修 士 論 文

自発的無効化によるキャッシュメモリの低消費電力化に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

藤田 剛憲

2007年3月

修 士 論 文

自発的無効化によるキャッシュメモリの低消費電力化に関する研究

指導教官　田中清史 助教授

審査委員主査　田中清史 助教授
審査委員　日比野靖 教授
審査委員　井口　寧 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

510087 藤田 剛憲

提出年月: 2007 年 2 月

概 要

近年のマイクロプロセッサにおいて消費電力の増大が問題となっている．また，キャッシュメモリの容量が増大し，現在ではプロセッサ全体の大部分がキャッシュメモリとなっている．さらに，近年では性能と消費電力の両面において有利なチップマルチプロセッサが増えつつある．そこで，本研究ではプロセッサの低消費電力化を達成するために，チップマルチプロセッサを対象としたキャッシュメモリの電力削減手法を提案する．

本研究の基本方針として，キャッシュ中の無効状態のブロックに対して供給電力を遮断することで，プロセッサの性能低下を引き起こさずにキャッシュメモリの消費電力を削減する．また，無効状態のブロックを積極的に増加させる方法として，ソフトウェアによる self-invalidation 手法を検討する．

本研究で得られた結果によると，提案手法により平均 23.4 %の消費電力が削減された．

目 次

第1章　はじめに

1.1　研究の背景

近年，バッテリで駆動するモバイル機器が増えてきた．特に，携帯電話，ノートパソコンに代表される高性能なモバイル機器において，高い処理能力，長い稼動時間が要求されている．また，半導体のプロセス技術は微細化が進み，マイクロプロセッサの動作周波数は飛躍的に向上した．その一方で，微細化により，トランジスタのリーク電流が無視できない程に大きくなっており，マイクロプロセッサの消費電力は増大している．したがって，高性能なプロセッサを搭載したモバイル機器において，要求される処理能力は達成されるが，稼働時間が犠牲になっている場合がある．逆に，動作速度の遅いプロセッサを用いた場合は，要求される稼働時間を満たされるが，処理能力が満たされなくなり，モバイル機器の機能が制限されるかもしれない．

以上から，プロセッサの性能を維持したまま，プロセッサの消費電力を削減することは重要である．

1.2　研究の目的

近年のプロセッサにおいて，キャッシュメモリは，プロセッサの面積の大部分を占有しており，プロセッサで消費される電力の大部分も，キャッシュメモリが消費している．近年，キャッシュメモリの電力削減を目的とした研究が発表されている．

キャッシュメモリの消費電力を削減する手法の一つとして gated-Vdd[1] が提案されている．Gated-Vdd は，キャッシュを構成する SRAM セルと GND の間に高い閾値のトランジスタを設け，このトランジスタによって，供給電力を制御している．キャッシュメモリ使用率が低いとき，gated-Vdd により，キャッシュメモリの一部分に対して，電力供給を行わないようにすることで，キャッシュメモリの電力を削減する．このため，キャッシュメモリの使用率が高くなると，キャッシュミスペナルティが増加する．

このキャッシュミスペナルティの増加を抑制し，キャッシュメモリの電力を削減する手法として，cache decay[2] が提案された．Cache decay はキャッシュアクセスの局所性に注目し，一定時間アクセスされなかったキャッシュブロックは再利用されないブロック (デッドブロック) と予測し，このデッドブロックに対して gated-vdd による電力制御を行う．

また，近年の高性能プロセッサは，単一チップ内に複数のプロセッサコアを有するチップマルチプロセッサ (CMP) 構成が増えてきた．チップマルチプロセッサのメモリシステム

は，キャッシュコヒーレント共有メモリが一般的である．これら共有メモリシステムでは，データの一貫性を保つために，メモリブロックのキャッシュ上のコピーを無効化する要求が頻繁に発生する．この無効化要求によって無効になったキャッシュブロックは，キャッシュメモリから追い出されるまで意味の無いデータを保持しており，無駄な電力を消費している．

本研究では，無効になったキャッシュブロックに対して供給電力を断つことでキャッシュメモリの消費電力を削減する．これにより，キャッシュミスが増えることは無く，性能を維持したまま消費電力を削減することが可能である．

さらに，無効になるキャッシュブロックを早い段階で判断し，無効にすることが出来れば，キャッシュメモリの消費電力をより大きく削減することが可能となる．

無効化要求を受け取ると思われるブロックを予測し，予測された当該キャッシュブロックをあらかじめ無効にすることを self-invalidation という．Self-Invalidation を行う方法として，メモリアクセスの履歴を活用する手法[1] が提案されているが，ハードウェアで履歴を覚えておくため，追加されるハードウェアが比較的大きい．Self-Invalidation を行うことは，キャッシュメモリの消費電力をより大きく削減可能とするが，ハードウェアによる手法では，追加ハードウェアで消費される電力を考慮する必要があり，プロセッサ全体の低消費電力化には適さない．

そこで，ソフトウェアによる self-invalidation 方式を検討する．そのために，メモリアクセス後アクセスブロックに self-invalidation を行う命令 (last-touch 命令) を新たに導入する．実行されるプログラムにおいて，メモリ参照命令によりアクセスされたブロックが，

1. アクセス後必ず無効化要求を受け取る場合，
2. それ以後他の命令によってアクセスされない場合，

そのメモリブロックを最後に参照した命令を last-touch 命令に置換したバイナリコードを作成する．本研究では，このソフトウェアによる self-invalidation を行い，キャッシュメモリの消費電力を削減する．

1.3 論文の構成

本論文は全５章で構成される．

続く２章では，従来のキャッシュ電力削減手法，self-invalidation について説明し，その関連研究を紹介する．３章では，self-invalidation をソフトウェアで行う方法を提案する．４章では，３章で提案した手法に対する評価を行う．５章で，本研究のまとめと今後の課題について述べる．

[1] Dynamic Self-Invalidation (DSI)[3]，Last-Touch Prediction (LTP)[4]

第2章　関連研究

本章では，キャッシュメモリの消費電力削減に関する研究について紹介する．また，マルチプロセッサ環境におけるキャッシュコヒーレンシについて説明する．さらに，本研究において重要となる self-invalidaion について説明する．

2.1　キャッシュの電力削減手法

ここでは，キャッシュメモリの電力削減方法として発表された gated-Vdd[1] と cache decay[2] について説明する．

2.1.1　Gated-Vdd[1]

キャッシュのリーク電流を削減するため，キャッシュを構成する SRAM セルとグランドの間に閾値の高いトランジスタを設け (図 2.1)，このトランジスタのスイッチングにより，キャッシュに対する供給電圧を制御する．

キャッシュを構成する全 SRAM セルを図 2.1 の構成にすると，キャッシュ面積があまりにも増大してしまう．この論文では，各キャッシュブロックにつき 1 つずつ gated-Vdd を設けるのが性能と面積のトレードオフが最も優れているとされている図 (2.2).

