エネルギー制御を備える自動メモ化プロセッサ

島崎裕介[†] 津邑公暁[†] 中島浩[††]
松尾啓志[†] 中島康彦[†††]

我々は，計算再利用技術に基づく自動メモ化プロセッサを提案している．本稿では，自動メモ化プロセッサに電力評価モジュールを実装し，シミュレータレベルでの消費エネルギー評価を行った．メモ化により高速化を図ることは可能だが，プログラムによってはメモ化の効果が現れず，メモ化機構追加分のエネルギーを余分に消費する場合がある．そこで新たに，計算再利用率に応じてメモ化の中断や再開を行わせることで，本来メモ化の効果が得られる時間区間のみメモ化機構を動作させる機能を実装した．この機能による削減サイクル数の低下はほとんど見られず，プログラム中全ての関数をメモ化対象とする従来手法の場合，SPEC CPU95 では平均 15.4% ，GA プログラムでは平均 1.9%のエネルギー増加が起きていたが，本手法により SPEC CPU95 で平均 5.9%の増加，GA プログラムでは平均 3.7%の減少となり消費エネルギーを削減することができた．この結果から，メモ化の中断および再開により，高速性をほとんど損なうことなく低消費エネルギー化を実現できることが分かった．

An Auto-Memoization Processor with Energy Control Technique

Yusuke Shimazaki,[†] Tomoaki Tsumura,[†] Hiroshi Nakashima,[††]
Hiroshi Matsuo[†] and Yasuhiko Nakashima[†††]

We have proposed an auto-memoization processor. We implemented power-analysis method for our auto-memoization processor. This paper describes the energy consumpiton of auto-memoization processor. With memoization, some programs turn out to finish faster, but others do not. In the latter case, the processor increases their energy comsumption, without speed-up by the memoization. To decrease the energy comsumption, we added a function to the auto-memoization processor to discontinue and continue memoization. The result of the experiment with SPEC CPU95 suite benchmarks and GENEsYs benchmarks shows that original memoization units increase whole energy by 15.4% and 1.9% on geometric mean respectively. After adding the proposing function, in each benchmarks, the processor turned out to increase whole energy comsumption by 5.9% with SPEC CPU95, and decrease whole energy consumption by 3.7% with GENEsYs, on geometric mean. Consequently, by discontinuing and continue memoization, the memoization units can decrease total energy consumption with few speed-down against traditional auto-memoization processor.

1. はじめに

ゲート遅延が支配的であったこれまでは，微細化による高クロック化で高速化を実現できた．しかし配線遅延の相対的な増大に伴い，高いクロックだけでは高速化を実現しにくくなっている．このような状況の下 SIMD やスーパスカラなどの命令レベル並列性（ILP: Instruction Level Parallelism）に基づく高速化手法が注目されてきた．しかし，多くのプログラムは明示的な ILP を持たないことや，ILP を抽出できる場合でもメモリスループットなどの資源的制約があることから，これらの手法にも限界がある．

そこで我々は，従来の高速化手法とは着眼点の異なる計算再利用技術に基づく，自動メモ化プロセッサを提案している[1)]．**メモ化**（**Memoization**）[2)] は本来，主に lisp などで使用されるプログラミングテクニックであり，計算量の大きい関数に対してその入出力を保存しておくことで，同一入力による当該関数の再計算を省略し実行を高速化する手法である．我々の提案する自動メモ化プロセッサは，既存のバイナリプログラムにおいて関数を実行時に動的に検出し，その入出力をハードウェアで記憶することで再計算の省略を自動的に行うアーキテクチャである．

この自動メモ化プロセッサは，動的に切り出した命令区間の入出力を表の形で保存する．また入力一致比較のため，当該区間実行のたびにこれを検索する必要

† 名古屋工業大学
Nagoya Institute of Technology
†† 京都大学
Kyoto University
††† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

がある．我々はこの表を CAM (Content Addressable Memory) を用いて構成することを仮定しているが，この表は参照頻度が大きく，また CAM 自体も消費電力が大きいため，プロセッサ全体のエネルギー消費量を大きく増加させてしまう可能性がある．

計算再利用技術の消費電力に対する影響に関しては，load/store 命令のみを再利用する場合に消費電力が抑えられる場合もあるという報告[3] があるが，これまで厳密な調査は行われていなかった．そこで本稿では，SimpleScalar 向け電力評価フレームワークである Wattch を参考に自動メモ化プロセッサシミュレータに電力評価機構を実装し，その消費エネルギーの評価を行った．また，メモ化の技術を用いても，プログラムによってはメモ化の効果がなく高速化が図れない場合があることが分かっている．このようなプログラムに対してメモ化の効果を動的に観測し，効果が得られにくいと判断した場合にはメモ化を中断してメモ化機構への電力供給を遮断する，という動作を実装し評価を行った．

