Vol.2019-ARC-237 No.6
2019/7/24

計算再利用を用いたプロセッサ省電力化の検討

武石 隆太郎[1]　津邑 公暁[1]　中島 康彦[2]

概要：ゲート遅延に対する配線遅延の相対的な増大から，集積回路の微細化による高クロック化だけではマイクロプロセッサの性能向上を実現することは困難になってきた．こうした中で，SIMD やスーパースカラなどの命令間の並列性に着目した高速化手法が注目されるようになった．しかし，プログラム中の並列性を持つ部分全てに並列化を適用するのは困難なため，これらの手法よる性能向上にも限界がある．並列化による高速化が注目される一方で，我々は独自性の高い計算再利用に基づく高速化手法を採用した自動メモ化プロセッサを提案している．計算再利用とは処理の実行自体を省略することで高速化を図る手法であるが，実行の省略により，実行に必要なプロセッサユニットでの動的消費電力が低減するため，消費電力削減にも効果がある可能性がある．そこで本稿では，計算再利用が持つ，プロセッサ省電力化に対するポテンシャルを検証するため，自動メモ化プロセッサの詳細な電力評価を行った．評価の結果，一部のベンチマークにおいてはプログラムの実行に必要な消費エネルギーが減少することを確認した．

1. はじめに

これまで，ムーアの法則およびデナード・スケーリングに基づき，微細化によりプロセッサの消費電力の削減およびクロック周波数の向上が実現されてきた．しかし，CMOS トランジスタの小型化が進むにつれリーク電流が増加し，今後のプロセッサの性能向上には消費電力の改善が必要である．それに加え，現在はスマートフォンやタブレット端末などのモバイル機器が広く普及している．これらのモバイル機器では高い性能と，より長い駆動時間との両立が求められており，消費エネルギーを抑えつつ高いパフォーマンスを達成することが課題となっている．このような背景から，プロセッサの性能向上を図りつつ電力効率を改善するため，半導体レベルだけでなく，ロジックレベル，アーキテクチャレベルなど，様々なレイヤーで省電力化の研究がなされている．

これまで我々は，アーキテクチャレベルでのプロセッサの高速化および省電力化の研究を行ってきた．なかでも，我々はプロセッサの高速化として独自性の高い，計算再利用を用いた**自動メモ化プロセッサ（Auto-Memoization Processor）**[1] を提案している．計算再利用とは再利用対象である命令区間の実行自体を省略することで高速化を図る手法である．自動メモ化プロセッサは，関数およびループを再利用対象の命令区間とし，過去に実行した命令区間を同一の入力により再実行する際に，記憶した出力を再利用することで，その命令区間の実行を省略する．実行自体の省略により，実行ユニットで消費される動的電力が低減するため，このアイディアはプロセッサの消費電力削減にも効果がある可能性がある．

そこで本稿では，自動メモ化プロセッサの詳細な電力評価を行い，計算再利用が持つ，プロセッサ省電力化に対するポテンシャルについて検証する．その上で，より電力効率を向上させるために，メモ化専用ユニットの消費エネルギー削減の方法についても検討する．

2. 自動メモ化プロセッサ

本章では，自動メモ化プロセッサの動作原理とその構成について概説する．

2.1 自動メモ化プロセッサの概要

自動メモ化プロセッサは，ハードウェア上で動的にメモ化を適用可能とするための専用ユニットを備えたプロセッサである．自動メモ化プロセッサは，通常のプロセッサが行う，演算ユニット，デコーダ，レジスタ等を用いた実行処理を，メモ化専用ユニットを用いた，命令区間の過去の出力を読み出す処理で代替する．なお，**メモ化（Memoization）**[2] とは計算再利用を命令区間に適用することを言い，元来，高速化のためのプログラミングテクニックである．通常，メモ化を適用するためには，プログラムを記述しなおす必要があり，既存のロードモジュールやバイナリをそのまま高速化することはできない．その上，ソフトウェアに

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

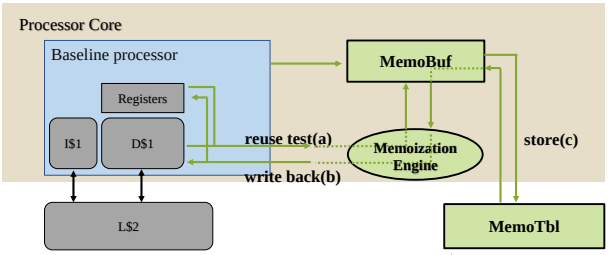

図 1　自動メモ化プロセッサのハードウェア構成

よるメモ化 [3] はオーバヘッドが大きく，限られたプログラムでしか性能向上が得られない．これに対し自動メモ化プロセッサは，ハードウェアを用いて動的に関数やループを検出しメモ化を適用できるため，既存のバイナリを変更する必要がない上，入力の一致比較や過去出力の再利用をハードウェアによって支援するため，オーバヘッドが小さく，より多様なプログラムを高速化出来る．