この方式は，キャッシュメモリへの電力供給を遮断するため，そこに保持されていたデータは消滅するという性質を持っている．

2.1.2　Cache Decay[2]

キャッシュアクセスには局所性があり，各ブロック毎にアクセスが頻繁に発生する期間 (live time) とラストアクセスからリプレースされるまでの期間 (dead time) が存在する．Cache decay は dead time 中のキャッシュブロックに対する電力供給を，gated-Vdd[1] により遮断することで，電力削減を達成している．

図 2.3 に，あるキャッシュブロックのアクセスパターンと cache decay での電力制御イメージを示す．キャッシュアクセスには時間的局所性が存在する．これを利用し，ある定められた期間 (decay interval) アクセスされなかったキャッシュブロックは，dead time 中であると判断し，リプレースされるまでの間，電源をオフにする．Cache decay を行うた

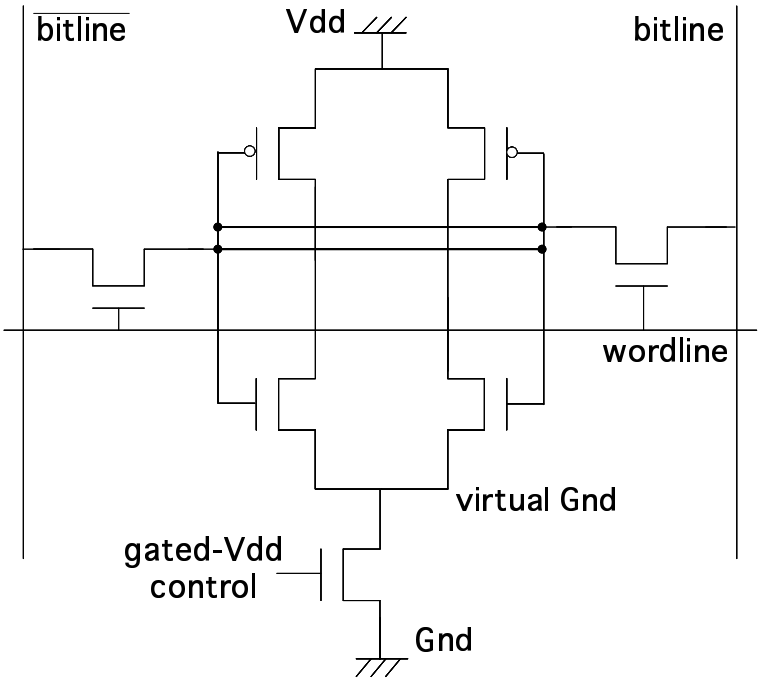

図 2.1: SRAM with an NMOS gated-Vdd

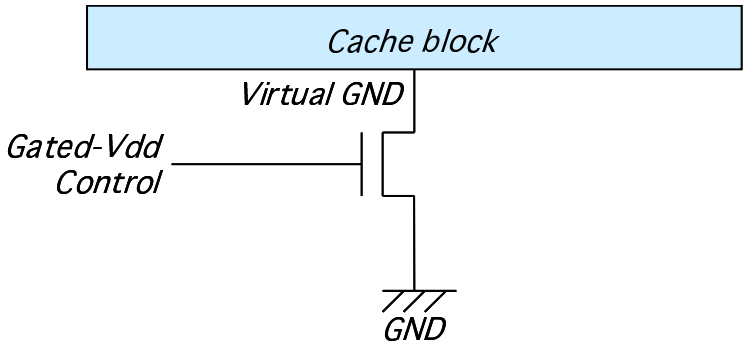

図 2.2: キャッシュブロック単位での電圧制御

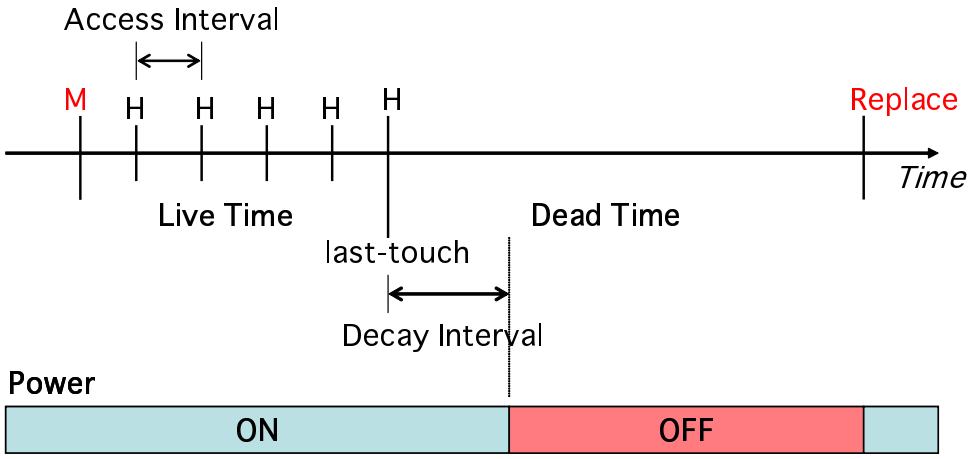

図 2.3: キャッシュブロックへのアクセスパターンと電力制御

めには，decay interval の設定が重要である．Decay interval を同一ブロックへのアクセス間隔 (access interval) より小さくした場合，キャッシュミスが増加し，プロセッサの処理速度が著しく低下する．逆に，dead time より大きく設定すれば，処理速度の低下は起きないが，電力の削減はほとんど期待できない．Decay interval は，access interval より大きく，dead time より小さい値を設定する必要がある．

2.2 キャッシュコヒーレンシ

Self-Invalidation の説明を行う前に，マルチプロセッサにおけるキャッシュコヒーレンシについて触れておく．マルチプロセッサ構成において，共有データのコピーが複数のキャッシュメモリに存在する場合，あるプロセッサがローカルのコピーを更新すると，他のコピーは古いデータを持つことになり，データの不整合が起こり，正しい結果が得られない．これを解決しどのプロセッサも必ず最新のデータにアクセスできるようにする必要がある．

2.2.1 キャッシュコヒーレンシ・プロトコル

キャッシュコヒーレンシを維持するためのプロトコルには，データの書き込みに対処する方法により下記の 2 通りがある．

ライト・インバリデート (write-invalidate) あるプロセッサがデータを書き込む場合，自身のキャッシュ内のデータを更新する前に，他のキャッシュ内の該当データのすべてのコピーを無効化する．書き込みを行うプロセッサは，無効化要求を該当データ

のコピーを持つすべてのキャッシュに送る．無効化要求を受け取ったキャッシュは自分の所に該当データのコピーを保持しているか否かをチェックする．保持している場合は，該当データが含まれるブロックを無効にしなければならない．

ライト・アップデート (write-update) 共有されている各キャッシュブロックを無効化する代わりに，書き込みを行うプロセッサは，該当データのコピーを持つすべてのキャッシュに新データを送り，それを受け取ったキャッシュは各コピーを新しい値で更新する．