2. 自動メモ化プロセッサ

2.1 概要と動作モデル

メモ化とは，関数の入出力ペアを配列等に記憶させることで，当該関数の同一入力による実行を省略する高速化手法である．我々の提案している自動メモ化プロセッサは，プログラム実行中に関数を命令区間として動的に検出し，それらを自動的にメモ化する．具体的には，`call` 命令のターゲットと `return` 命令の間の区間を関数として検出する．

図 1 にプロセッサ構成の概略を示す．自動メモ化プロセッサは，入出力を記憶するためのテーブル（以下，**MemoTbl**）と，MemoTbl への書込バッファ（以下，**MemoBuf**）を持つ．メモ化機構は上記命令区間の開始を検出すると，MemoTbl に記憶されている当該区間の入力アドレス全てに対応する値をキャッシュから読みだし，MemoTbl 内の入力値とその値とを比較する．MemoTbl 内に完全に一致するエントリが存在した場合，そのエントリに対応する出力を MemoTbl から読みだし，レジスタおよびキャッシュに書き戻すことで命令区間の実行を省略する．

一致するエントリが存在しなかった場合，通常どおり命令区間を実行するが，その際レジスタおよびキャッシュへの参照を入力，書込みを出力として MemoBuf に記録する．そして命令区間を終端まで実行すると，MemoBuf 内に蓄積された入出力セットを MemoTbl に保存する．関数の入力として扱われるものには関数の引数および関数内で参照・書込みが行われた変数があり，出力として扱われるものには書込みが行われた変数があたるが，局所変数は除外される．我々の手法では，一般に OS がデータサイズおよびスタックサイズの上限を決定することを利用し，この上限および関数呼出が行われる直前のスタックポインタの値と変数アドレスとの関係から，局所変数を判別している．

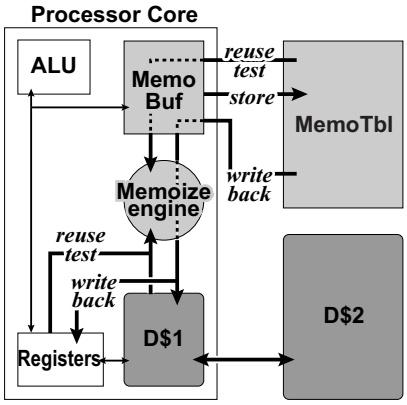

図 1 自動メモ化プロセッサの概要

2.2 メモテーブルの構成

一般に命令区間内においては，ある入力の値によって次に参照すべき入力アドレスが変化する．変数に主記憶アドレスが格納されている場合や，条件分岐の存在がこの原因である．つまりある命令区間の全入力パターンはツリー構造で表すことができ，ある入力セットはそのツリー上の 1 本のパスで表される．このようなデータ構造の格納・検索を可能とするため，我々は MemoTbl を，以下の複数の表を用いて構成している．

RF: 命令区間の開始アドレスを記録しておくための表であり，RAM で構成する．

RA: 命令区間の入力アドレスを記録しておくための表であり，RAM で構成する．

RB: 命令区間の入力値を記録しておくための表であり，CAM で構成する．

W1: 命令区間の出力アドレスおよび出力値を記録しておくための表であり，RAM で構成する．

紙面の都合上，詳細は文献 1) にゆずるが，MemoTbl 検索手順の概要は以下の通りである．まず現在のレジスタ上の入力値を RB から検索する．RB の各エントリは，次入力アドレスを格納する RA エントリへのインデクスを保持している．RB 内で入力が一致するエントリが存在した場合，RA から得た次入力アドレスを用いてキャッシュを参照し，次入力値を得る．この入力値を用いて再び RB を検索する．これを繰り返し，全ての入力の一致が確認できると，入力セットの終端を保持する RB エントリは W1 へのインデクスを保持しており，これを用いて W1 を参照して出力値を得，これをレジスタおよびキャッシュに書き戻すことで命令区間の処理を省略する．

3. 消費電力の見積り

3.1 Wattch

自動メモ化プロセッサのエネルギー消費量を見積も

るにあたり，**Wattch**[4] を参考に，シミュレータに消費電力推定の機構を実装した.[5] Wattch は，アーキテクチャレベルの消費電力シミュレータであり，SimpleScalar へのパッチという形で提供されている．以下，Wattch における電力の測定方法について説明する．

3.1.1 ベース消費電力の決定

Wattch はプロセッサ内ユニットを RAM，CAM，組合せ回路，クロック回路の 4 つに大別し，ユニットそれぞれに対する静電容量や電源電圧，クロック周波数を用いて，ベース電力を決定している．ここでベース電力とは，プログラムの処理において最終的に消費した電力を求めるために，一定のアクセス回数に対して消費するとされる一定の電力である．RAM や CAM 部分に関しては，cacti[6] を参考に実装モデルを決定している．cacti はキャッシュにおいて遅延時間が最適となるハードウェア構成を決定するツールである．Wattch ではその遅延時間とハードウェア構成をもとに，キャッシュ全体やキャッシュの TLB (CAM) の消費電力を算出する．