自動メモ化プロセッサのハードウェア構成の概略を図 1 に示す．自動メモ化プロセッサは，一般的なプロセッサが持つ構造に加えて，独自の機構として，命令区間およびその入力と出力の組（入出力セット）を記憶しておく表である**再利用表（MemoTbl）**，**メモ化制御機構（Memoization engine）**および MemoTbl への書き込みバッファとして働く**再利用バッファ（MemoBuf）**を持つ．MemoTbl はサイズが大きく，コア外に存在し，コアからのアクセスコストが大きいため，入出力を検出する度に MemoTbl へアクセスするとオーバヘッドが大きくなる．そこで，このオーバヘッドを軽減するために，一時的に入出力を記憶する，小さな作業用バッファである MemoBuf をコアの内部に設けている．なお，自動メモ化プロセッサでは，レジスタおよびメモリ参照を「入力」，レジスタおよびメモリへの書き込み，さらに計算再利用の対象命令区間が関数の場合はそれに加え返り値を「出力」として扱う．

自動メモ化プロセッサは再利用対象の命令区間の開始を検出すると，MemoTbl を参照し現在の入力セットと MemoTbl に記憶されている過去の入力セットとを比較する．これを**再利用テスト（reuse test）**（図 1(a)）と呼ぶ．もし，現在の入力セットが MemoTbl に記憶されたいずれかの入力セットと一致する場合，メモ化制御機構はその入力セットに対応する出力セットをレジスタやキャッシュに書き戻し（write back）（図 1(b)），命令区間の実行を省略する．一方，現在の入力セットが MemoTbl のいずれの入力セットとも一致しない場合，自動メモ化プロセッサはその命令区間を通常実行しながら，その入出力を MemoBuf に蓄えていき，実行終了時に MemoBuf の内容を MemoTbl に登録（store）（図 1(c)）することで将来の再利用に備える．

一般に命令区間内では，複数の入力値が順に参照され使用される．しかし，同じ命令区間でも，その入力アドレスの列が異なる場合がある．例えば，条件分岐命令を実行すると，次に参照されるアドレスはその条件分岐命令の分岐結果によって変化する．そこで，自動メモ化プロセッサは，全入力パターンを木構造で表現し，MemoTbl に登録する．例えば，自動メモ化プロセッサが図 2 に示すプログラムを実行する場合，関数 calc の全入力セットは図 3 に示すような木構造で表現されることになる．なお，図 3 中のノードは命令区間の入力値を，エッジは入力値と次に参照される入力値との対応関係を示しており，End はそれ以上の入力値が存在しないことを示す．また，図 3 の (i)，(ii)，(iii) はそれぞれ図 2 の 8 行目，9 行目，10 行目における関数呼び出しの際の入力セットに対応する．この例において，入力セット (i)，(ii) では，変数 b が 3 番目に参照されるのに対して，入力セット (iii) では，変数 c が 3 番目に参照される．これは，2 番目に参照される変数 a の値が異なることにより，プログラムの 5 行目における条件分岐の結果が変化するためである．

次に，この木構造で表現された全入力パターンを登録するための表である MemoTbl の構成を図 4 に示す．MemoTbl は，命令区間を記憶する**命令区間表（FLTbl）**，入

```
1  int a = 3, b = 4, c = 8;
2  int calc(x){
3    int tmp = x + a + 1;
4    if(tmp < 7) return tmp + b;
5    else        return tmp + c;
6  }
7  int main(void){
8    calc(2); /* x = 2, a = 3, b = 4 */
9    b = 5; calc(2); /* x = 2, a = 3, b = 5 */
10   a = 5; calc(2); /* x = 2, a = 5, c = 8 */
11   a = 3; calc(2); /* x = 2, a = 3, b = 5 */
12   return(0);
13 }
```