2.2.2 キャッシュコヒーレンシの実装

キャッシュコヒーレンシの維持を実現する代表的な方法に，スヌープ方式とディレクトリ方式がある．以下にそれぞれの特徴を示す．

スヌープ方式 すべてのキャッシュ・コントローラが共有バスを監視し，書き込まれるデータ・ブロックのコピーが自身のキャッシュに保持されているか否かチェックし，無効化もしくは新しい値に更新する．共有バスを持たない分散型メモリシステムへの適用は困難である．

ディレクトリ方式 主記憶中に各ブロックの状態を記録したディレクトリが論理的に１つ存在する．ディレクトリに記録される情報は，該当ブロックのコピーを保持しているキャッシュはどれか，そのキャッシュはダーティであるか，などである．ディレクトリのエントリを分散配置することができるため，分散型メモリシステムに適している．

2.3 Self-Invalidation

ライト・インバリデート方式のキャッシュコヒーレンシ・プロトコルにおいて，他のプロセッサからの無効化要求を受け取る前に，該当ブロックを保持するキャッシュ自身の判断によりそのブロックを無効にすることを self-invalidation という．

2.3.1 Dynamic Self-Invalidation(DSI) [3]

ディレクトリ方式でライト・インバリデートプロトコルを実現する環境において，以下の方法で self-invalidation を行う．

- ディレクトリがブロック毎に，バージョンナンバー (VN) と呼ばれる情報を維持する．
- 任意のプロセッサが，あるブロックへの書き込みを行うために排他的アクセスを要求すると，ディレクトリは該当ブロックの VN をインクリメントする．

- キャッシュコントローラはディレクトリと同様にブロック毎に VN を保持する．
- タグは一致したが無効状態であるためにキャッシュミスが発生した場合，キャッシュコントローラは該当ブロックデータの要求と共に，VN をディレクトリに送信する．
- ディレクトリは，キャッシュミスによって要求されたブロックの現在の VN と，受信した VN が異なる場合，要求ブロックのデータに self-invalidation 情報を付加する．
- self-invalidation 情報が付加されたブロックを受け取ったキャッシュは該当ブロックをフィルし，一定時間経過するとそのブロックを無効にする．

DSI 方式は，過去に他のプロセッサにより無効化されたブロックは，再び無効化される可能性が高いという予測を行っている．少なくとも，VN を保持する数ビットがディレクトリとキャッシュタグに必要となる．

2.3.2 Last-Touch Prediction(LTP)[4]

DSI と同様に，過去の情報を基に self-invalidation を行うべきか否かを決定する．DSI と異なりディレクトリ方式以外でも実装することが可能である．LTP は，２レベルのトレースに基づいている．

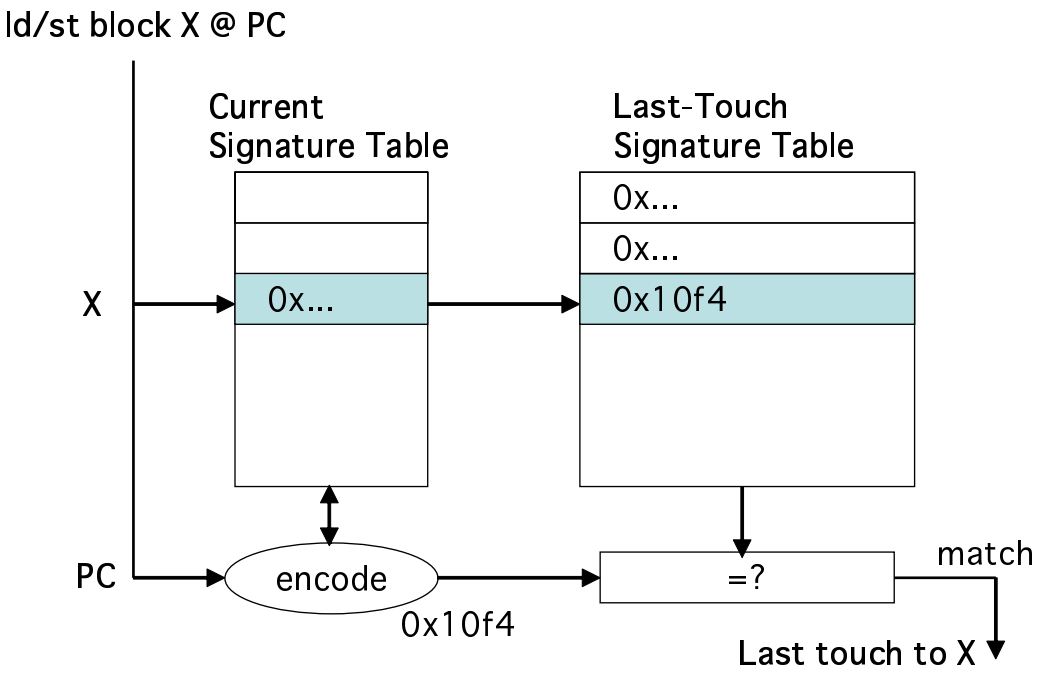

図 2.4: Two-Level trace-based LTP

current signature メモリアクセスの現在の履歴で，ブロック毎に用意された current signature table に保持される．

last-touch signature 以前に観測されたメモリアクセス履歴で，last-touch signature table に保持される．

また，LTP 方式は以下の２つのフェーズにより self-invalidation 予測を行う．

last-touch 学習 ブロック毎にキャッシュミスを起こした命令の PC で current signature を初期化する．ブロックに対する後続メモリアクセスが発生する度に，そのメモリアクセスを行った命令の PC で current signature を更新する．無効化要求が対象ブロックに到達すると，current signature が last-touch signature table に保存される．

last-touch 予測 学習フェーズと同様にキャッシュミスが発生すると current signature を初期化し，後続のメモリ参照命令が current signature を更新する．Last-touch signature table 内の対応エントリと current signature を比較し，一致した場合 self-invalidation を行う．

この方式には２つのテーブルが必要となり，追加されるハードウェアは非常に大きい．

このように，従来の self-invalidation はハードウェアによる方法である．本研究では，ソフトウェアによる方式を検討する．

第3章　提案手法

本章では，キャッシュメモリの消費電力を削減するために本研究で用いる手法について述べる．

3.1　ソフトウェアSelf-Invalidation

ソフトウェア方式のself-invalidationは，プログラム中のメモリ参照命令のうち，メモリブロックの最終アクセスとなる命令を特殊な機能を持った命令に置換し，それを実行することで行う．

3.1.1　ラストタッチメモリ参照命令

ソフトウェアself-invalidationを行うために，まずラストタッチメモリ参照命令を導入する．ラストタッチメモリ参照命令は，対象メモリブロックに対する読み出し／書き込み動作と同時にアクセスしたキャッシュコピーを無効化する機能を持つ．通常，キャッシュのブロックサイズはマルチワードである．しかし，メモリ参照命令によるキャッシュに対する読み出し／書き込みは，ブロックサイズよりも小さいサイズで行う．

