに比べ ED 積削減率が伸びない原因は，2 つあると考えられる．1 つ目は，プログラムの調査によって決定した低速実行期間が適切でない可能性である．本研究では，129.compress と 130.li のプログラム調査をもとに低速実行期間を設定した．しかし，調査対象のプログラム数が十分でなかったため，低速実行期間が適切でなかった可能性がある．これが，134.perl のように (M50) が (M32) より ED 積が悪化した理由に繋がるのではないかと考えられる．2 つ目は，パイプラインステージ数を変更するコストで提案手法の効果が打ち消された可能性である．プロセッサの状態を変更する信号の出力時に，例えばデータキャッシュミスによるパイプラインストールが発生していた場合，ストール期間中は命令がリタイアされないため，プロセッサの実行状態を変更することができない．よって，パイプラインストールによる命令がリタイアされない

状況は，提案手法に大きく影響する可能性がある．また，130.li，132.perl が提案手法の効果を得にくいのは，全実行サイクル数に対してメモリアクセスにかかるサイクル数の割合がそれぞれ 0.05%，0.7%程度と他のベンチマークプログラムと比べて少ないことが影響していると考えられる．

次に ED^2 積の結果について考察する．129.compress の (M32)，(M50) において ED^2 積の悪化がみられたのは消費エネルギー削減率が実行時間の増加率の 2 乗以下であったためである．反対に，134.perl で ED^2 積が向上したのは提案手法を使用することで，逆に実行が (DP) に比べ高速化したたためである．高速化した理由として，提案手法により低速実行することで，IPC の低下率が通常状態で実行した場合より少なく，本来 (DP) では IPC が閾値を下回るサンプリングフェーズで閾値を上回る場合が存在し，通常状態で実行した期間が増加してしまったためである．

本研究ではメモリアクセスレイテンシを変更して，仮想的ではあるが高クロック周波数で動作するプロセッサモデルの評価を行った．一般に，高クロック周波数で動作するプロセッサは深いパイプラインステージ段数で構成されており，これが増加するにしたがい，パイプラインステージ数を変更するコストが増加すると考えられる．それは，パイプライン上に流れる命令数の増加にしたがい命令間の依存関係が複雑になるため，パイプラインストール期間も長くなるからである．このため，パイプラインステージが深くなるにつれて切り替えコストが増加する．よって提案手法の効果が低下する可能性がある．

6　おわりに

本研究では従来手法である DVS と PSU のさらなる省電力化手法として命令キャッシュミス監視によるプロセッサ状態制御手法を提案した．提案手法の有効性を確認するため，SPEC CPU95 ベンチマークを用いて評価を行った．その結果，従来手法である DVS と PSU を組み合わせた手法と比較して有効性を確認した．

今後の課題として 2 次キャッシュヒットによるパイプラインストールコストを考慮した PSU 制御が挙げられる．メモリアクセスに比べ 2 次キャッシュヒットによるコストは少ないが，実行するプログラムによってはメモリアクセスが存在せず，2 次キャッシュヒットが多発する区間が存在することを確認している．この区間のパイプライン負荷は低下しており，プロセッサの状態を制御することでさらに省電力化できると考えられる．また，データキャッシュミスによるメモリアクセスコストを考慮する PSU 制御も挙げられる．データキャッシュミスが頻発する区間の発生頻度や特徴をさらに

調査することにより，これを考慮した消費電力削減が可能であると考えられる．さらに，命令キャッシュミスやデータキャッシュミスはキャッシュの構成によって発生する箇所やタイミングが変化する．そのためキャッシュの構成を意識し，命令列やアクセスするアドレスからキャッシュミスの発生確率を計算することで，さらなる省電力化が可能であると考えられる．

本研究では，停止するパイプラインレジスタに命令が存在しないタイミングでステージを統合する．しかし，ステージ統合信号が流れたタイミングでパイプラインストールが発生した場合，直ちにステージを統合することができない．このような場合，ストールが解決されるのを待つコストは大きいと考えられる．そのためこのような場合，パイプラインの命令を全て破棄し，ステージ統合の後，破棄した命令を再度フェッチする方が効率良く切り替えを行うことができる可能性があり，今後調査する必要がある．

また，本研究で実装に用いたシミュレータはパイプラインステージ段数が 5 段であった．しかし，現在のプロセッサのパイプラインステージ段数はさらに深いものが多く，今後ステージ段数が深いプロセッサでも評価を行っていく必要がある．

謝辞

本研究のために，多大な御尽力を頂き，御指導を賜わった名古屋工業大学の松尾啓志教授，津邑公暁准教授，齋藤彰一准教授，松井俊浩助教に深く感謝します．また，本研究の際に多くの助言，協力をしていただいた松尾・津邑研究室および齋藤研究室の方々に深く感謝致します．中でも，稲葉崇文氏，加藤拡氏，浅井宏樹氏，池谷友基氏，近藤勝彦氏，安井裕亮氏には本研究に関する多大の助言を頂きました．ここに深く感謝の意を表します．

参考文献

[1] *A User's and Hacker's Guide to the SimpleScalar Architectural Research Tool Set*.

[2] Todd Austin, DanErnst, EricLarson, Chris Weaver : *SimpleScalar Tutorial for release 4.0*

[3] David Brooks, Vivek Tiwari, Margaret Martonosi : *Wattch:A Framework for Architectural-Level Power Analysis and Oprimizations*(2000).

[4] Trevor Pering, Tom Burd, Robert Brodersen : *Simulation and Evaluation of Dynamic Voltage Scaling Algolithms*

[5] Padmanabhan Pillai, Kang G. Shin : *Real-Time Dynamic Voltage Scaling gor*

Low-Power Embedded Operating Systems

[6] *Intel: Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*(2004)

[7] http://www.intel.co.jp/jp/intel/pr/press99/990225b.htm

[8] *AMD: AMD PowerNow! Technology Dynamically Manages Power and Performance*(2000)

[9] 嶋田創, 安藤英樹, 島田俊夫 : パイプラインステージ統合とダイナミックボルテージスケーリングを併用したハイブリッド消費電力削減機構, 先進的計算基板システムシンポジウム (2004).

[10] 嶋田創, 安藤英樹, 島田俊夫 : 低消費電力化のための可変パイプライン, 情報処理学会論文誌計算機アーキテクチャ, No. 145-9 (2001).