図 2　サンプルプログラム

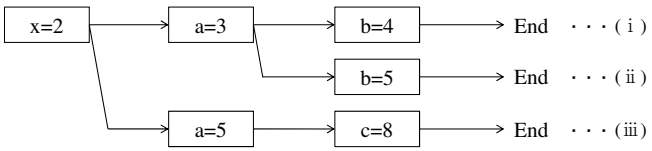

図 3　入力パターンの木構造

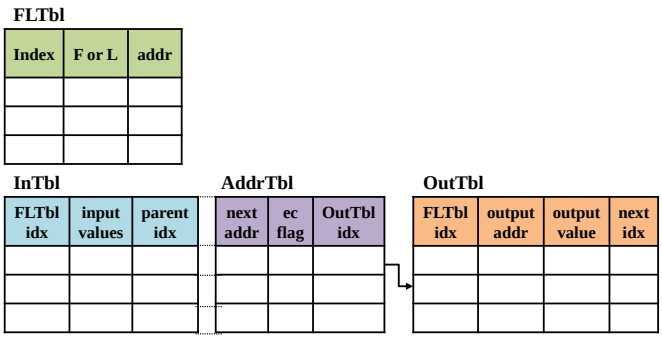

図 4　MemoTbl の構成

力を記憶する**入力表（InTbl）**，入力アドレスを記憶する**アドレス表（AddrTbl）**，および出力を記憶する**出力表（OutTbl）**の 4 つの表から構成される．FLTbl，AddrTbl, OutTbl は RAM で, InTbl は **3 値 CAM（Ternary Content Addressable Memory）**で実装する．

FLTbl は 1 エントリが 1 命令区間に対応しており，その行番号（Index）を各命令区間の識別番号とする．また，関数とループを判別するフラグ（F or L）と，命令区間の開始アドレス（addr）を持つ．

InTbl の各エントリは FLTbl の行番号 Index に対応するインデクス（FLTbl idx）を持ち，この値を用いてどの命令区間の入力値を記憶しているかを判別する．また，命令区間の全入力パターンを木構造で管理するために，入力値（input values）に加えて親エントリのインデクス（parent idx）を持つ．この input values はキャッシュラインに含まれる全ての入力値を記憶する．この際，キャッシュライン中で入力として参照されず，比較する必要のないアドレスの入力値についてはドントケアで表現される．なお，レジスタの値が入力値となる場合も同様に，複数レジスタの入力値をまとめて 1 つの入力エントリに記憶する．

AddrTbl は InTbl と同数のエントリを持ち，各エントリは 1 対 1 に対応する．AddrTbl の各エントリは入力値検索のために，次に参照すべきアドレス（next addr）を持つ．また，入力セットの終端エントリか否かを保持するフラグ（ec flag）を持ち，そのエントリが終端エントリである場合，出力を記憶している表である OutTbl のエントリを指すインデクス（OutTbl idx）も持つ．

OutTbl の各エントリは FLTbl のインデクス（FLTbl idx）に加えて，命令区間の出力先のアドレス（output addr）および出力値（output values）を持つ．また，出力セットの各エントリをリスト構造で管理するため，次に参照すべきエントリのインデクス（next idx）を持つ．

2.2 自動メモ化プロセッサの動作

本節では，自動メモ化プロセッサの動作について説明する．特に，再利用テストに成功した場合と失敗した場合，それぞれにおけるパイプラインの動作について具体的に説明する．

2.2.1 再利用テスト成功時

再利用テスト成功時のパイプライン処理の様子を**図 5** に示す．なお，以降の説明を簡略化するために，自動メモ化プロセッサのパイプラインのウェイ数は 2 とし，各パイプラインは Fe（フェッチ），De（デコード），Ex（命令実行），Re（リタイア）の 4 段構成をとり，各ステージは 1 サイクルで処理されると仮定する．また，キャッシュミスなどによりパイプラインがストールすることなく，理想的な状態で各ステージでの処理が実行されるものとする．この図において，`callq` は関数呼び出しのための命令，`retq` は関数復帰のための命令であり，`pushq` から `retq` までが関数内部の命令である．なお，関数の先頭命令をフェッチしてから，関数の復帰命令をリタイアするまでに要するサイクル数が，その関数の実行に要するサイクル数である．自動メモ化プロセッサは関数呼び出しのための命令を検出すると，計算再利用を適用できるかどうかを確かめるために再利用テストを行う．この再利用テストを正しく行うためには，関数の入力となる値がレジスタやキャッシュに存在している必要があるため，プログラムオーダ上で関数呼び出し命令よりも前の命令によるレジスタやキャッシュへの書き込みは全て完了している必要がある．よって再利用テストは，関数呼び出し命令がリタイアされるタイミング（図の例では 4 サイクル目）で行う．なお，再利用テストを行っている間も，関数内の命令を再利用テストとオーバラップして引き続き実行させることによって，再利用テスト失敗時のペナルティを緩和する．ただし，再利用テスト中は，リタイアステージでの処理を禁止している．これは，関数内の命令がリタイアされてしまうと，一致比較対象の入力値が上書きされる可能性があるからである．

図の例ではその後，6 サイクル目において，この再利用テストに成功し，計算再利用を適用できることが判明したとすると，メモ化機構は過去の実行結果を MemoTbl からレジスタやキャッシュに書き戻すと同時に，パイプラインをフラッシュする．これは，既にパイプラインに投入されている命令は再利用対象の命令区間である関数内の命令であり，実行する必要がないためである．