例として，ブロックサイズが４ワードの場合を考える．単一のメモリ参照命令がブロック内の４つのワードにそれぞれ一回ずつアクセスした場合，４回目のメモリアクセスが，このブロックにとっての最終アクセス (ラストタッチ) となる．このような場合，メモリアクセスと同時にブロック全体を無効にする命令を使用すると，４回のメモリアクセスすべてで，キャッシュミスが発生する．そこで，無効化機能を２通り用意するために，以下の２つのラストタッチメモリ参照命令を用意することで，マルチワードブロックのキャッシュに対応可能にする．

ラストタッチブロック (ltb ld/st) この命令は，アクセスするワードがキャッシュブロック内のどのワードかに関わらず，読み出し／書き込み動作と同時に該当キャッシュブロックを無効にする．

ラストタッチワード (ltw ld/st) ラストタッチブロックとは異なり，この命令は読み出し／書き込み動作と同時に，キャッシュブロックのどのワードにラストタッチしたかを記憶する．ブロック内の全ワードがこの命令によりアクセスされた場合，ブロックは無効になる．

3.1.2 命令の置換

置換対象となるメモリ参照命令は，その命令によってアクセスしたメモリブロックが，必ず他のプロセッサから無効化要求を受け取る場合と，再び同じプロセッサからアクセスされることが無い命令の２通りである．

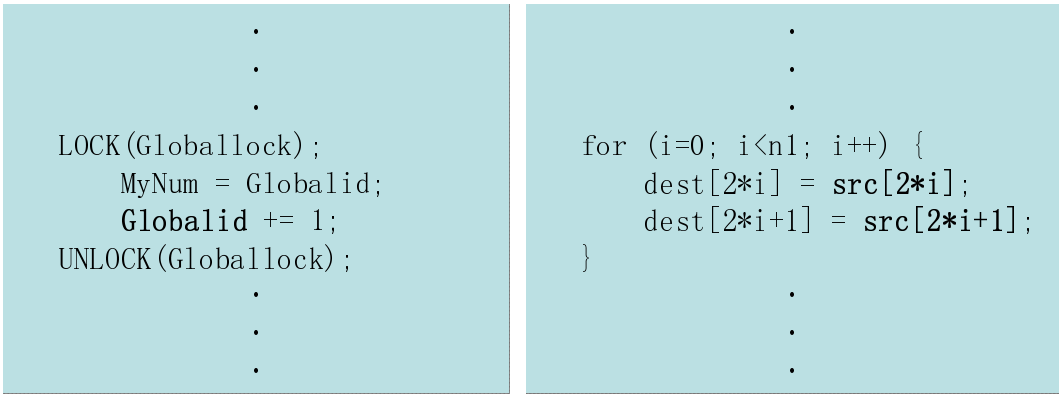

図 3.1: ラストタッチメモリ参照命令適用箇所

図 3.1(a) は，共有変数 Globalid に書き込みが行われた後，他のプロセッサが同じ領域に対して書き込みを行う例である．この場合，共有変数 Globalid へのメモリ書き込み命令を，ラストタッチブロック命令に置換する．

図 3.1(b) では，ループ内の配列 src に対するメモリ参照命令は，数周 (ブロック内のワード数の半分の回数) に１度メモリブロックへの最終アクセスとなる．この場合，ラストタッチワード命令に置換することで，キャッシュミスの増大を発生させずに，self-invalidation を行うことが可能である．

3.2 低消費電力キャッシュ機構

ソフトウェア self-invalidation を用いてキャッシュの消費電力を削減するためには，従来のキャッシュの構造に少量の変更を加える必要がある．以下に新たに付加する機構を説明する．

3.2.1 ラストタッチフラグビット

ラストタッチワード命令によるメモリアクセスが，キャッシュブロック内のどのワードに対して行われたかを記憶するための領域が必要である．これを，ラストタッチフラグ

ビットと呼ぶ．ブロック内の１ワードにつき１ビットを割り当てる．

図 3.2 に本研究で用いるキャッシュの構成を示す．ラストタッチブロック命令によるアクセスが行われたキャッシュブロックの有効ビットはクリアし，無効状態にする．ラストタッチワード命令でアクセスされた場合は，キャッシュタグ中の対応するワード位置のラストタッチフラグがクリアされる．ラストタッチフラグがすべてクリアされると，有効ビットもクリアされ，そのキャッシュブロックは無効状態になる．

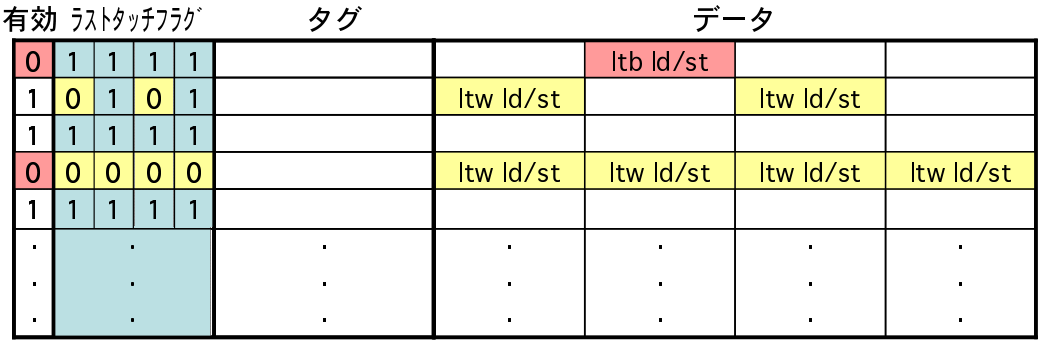

図 3.2: キャッシュの構成

3.2.2 キャッシュ電力削減機構

Gated-Vdd による供給電圧の制御を行い，キャッシュの電力消費を削減する．キャッシュブロック毎に１つの gated-Vdd を付加し，ブロック単位での電圧制御を行う．Gated-Vdd の制御信号として，キャッシュの有効ビットを使用する (図 3.3)．また，消費電力削減の対象は，キャッシュのデータ部のみである．

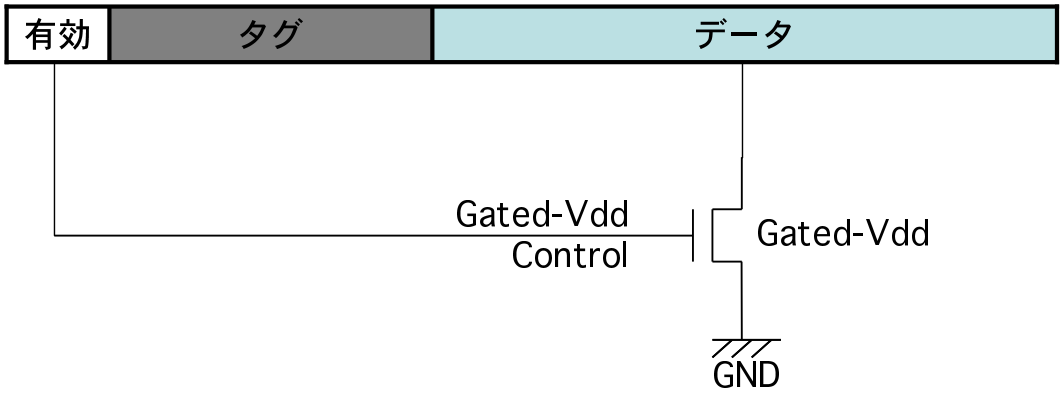

図 3.3: キャッシュ供給電力の制御

第4章　評価

本章では，シミュレーションによる提案手法の評価について述べる．

4.1　評価方法

本研究では，CPU シミュレータを用いて提案手法を評価する．SPLASH-2 ベンチマーク [5] の FFT，LU，RADIX，CHOLESKY の４つプログラムに対して，提案手法を施す．提案手法を組み込む前後でのキャッシュの静的消費電力を比較することで評価を行う．

4.1.1　シミュレータ仕様

シミュレータの主な構成は以下の通りである．

- SPARC V9 命令セットアーキテクチャ[6]
- 2 コアチップマルチプロセッサ
- 32KB L1 命令キャッシュ
- 32KB L1 データキャッシュ

命令，データキャッシュ共に，16Byte ブロック 4 ウェイセットアソシアティブとした．キャッシュの置換アルゴリズムは LRU を用いた．キャッシュミス時の CPU ストール時間は 10 クロックサイクルとした．キャッシュへの書き込みはライト・バック方式である．

ダーティ状態のブロックに対して self-invalidation，もしくは他のプロセッサからの要求により無効化を行った場合，主記憶に対するライト・バックが完了するまでデータを消去することは出来ない．そこで，ライトバッファの存在を仮定する．ダーティ状態のブロックが無効化されると，そのデータはライトバッファに蓄えられる．これにより，主記憶に対するライト・バックが完了する前に無効化を行い，クリーン状態のキャッシュブロックと同様に電力をカットすることができる．以上の理由から，キャッシュにはライトバッファを設け，その容量は十分に大きいものとした．

また，キャッシュコヒーレンシの維持には，スヌープ方式ライト・インバリデートプロトコルを採用した．共有空間への書き込みに伴う無効化要求のやり取りは，1 クロックサイクルで完了するものとする．

4.1.2 キャッシュ消費電力計算

表 4.1: SRAM 1cell 当たりのリークエネルギー

Technique	Active Leakage Energy(nJ)	Standby Leakage Energy(nJ)
no gated-Vdd	1740	N/A
gated-Vdd	1740	53

キャッシュの消費電力の見積もりに，gated-Vdd[1] で紹介されている値を採用した (表 4.1)．プログラム開始から終了までのキャッシュメモリ 1cell の消費電力 E_{leak}[nJ] は次の式になる．

$$E_{leak} = E_{standby} \times \frac{t_s}{f_c} + E_{active} \times \frac{(t_e - t_s)}{f_c}$$

ここで，$E_{standby}$ は Standby Leakage Energy[nJ]，E_{active} は Active Leakage Energy[nJ]，t_s はスタンバイ時間 [cycle]，t_e はプログラム実行時間 [cycle]，f_c はクロック周波数 [Hz] とする．また，スタンバイ状態からアクティブ状態になる際のレイテンシは，キャッシュミスのレイテンシよりも小さいため，特に考慮しない．

本研究では，キャッシュブロック単位での電圧制御を行うため，1 ブロックの消費電力はブロックサイズを B[Byte] とすると，以下のようになる．

$$E_{block} = E_{leak} \times B \times 8$$

これを，各ブロック毎に計算し，集計することでキャッシュ全体の消費電力を見積もる．

4.1.3 SPLASH-2 ベンチマーク [5]

SPLASH-2 ベンチマーク集は，マルチプロセッサ環境の性能を評価するために作られた分散計算プログラム集である．その中から今回使用した SPLASH-2 の各プログラムを以下に示す．

FFT Complex 型データに対して，一次元高速フーリエ変換を行うプログラムである．

LU 行列 A を下三角行列 L と上三角行列 U の積 (A=LU) に分解するプログラムである．密行列の LU 分解 (contiguous blocks) と疎行列の LU 分解 (noncontiguous blocks) の二種類がある．

RADIX 整数値に対して，基数ソートを行うプログラムである．

CHOLESKY 正定値対称行列 A を対角成分がすべて正であるような下三角行列 L とこの行列 L の共役転置 L*との積 (A=LL*) に分解するプログラムである．LU 分解の特殊な形である．

シミュレーションに使用した SPLASH-2 の各プログラムの入力パラメータは CPU シミュレータに合わせて，プロセッサ数 2，キャッシュサイズ 32KByte を指定した．また，並列計算の分割サイズはキャッシュブロックサイズ 16Byte に合わせた．各プログラムの入力データを表 4.2 に示す．CHOLESKY のみ入力ファイルを指定する．入力ファイルは，SPLASH-2 ベンチマークに標準で入ってるものから選択した．

表 4.2: 各プログラムの入力データサイズ

	入力データサイズ
FFT	65536 complex
LU cont	256×256 matrix
LU noncont	256×256 matrix
RADIX	262144 keys
CHOLESKY	wr10.O ファイル

4.1.4 置換命令決定アルゴリズム

本研究では，メモリアクセスのトレース情報を基に，ラストタッチメモリ参照命令に置換する命令を決定する．以下にそのアルゴリズムを示す．

1. 共有領域へのアクセスが行われた場合，そのワードアドレスを最後に発行した命令 (ラストタッチ命令) の PC を記憶する．これを，アドレスベーストレースと呼ぶ．
2. メモリ参照命令毎 (PC 毎) に，その命令が発行した全ワードアドレス情報とワード毎にアクセスした回数を記憶する．これを，PC ベーストレースと呼ぶ．
3. 無効化要求を受け取った場合，そのブロック内の全ワードアドレスに対する現在までのトレース収集は一度終了する．その後再び同じアドレスに対してアクセスされた場合は別のアドレスとして解釈する．
4. アドレスベーストレースに記憶された PC を命令置換候補とする．
5. 命令置換候補をキーとして，PC ベーストレースを探索し，当該 PC が発行した全アドレスに対して 1 回ずつしかアクセスしてない場合，その命令は置換対象命令となる．
6. 置換対象命令が参照する全アドレスをキーとして，アドレスベーストレースを再び参照し，キーアドレスのラストタッチ命令が置換対象命令の PC と一致しないものがあれば，置換対象命令から外す．

7. 置換対象命令のうち，その命令が発行したアドレスが各ブロックにつき１回のみであれば，ラストタッチブロック命令に置換し，その他の場合はラストタッチワード命令に置換する．