その後，7 サイクル目にて，関数復帰先の命令（`leaq`）をフェッチし，関数復帰後の実行を継続する．図 5 に示す例の場合では，再利用テストの成功により，関数の実行を省略したことによって，1 サイクル目から 6 サイクル目までの関数内部の命令実行は全て無駄となってしまうが，関数の実行に要したのは 6 サイクルのみであり，メモ化を行わない通常実行時と比べて，4 サイクル削減される．

2.2.2 再利用テスト失敗時

次に，再利用テスト失敗時のパイプライン処理の様子を**図 6** に示す．6 サイクル目において，この再利用テストに失敗し，この関数に計算再利用を適用できないことが判明すると，禁止していたリタイアステージでの処理が再開され，通常通り命令が実行される．この図の例の場合，関数の実行には 12 サイクルを要しており，メモ化を行わない通常実行時と比べて，2 サイクル増加している．

3. 消費エネルギー評価環境の設計

本章では，自動メモ化プロセッサの消費エネルギーの評価方法について概説する．

3.1 エネルギー評価の方針

自動メモ化プロセッサの消費エネルギーを計測するため

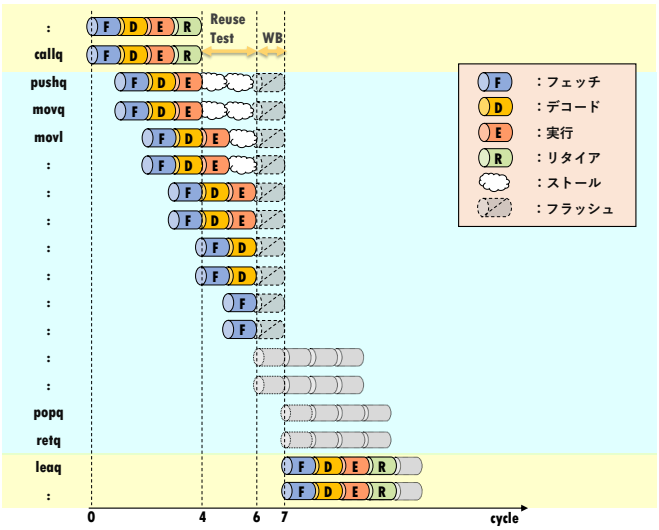

図 5　再利用テスト成功時のパイプラインの様子

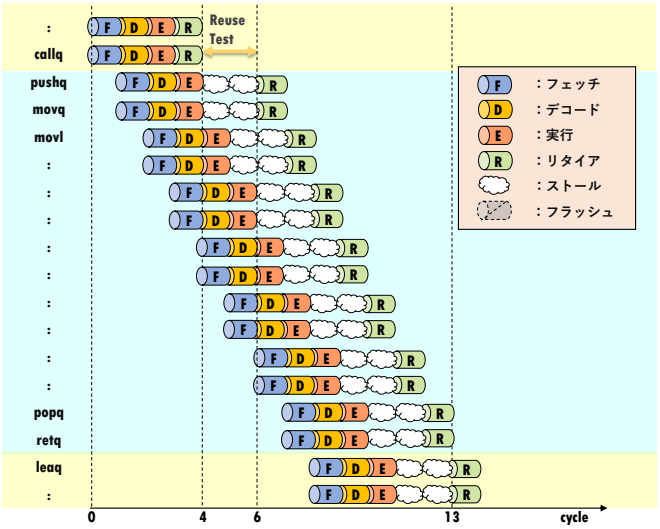

図 6　再利用テスト失敗時のパイプライン処理の様子

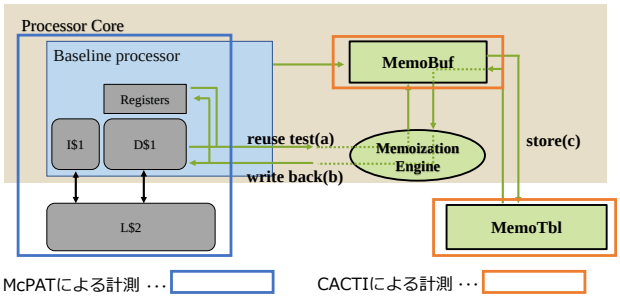

図 7　エネルギー評価の方針

に **McPAT**[4] および **CACTI-P**[5] を用いた．自動メモ化プロセッサは一般的なプロセッサが持つ構造に加え，メモ化専用ユニットを持つ．以降，メモ化専用ユニットを除いた部分にあたる，一般的なプロセッサ構成を，ベースプロセッサと呼ぶことにする．McPAT はプロセッサの電力評価に広く使われており，アーキテクチャレベルでの電力評価を行えるツールである．ベースプロセッサの消費電力についてはこの McPAT を用いることで計測した．McPAT を用いた計測によって得られた消費電力にプログラムの実行時間を掛け合わせることで，消費エネルギーを算出する．一方で，メモ化専用ユニットである MemoTbl および MemoBuf は一般的なプロセッサ構造に含まれないため，McPAT による計測が不可能である．そこで，これらのメモ化専用ユニットに対しては，メモリの消費エネルギーおよび消費電力を計測できる CACTI-P を用いた．