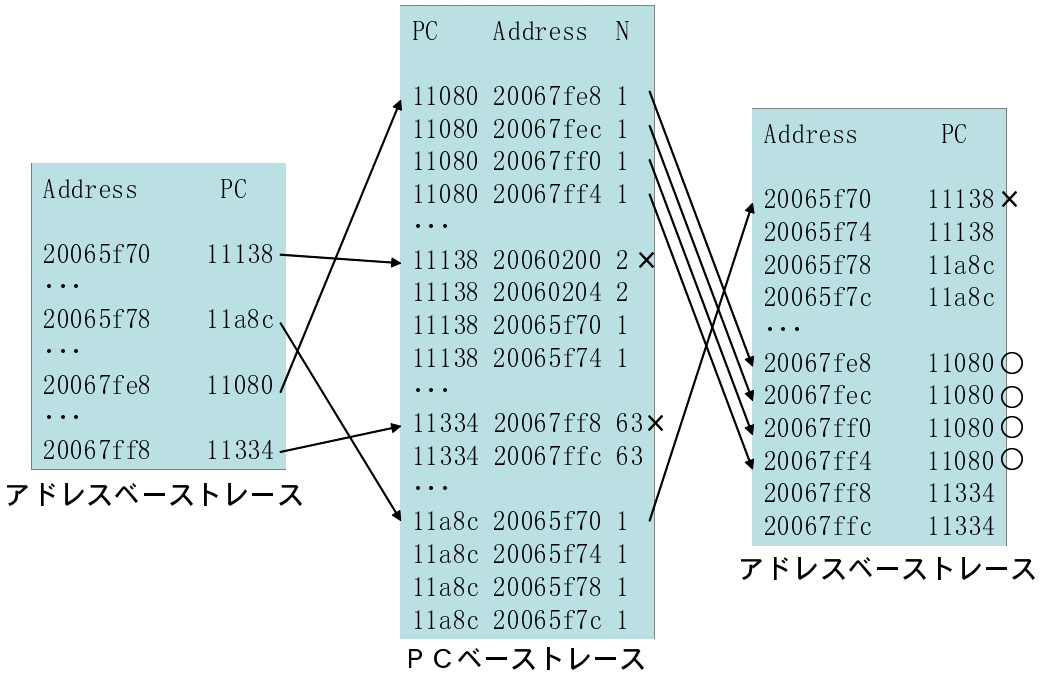

図 4.1: 置換命令決定の例

図 4.1 は，置換命令探索の例である．アドレスベーストレース中の Address 項目は発行されたアドレス，PC はそのアドレスを最後に発行した命令である．また，PC ベーストレース中の N は，それぞれのアドレスに対するアクセス回数である．

まず，命令置換候補として，PC11138 を選ぶ．この値をキーとして，PC ベースアドレスを探索すると，アドレス 20060200 に対して 2 回アクセスを行う．したがって，この命令は置換対象とならない．

次に，命令置換候補として PC11a8c を選択する．この値で PC ベーストレースを探索すると，全ての発行アドレスに対して 1 回ずつアクセスしている．したがって，この命令は置換対象命令となる．しかし，PC11a8c の命令が発行するアドレス 20065f70 をキーとして，再びアドレスベーストレースを参照すると，違う PC の値が書かれているため，この時点でこの命令は置換対象命令から外される．

次に，PC11080 の命令を見ていく．PC ベーストレースにおいて本命例は，全ての発行アドレスに対して 1 回のみアクセスしており，さらにアドレスベーストレースにおいて，その発行アドレスのラストタッチ命令は同一 PC である．この命令は置換対象命令となる．この PC が発行したアドレスから，各ブロックにつき 2 回以上アクセスすることが分かる．

したがって，この命令はラストタッチワード命令に置換する．

最後に，PC11334 の命令を見る．この命令は発行した 1 つのアドレスに対して，63 回アクセスするため，この命令を置換することは出来ない．

4.2 結果

シミュレーションの結果を示す．まず，無効状態のキャッシュブロックの電力を制御することの効果を示し，次に，ソフトウェア self-invalidation の効果を示す．そして，最後にプログラムを通常実行する場合と，提案手法を用いて実行する場合の消費電力の比較を行う．

4.2.1 Gated-Vdd による電力削減

ソフトウェア self-invalidation の性能を見る前に，gated-Vdd の制御信号に有効ビットを使用することの評価を行う．

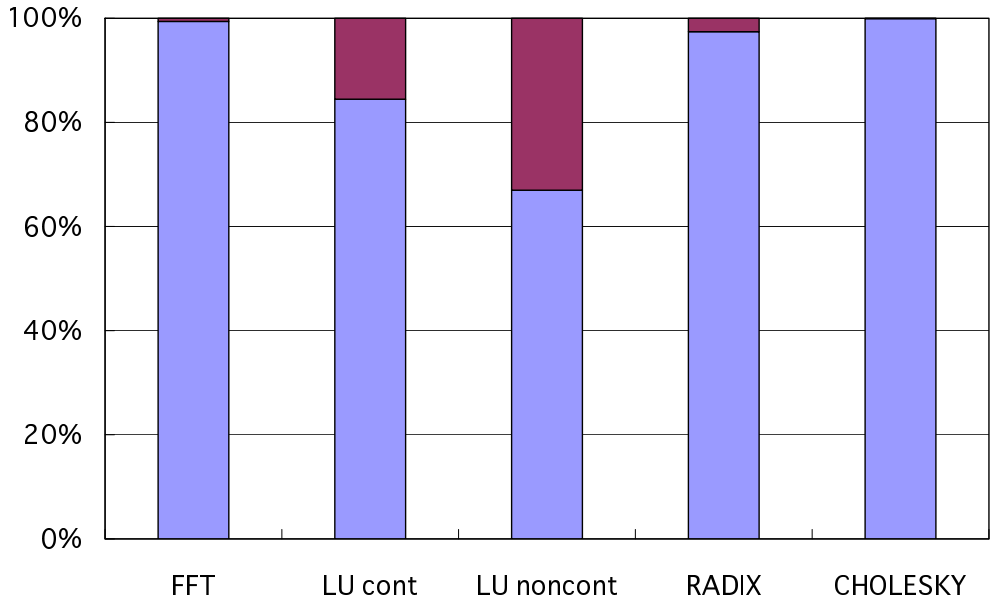

図 4.2: 有効ビットによる Gated-Vdd 制御

図 4.2 は，各プログラムにおいて Gated-Vdd による電力制御を行わない場合 (通常実行時) と，有効ビットを gated-Vdd 制御に使用した場合 (gated-Vdd 実行時) のキャッシュメモリの消費電力の関係を割合で表示したものである．通常実行時のキャッシュ消費電力を

100 %とし，それに対して gated-Vdd 実行時の消費電力が何%であるかで示している．図 4.2 から LU cont で約 15 %，LU noncont で約 33 %，RADIX で約 2.5 %の電力を削減することができている．FFT，CHOLESKY では電力削減はほとんど出来ていないことが分かる．

表 4.3: キャッシュ無効化回数

	インバリデート回数
FFT	39
LU cont	130720
LU noncont	232019
RADIX	812
CHOLESKY	106

表 4.3 は，それぞれのプログラム実行で，無効化要求により実際に無効化が行われた回数である．図 4.2 において，電力削減率が大きかった LU noncont，LU cont，RADIX の順に，無効化回数が多いことが分かる．Gated-Vdd の制御に有効ビットを用いているため，この関係が成り立つのは自然である．

4.2.2 ソフトウェア Self-Invalidation の評価

ここでは，ラストタッチメモリ参照命令を用いたソフトウェア self-invalidation 単体の性能を見る．

図 4.3 は，各プログラム別のソフトウェア self-invalidation を行わず gated-Vdd による電力制御のみを行った時 (gated-Vdd 実行時) と，ソフトウェア self-invalidation を行い，かつ gated-Vdd による電力制御も行った時 (ソフトウェア self-invalidation 実行時) のキャッシュメモリの消費電力を比較したものである．Gated-Vdd 実行時のキャッシュ消費電力をを 100 %とし，それに対してソフトウェア self-invalidation 実行時の消費電力が何%であるかで示している．FFT で約 2.0 %，LU cont 約 6.0 %，LU noncont 約 20 %，RADIX 約 45 %，CHOLESKY 約 1.0 %の電力を削減することができた．

表 4.4 に，self-invalidation が実行された回数を示す．また，表の右端の列はラストタッチワード命令が実行された総数である．おおよそラストタッチワード命令 4 回に 1 回 self-invalidation が行われる．図 4.3 と表 4.4 から，self-invalidation が多く行われたプログラムほど，電力削減率が大きいことが分かる．

表 4.5 は，ソフトウェア self-invalidation を行った場合と，行わなかった場合の各プログラムの実行サイクル数である．どのプログラムも，ソフトウェア self-invalidation を実行することで，実行サイクル数が小さくなっていることが分かる．

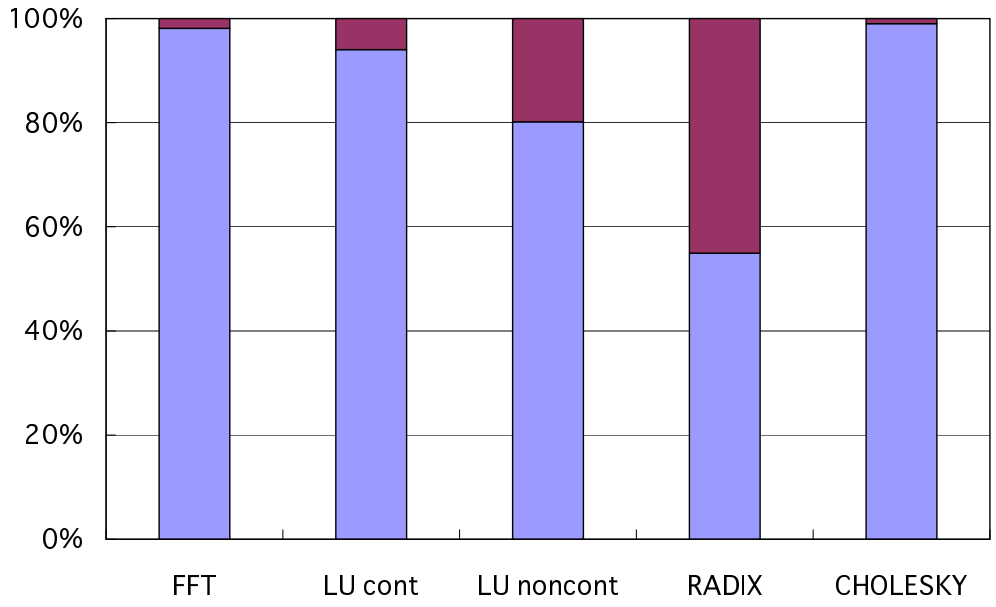

図 4.3: ソフトウェア Self-Invalidation による電力削減率

表 4.4: Self-Invalidation 回数

	Self-Invalidation 回数		ラストタッチワード命令実行数
	ラストタッチブロック	ラストタッチワード	
FFT	36	66044	264190
LU cont	18	82150	396571
LU noncont	18	157156	908555
RADIX	25	272420	1179675
CHOLESKY	9	14839	71630

表 4.5: 実行時間の差

	通常実行 実行サイクル数	ソフトウェア Self-Invalidation 実行サイクル数
FFT	118641649	118641584
LU cont	100084316	100084040
LU noncont	98001681	97693117
RADIX	122541690	122418679
CHOLESKY	39266223	39258305

表 4.6: 提案手法によるキャッシュミス数の変化

	通常実行	提案手法	減少率 (%)
FFT	920109	920098	0.0012
LU cont	577608	577489	0.021
LU noncont	2077916	2015857	3.0
RADIX	523157	498436	4.7
CHOLESKY	482915	481376	0.32

表 4.6 に，各プログラムを通常実行した場合と提案手法で実行した場合のキャッシュミス数を示す．すべてのプログラムで，キャッシュミス数が減少した．LRU は，長いアクセス間隔を持つブロックをリプレース対象として選択する．そのため，まだ live time 中のブロックをリプレース対象として選択する可能性がある．それに対して，self-invalidation を行った場合は，確実に dead time 中のブロックをリプレース対象として選択することが可能となり，キャッシュミスが減少する．以上が実行速度が向上した要因である．

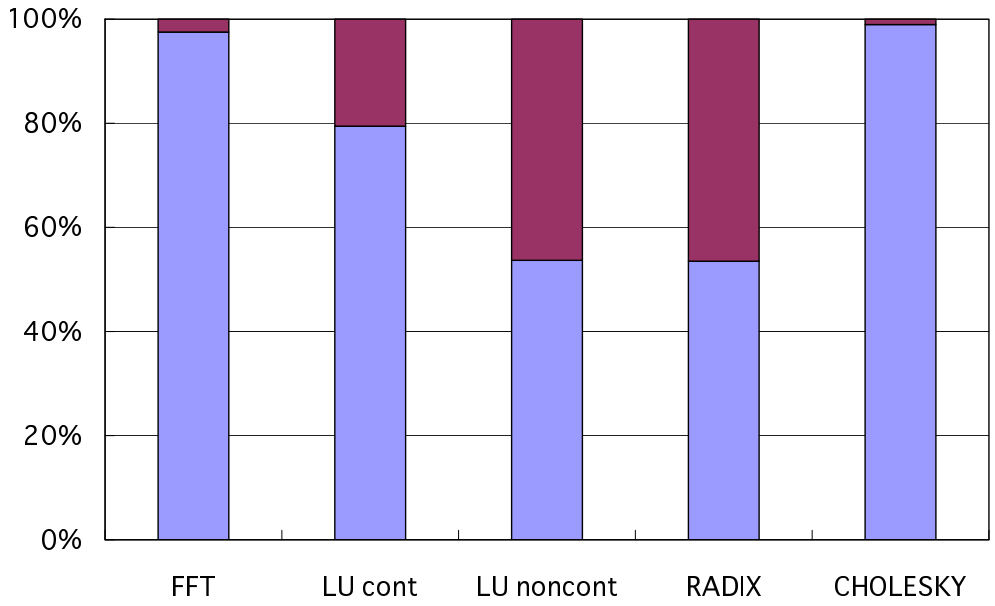

図 4.4: 提案手法の電力削減率

図 4.4 は，各プログラムを通常実行した場合のキャッシュの消費電力と，ソフトウェア self-invalidation 実行した場合のキャッシュ消費電力の比較である．FFT で約 2.5 %，LU cont で約 20 %の消費電力を削減することができた．有効ビットを使用した gated-Vdd による電力削減が良好だった LU noncont では約 46 %，ソフトウェア self-invaidation が最も多く行われる RADIX は約 46 %の電力削減を達成することができた．CHOLESKY は他のプログラムに比べて最も電力削減が難しく約 1.0 %しか電力削減することが出来なかった．プログラムによって結果には大きくバラつきが出たものの，全てのプログラムにおいて消費電力を削減することができた．