3.2　ベースプロセッサのエネルギー評価

前節で述べたように，ベースプロセッサの電力評価には McPAT を用いた．McPAT はプロセッサの構成および仕様が記述されたファイルを入力として受け取り，その記述に基づき，コアやメモリコントローラといったユニット毎の電力を算出する．入力ファイルにはパラメタとしてコアの数，キャッシュの階層の数，メモリコントローラの数，プロセスルールや CPU 温度，パワーゲーティングの切り替え等を記述する．本研究で扱う自動メモ化プロセッサはベースアーキテクチャとして ARM Cortex-A9[6] を想定しているため，プロセスルールや動作クロック等のパラメタはこれに従って定義した．また，コア数は 1，コアの外部には命令およびデータで共有の 2 次キャッシュ及びメモリコントローラを持つものとして計測する．以上の内容を記述したファイルを，McPAT に入力として与えることで，自動メモ化プロセッサのベースプロセッサの消費電力を計測した．McPAT で得られたベースプロセッサにおける消費電力と，プログラムの実行時間とを掛け合わせることで，ベースプロセッサにおける消費エネルギーを算出した．なお，実行時間はプログラムの実行に要した総サイクル数を動作周波数で割ることで算出でき，動作周波数は Cortex-A9 の最大周波数である 2GHz としている．

3.3　メモ化専用ユニットのエネルギー評価

MemoTbl や MemoBuf といったメモ化専用ユニットについては，ユニットレベルでのキャッシュメモリの電力評価が可能な CACTI-P を用いる．CACTI-P を用いて，キャッシュメモリをシミュレートするためには，キャッシュサイズ，メモリのタイプ（CAM，SRAM，DRAM），連想度，プロセスルールなどのパラメタを指定する必要がある．なお，MemoTbl に含まれる AddrTbl，FLTbl，OutTbl および MemoBuf は SRAM，MemoTbl に含まれる InTbl は高速な連想検索を可能にするため，CAM での実装を想定している．CACTI-P はこれらのパラメタから 読み出し 1 回，および書き込み 1 回に要する消費エネルギーと，リーク電力とをそれぞれ算出する．なお，CAM によって実装されているメモリを電力評価の対象とした際には，これらの消費エネルギーに加え，連想検索 1 回に要する消費エネル

ギーも算出される．このようにして得られた，動作 1 回あたりの消費エネルギーと動作回数とを掛けあわせることでメモ化専用ユニットの消費エネルギーを算出した．キャッシュメモリユニットの動作は read（特定行からの読み出し），write（特定行への書き込み），search（連想検索）および idle（何もしていない）の 4 つに分類できるため，メモ化専用ユニット $unit$ の消費エネルギー E_{unit} は以下の式で求められる．

$$E_{unit} = \sum_{act}(E_{act,unit} \times Acc_{act,unit}) + Ei_{unit} \times T_{unit} \quad (1)$$

ここで，Ei_{unit} および Ti_{unit} はそれぞれ，メモ化専用ユニット $unit$ におけるリーク電力および idle 時間である．式 (1) において，$E_{act,unit}$ はユニット $unit$ における動作 $act(\in \{read, write, search\})$1 回に要する消費エネルギー，$Acc_{unit,act}$ は動作 act が行われた回数を表す．

メモ化専用ユニットの消費エネルギー導出には CACTI-P で計測した動作 1 回に要する消費エネルギーの情報に加えて，各動作の回数が別途必要になるため，自動メモ化プロセッサにメモ化専用ユニットに対するアクセスカウンタを実装した．前述したようにキャッシュメモリは idle，read，write，または search のいずれかの動作をする．キャッシュメモリの動作は基本的に idle であり，特定行からの読みだし，書き込み，連想検索による読み出しのいずれかの動作を行うタイミングで read，write および search のアクセス回数を加算することで，キャッシュメモリユニットの動作回数を正確に把握する．

4. 評価

計算再利用が持つ省電力化に対するポテンシャルを確かめるため，ベンチマークプログラムを用いてサイクルベースシミュレーションにより自動メモ化プロセッサの電力評価を行った．

4.1 評価環境

スーパスカラ型の ARM をベースとするインハウスシミュレータを用い，消費エネルギーおよび実行サイクル数について評価した．消費エネルギーは 3 章で述べた電力評価方法により算出した．評価に用いたシミュレータの仕様を**表 1** に示す．MemoTbl 内の InTbl に用いる CAM の構成は MOSAID 社の DC18288[7] を参考にし，サイズは 64Bytes 幅 × 4K 行の 256KBytes とした．

4.2 評価結果

評価には，汎用ベンチマークである SPEC CPU95 INT を用いた．**図 8** のグラフは各ベンチマークプログラムの実行に要したサイクル数，**図 9** のグラフは各ベンチマークプログラムの実行に要した消費エネルギーを示している．なお，図 8 では通常実行時（メモ化なし）の実行サイクル数を，図 9 では通常実行時の消費エネルギーを，それぞれ 1 として正規化している．図 8 の凡例は実行サイクル数の内訳を示しており，r_cycle は 1 次および 2 次キャッシュミスペナルティを含んだ命令実行サイクル数，w_cycle は MemoTbl に記憶されている出力のレジスタやメモリへの書き戻しに要したサイクル数を示している．また，再利用テストとオーバラップして命令が実行されたサイクル数のうち，再利用テストに失敗した結果命令がフラッシュされず無駄にならなかったサイクル数を s_exec_overlap，再利用テストに成功し，命令がフラッシュされた結果バブルとなったサイクル数を reuse_bubble として示している．search_bubble は，再利用テスト中のリタイア待ちにより生じたストールサイクル数，reuse_pending は，再利用テスト時にキャッシュコントローラがビジーであった際の待機サイクル数を示している．

図 9 の凡例は消費エネルギーの内訳を示しており，Baseline はベースプロセッサにおける消費エネルギーの合計を，MemoBuf，FLTbl，AddrTbl，InTbl，OutTbl はそれぞれ該当するメモ化専用ユニットにおける消費エネルギーを示している．

評価の結果，各ベンチマークを通常実行した場合と比較して自動メモ化プロセッサ上で実行した場合，プログラム

表 1 評価環境

MemoBuf	128 KBytes
MemoTbl CAM	256 KBytes
Comparison	
register and CAM	9 cycles/64Bytes
Cache and CAM	10 cycles/64Bytes
Write back (MemoTbl to Reg/Cache)	1 cycles/64Bytes
L1 I-cache	16 KBytes
line size	64 Bytes
ways	4 ways
miss penalty	8 cycles
D1 cache	32 KBytes
line size	64 Bytes
ways	4 ways
miss penalty	8 cycles
D2 cache	2 MBytes
line size	64 Bytes
ways	4 ways
miss penalty	40 cycles
pipeline stage	9 stages
pipeline ways	2 ways

表 2 命令削減率

124.m88ksim	129.compress	130.li
13.2%	0.8%	4.2%
132.ijpeg	134.perl	147.vortex
0.9%	2.7%	28.1%

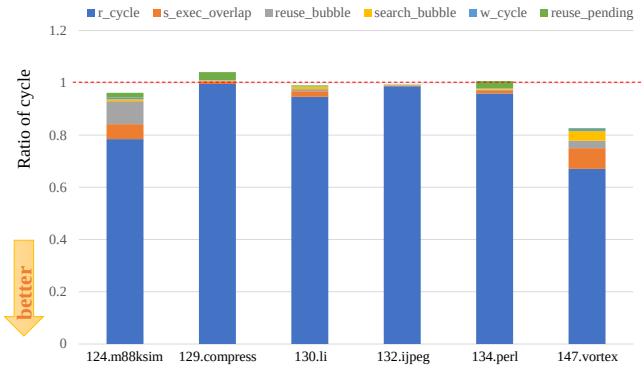

図 8　実行サイクル数比

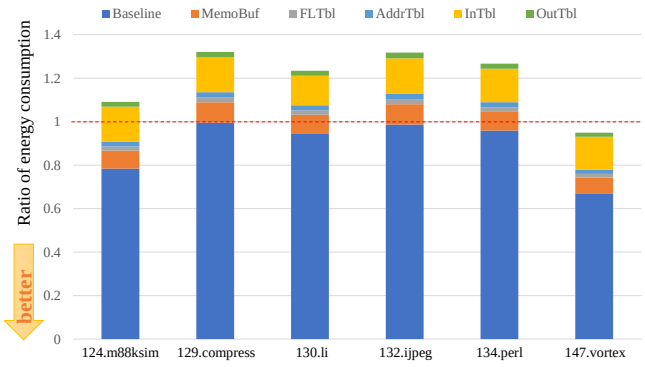

図 9　消費エネルギー比

の実行に要するサイクル数は平均 3.0%，最大 17.4%削減されることを確認した．また，消費エネルギーは通常実行した場合と比較して平均 19.6%増加したが，ハードウェアが増加しているにも関わらず 147.vortex のように消費エネルギーが減少する場合や，124.m88ksim のように増分が小さく抑えられる場合があることが確認できた．今回の実装では特にエネルギー削減のための対策を行っていないにも関わらず，このような結果が得られたことから，計算再利用は省電力化に対して一定のポテンシャルを有すると考えられる．