図 4.5 は，通常実行時，gated-Vdd 実行時，ソフトウェア self-invalidation 実行時におけるキャッシュの消費電力の関係である．通常実行時の消費電力を 100 %として表示している．各プログラムにおけるそれぞれの実行状態の消費電力の関係がわかる．表 4.7 には，各プログラムにおける通常実行時に対するキャッシュ消費電力削減率とその平均値をまとめる．

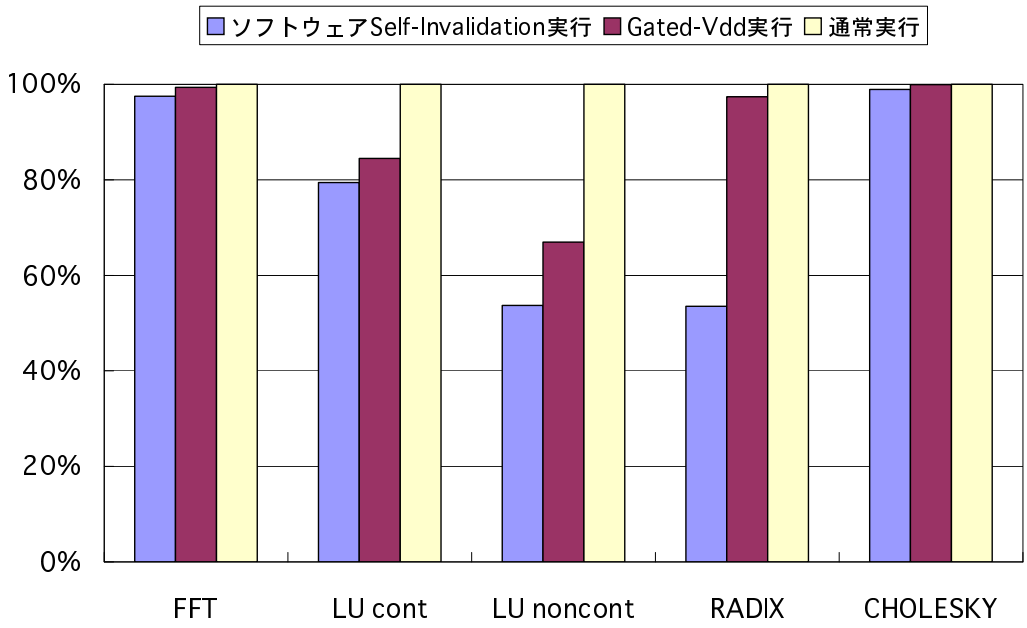

図 4.5: ３つの実行状態のキャッシュ消費電力

表 4.7: 実行状態ごとの電力削減率 (%)

	Gated-Vdd 実行	ソフトウェア SI 実行
FFT	0.6	2.5
LU cont	15.5	20.6
LU noncont	33.0	46.3
RADIX	2.6	46.5
CHOLESKY	0.08	1.0
average	10.4	23.4

第5章　まとめ

最後に本研究におけるまとめを行い，今後の課題について述べる．

5.1　まとめ

本論文では，チップマルチプロセッサに焦点を当て，プロセッサの消費電力の削減を行った．プロセッサの大部分を占めるキャッシュメモリをターゲットとし，キャッシュメモリの低消費電力化のために，キャッシュの有効／無効状態を gated-Vdd 制御に使用する方式を提案した．また，これを積極的に行うために，ソフトウェアによる self-invalidation 手法を提案した．

SPLASH-2 ベンチマーク集の中から FFT，LU，RADIX，CHOLESKY に対して，提案手法がどの程度キャッシュの消費電力を削減することができるか評価を行った．ベンチマークプログラムに対して提案手法を適用する際，そのプログラムのトレース情報を使用した．評価の結果，キャッシュの消費電力を平均 23.4 %削減することができた．また，キャッシュのリプレースの際，優先的に self-invalidation を行ったブロックがリプレース対象として選ばれることで，キャッシュミスが減少し，プログラム実行時間が短縮された．

5.2　今後の課題

今後の課題として以下の点を挙げる．

1. 提案手法を適用した実行バイナリを生成するコンパイラの開発
2. シングルプロセッサ環境での提案手法の適用とその効果の検証
3. キャッシュタグ追加情報のハードウェア量
4. プロセッサ全体の消費電力

1 に関して，今回はトレース情報を基に置換可能なメモリ参照命令を探索することで提案手法を適用した．コンパイル時にソースコードを解析することで，提案手法を適用できる場所を発見することができれば，プログラマが作成したどのようなプログラムに対しても提案手法を適用することができる．

2 について，本研究で提案したソフトウェア self-invalidation はマルチプロセッサ環境だけでなく，シングルプロセッサ環境にも応用可能である．その場合，cache decay[2] に代表される他のキャッシュ低消費電力化法との比較，組み合わせなどを検証することで，最適なキャッシュの消費電力削減法が発見できる．

3 について，今回は各キャッシュブロックのタグ情報に数ビットのラストタッチフラグを設けた．このハードウェア量を測定し，DSI[3]，LTP[4] のハードウェア量と比較・検証する必要がある．

4 に関して，今回は L1 データキャッシュメモリに対する電力評価しか行っていない．提案手法を適用した場合のプロセッサ全体の消費電力について評価する必要がある．

謝辞

本研究を遂行するにあたり，終始熱心に御指導して下さった田中清史助教授に心から深く感謝するとともに，ここに御礼申し上げます.

適切な御意見，御助言を頂いた日比野靖教授，井口寧助教授に深く感謝致します.

その他，貴重な御意見，討論を頂きました田中研究室の皆様をはじめとする多くの方に対して厚く御礼申し上げます.

最後に，日頃から暖かく見守って下さった両親，仲間，友人達に深く感謝致します.

参考文献

[1] Michael Powell, Se-Hyun Yang, Bebak Falsafi, Kaushik Roy, and T.N.Vijaykumar. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. In the Proceedings of the 2000 International Symposium on Low Power Electronics and Design, pp. 90–95, 2000.

[2] Stefanos Kaxiras and Zhigang Hu, Margaret Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In the Proceedings of the 28th Annual International Symposium on Computer Architecture, pp. 240–251, 2001.

[3] Alvin R. Lebeck and David A. Wood. Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors. In the Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 48–59, 1995.

[4] An-Chow Lai and Babak Falsafi. Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. In the Proceedings of the 27th Annual International Symposium on Computer Architecture, pp. 139–148, 2000.

[5] Stevan Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In the Proceedings of the 22nd Annual International Symposium on Computer Architecturethe, pp. 24–36, 1995.

[6] SPARC International, Inc. The SPARC Architecture Mannual Version 9. July, 2003.