4.3　考察

消費エネルギーが削減された 147.vortex および消費エネルギーの増加が比較的小さい 124.m88ksim は，計算再利用に適した命令区間を多く含んでいるという特徴を持つ．それらの命令区間に計算再利用を適用できたことで，ベースプロセッサの演算器等における消費エネルギーを大きく削減できたと考えられる．一方で，消費エネルギーが増加した 129.compress，130.li，132.ijpeg，134.perl には計算再利用に適した命令区間がほとんど含まれず，メモ化専用ユニットでの消費エネルギーが計上されたことで，総エネルギーが増加してしまったと考えられる．ここで，それぞれのベンチマークプログラムにおいて，通常実行時の命令数を 100%とし，計算再利用を適用することで削減できた命令数の割合を**表 2** に示す．消費エネルギーが増加したベンチマークプログラムでは命令削減率は 5%未満にとどまっていることが分かる．なお，我々が過去に提案した，単命令発行型の自動メモ化プロセッサでは，ループも再利用の対象としているが，今回評価に用いた，スーパスカラプロセッサをベースアーキテクチャとした自動メモ化プロセッサでは，ループに対する再利用は未実装であるため，今後この命令削減率は改善される可能性がある．

また，図 9 を見ると，メモ化専用ユニットの消費エネルギーのうち，InTbl の消費エネルギーが半分以上を占めていることが確認できる．InTbl は CAM での実装を想定しているが，CAM は RAM と比較して回路面積・消費電力ともに大きい．したがって InTbl について，RAM を用いた実装を検討することが，自動メモ化プロセッサの消費エネルギーを削減するための課題のひとつである．

5.　メモ化専用ユニットの省電力化の検討

5.1　InTbl の消費エネルギー削減

4.3 節でも述べたように InTbl はメモ化専用ユニットの中でも消費エネルギーが特に大きい．これは，InTbl に CAM を用いていることが原因であり，InTbl の構造，および実装方法を変更することで改善できる可能性がある．そこで，自動メモ化プロセッサが入力を木構造で管理していることに着目し，CAM の一部を RAM で代替することで消費エネルギーを削減できないか検討する．

再利用テストにおいて，ある 1 つの入力と一致するエントリを発見した場合，次の入力と一致する可能性のあるエントリは，入力の木構造においてその一致したエントリの，子にあたるエントリのいずれかである．そのため，子エントリ以外に対して値の一致を確認する必要はないが，現実装では CAM により全エントリを対象として一致比較している．そこで，子エントリのみを対象とした検索ができれば，全検索を高速に行える CAM を利用せずとも検索オーバヘッドを抑制できると考えられる．以降では，あるノードと共通の親ノードを持つノードのことを兄弟ノードと呼び，入力の木構造におけるルートノード，兄弟ノードを持たないノード，兄弟ノードを持つノード，の 3 つがそれぞれ持つ特徴と，それに合わせた InTbl 構成について説明する．

まずルートノードは，各入力セットの最初の入力を保持するノードであり，**図 10** に示す入力パターンの木構造では，x=2 のノードがこれに該当する．ルートノードはこの入力木を検索する際，必ず検索されるノードであり，他のノードに比べて参照される回数は多い．また，親を持たないため，親子関係による検索対象の絞り込みができない．よってルートノードに関しては，これまでどおり CAM で管理する．

続いて，兄弟ノードを持たないノードについて考える．

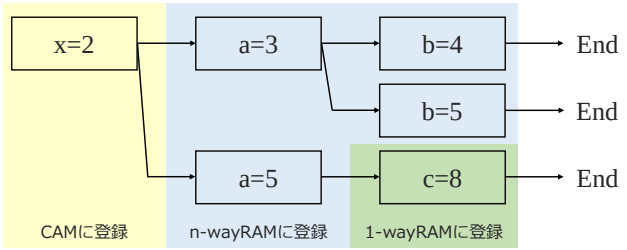

図 10 木構造に沿ったハードウェア割当

図 10 の例では c=8 のノードがこれに該当する．再利用テストの際，親ノードである a=5 が現在の入力と一致した場合，次の入力と一致する可能性があるのは，この c=8 のノードのみである．そのため，親ノードのエントリに，子ノードのエントリへのポインタを保持しておけば，次に検索が必要なエントリは一意に特定され，そのエントリとだけ値の一致比較を行えばよい．したがって，ルートノードを管理する CAM とは別に RAM で構成するテーブルを用意し，兄弟を持たないノードに関してはこの RAM を用いて管理しても，検索オーバヘッドは小さく抑えられる．

最後に，兄弟ノードを持つノードについて考える．図 10 の例では a=3 と a=5，また b=4 と b=5 が該当する．これらのノードについては，親ノードの入力が一致した場合，兄弟ノードを持たないノードのように，次に検索が必要なエントリを一つに絞りこむことはできないが，複数（往々にして少数）のエントリにまで絞りこむことが可能である．したがって，複数ウェイを持つキャッシュのように，一つのセットに複数の値を格納できる RAM を用意し，これを用いて兄弟ノードを持つノードを管理することで，複数の兄弟ノードの読み出し，および一致比較を並行して行えるようにし，検索オーバヘッドを抑制する．

以上のように，木構造で管理される入力の親子関係を用い，連想検索の対象とする必要のないエントリを RAM に登録することで，高速な検索と低い消費エネルギーを両立できると考えられる．

5.2 評価結果

5.1 で述べた方法により，どのくらいエネルギー削減が可能かを見積もるにあたり，予備評価を行った．今回の予備評価には，スーパスカラ型をベースとした自動メモ化プロセッサよりも実装が容易な，単命令発行をベースとした自動メモ化プロセッサを用い，InTbl を全て CAM で構成する場合と，InTbl の CAM を 5.1 節で述べた方針に基づいて一部 RAM で構成した場合とで，メモ化専用ユニットにおける消費エネルギーを比較した．ベンチマークには 4.1 節の評価と同様に SPEC CPU95 INT を用いた．

評価の結果，InTbl の一部を RAM で構成することで，メモ化専用ユニットの消費エネルギーが平均 32.1%，最大 45.5%削減されることを確認した．入力が持つ特徴は，単命令発行を想定した場合でもスーパスカラを想定した場合でも違いはないため，スーパスカラ型の自動メモ化プロセッサでも，これに近い結果が得られると考えられる．以上の方法により，InTbl の消費エネルギーが削減できれば，147.vortex だけでなく，エネルギー増加がほとんど見られなかった 124.m88ksim においても，総消費エネルギーは計算再利用の適用により減少する可能性がある．また，これら以外のベンチマークにおいても，総消費エネルギーが大きく抑制される可能性がある．今後は，スーパスカラ型の自動メモ化プロセッサにこれらの手法を実装し，詳細な評価を行っていく予定である．

6. おわりに

本稿では，計算再利用が持つ，プロセッサ省電力化に対するポテンシャルを確認するために，自動メモ化プロセッサの消費エネルギーを評価した．その結果，計算再利用のための専用ユニットが新たに追加されているにもかかわらず，一部のプログラムにおいて総消費エネルギーの減少が確認できた．専用ユニットの中でも，CAM での構成を想定している InTbl での消費電力が大きいことを確認した．この結果を受け，InTbl の一部を RAM で構成する手法について検討し，予備評価によりその手法の有効性を確認した．今後の課題としては，InTbl 以外のメモ化専用ユニットの消費電力の削減や，パワーゲーティング等一般的な省電力化手法の適用方法の検討などが挙げられる．

参考文献

[1] Tsumura, T., Suzuki, I., Ikeuchi, Y., Matsuo, H., Nakashima, H. and Nakashima, Y.: Design and Evaluation of an Auto-Memoization Processor, *Proc. Parallel and Distributed Computing and Networks*, pp. 245–250 (2007).
[2] Norvig, P.: *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann (1992).
[3] Huang, J. and Lilja, D. J.: Exploiting Basic Block Value Locality with Block Reuse, *Proc. 5th Int'l Symp. on High-Performance Computer Architecture (HPCA-5)*, pp. 106–114 (1999).
[4] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M. and Jouppi, N. P.: McPAT: an Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures, *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, IEEE, pp. 469–480 (2009).
[5] Li, S., Chen, K., Ahn, J. H., Brockman, J. B. and Jouppi, N. P.: CACTI-P: Architecture-Level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques, *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, IEEE, pp. 694–701 (2011).
[6] arm: Cortex-A9-Arm Developer-, `https://developer.arm.com/ip-products/processors/cortex-a/cortex-a9`.
[7] MOSAID Technologies Inc.: *Feature Sheet: MOSAID Class-IC DC18288*, 1.3 edition (2003).