以下，大別した各ユニットのベース電力の算出について簡単に述べる．

RAM：エントリ数，幅，読み書きポート数をパラメータとしてベースとなる消費電力が決定される．デコーダ，語ライン，ビットライン，出力のそれぞれでモデル化され，総消費電力が算出される．

CAM：RAM と同様，タグライン，マッチライン等をベースに算出される．

組合せ回路：リザルトバスや ALU など，それぞれの構成に応じて消費電力が計算される．

クロック回路：クロック線，クロックバッファ，クロックロードのそれぞれにおいてモデル化を行い，クロック回路における総消費電力が算出される．

3.1.2 消費エネルギーの算出

Wattch では各ユニットにおける消費エネルギー E，及び単位時間当りの平均電力 P を以下の式で算出している．

$$E = P_{base} \int_0^T A_{r/w}(t) \cdot a_f(t) dt \qquad (1)$$

$$P = E/T \qquad (2)$$

P_{base} はそのユニットにおけるベース電力であり，$A_{r/w}(t)$ はユニットへの時刻 t におけるアクセスに応じて正の相関を成して変化する係数である．また，$a_f(t)$ はゲート稼働率であり，$0 \leq a_f(t) \leq 1$ と動的に変化する場合と静的に $a_f(t) = 0.5$ と固定する場合がある．ただし，ユニットによっては計算過程に $a_f(t)$ を含めないものもある．

(1) 式でベース電力を稼働時間 T により積分することで，そのユニットにおける最終的な消費エネルギーを求める．また (2) 式では，(1) 式で求めた E を用いて単位時間当りの平均電力 P を求めている．以下，本稿では単に消費電力と記述した場合この平均電力を指すこととする．

3.2 実　　装

まず，メモ化機構に含まれないプロセッサ要素であるクロック回路，キャッシュ，ALU，レジスタファイルに関しては，Wattch の評価式をそのまま移植する形で実装した．

また 2.2 節で述べたように，我々の想定するメモ化機構は，MemoBuf および MemoTbl（RF, RA, RB, W1）から成る．これらについては，それぞれ以下のような RAM および CAM として消費電力評価関数の実装を行った．

MemoBuf：MemoTbl への書込バッファとして頻繁にアクセスされる．Wattch における 2 ポートの 1 次キャッシュと同じ電力評価式で実装した．ブロックサイズを 32B，総容量を 19kB とした．

RF：命令区間表であり，複数ポートは必要ないため 1 ポートの 1 次キャッシュとして実装した．サイズは幅 46B，エントリ数 256 の，総容量 12kB とした．

RB：入力値セットのための表であり，連想検索が必要となる．Wattch では TLB が CAM を使って構成されているため，これと同じ電力評価式を用いて実装した．幅 36B，深さ 1k 行の，総容量 36kB とした．

RA，W1：入力値のアドレス表および出力値表である．1 ポートの 2 次キャッシュと同じ評価式で実装した．RA，W1 の幅はそれぞれ 25B，75B とした．エントリ数は RB と同じ 1k であるため，総容量はそれぞれ 25kB，75kB となる．

4. 低消費エネルギーモデル

4.1 メモ化機構と消費エネルギー

前述したようにメモ化は，MemoBuf 及び MemoTbl (RF, RB, RA, W1) のメモ化機構を必要とし，これらユニットの消費電力を考慮する必要がある．

一方で，メモ化はプログラムの値の局所性を利用した高速化手法であるため，値の局所性が無いプログラムの場合にはメモ化による高速化が期待できない．メモ化によりプログラム実行の高速化が実現できるかどうかは実行時でないと分からず，値の局所性が顕著に現れない時間区間ではメモ化機構を用いる必要がない場合もある．そのような場合において問題となるのは，メモ化機構を稼働させることによる無駄な消費電力及びエネルギーの増大である．

プログラム中においてメモ化によるサイクル数削減効果の見込めない時間区間では，メモ化を行う必要が無いといえる．そこでプログラム中，動的にメモ化の効果を判定することで前述した区間を判定し，本来メモ化の効果が得られる区間のみをメモ化対象とするアルゴリズムを提案する．

この考えに基づく自動メモ化プロセッサの実現にあたり，新たに 2 バイト以下の小さなカウンタを 2 つ用意する．なお，この追加したカウンタはプログラム

カウンタに比べアクセス頻度は極めて低く，プロセッサ全体に及ぼす電力増加は無視できるほどわずかである．このカウンタを用い実行対象となるプログラム中において次の 2 つを計測する．

(1)　関数 call が発生した回数
(2)　計算再利用が成功した回数

(2) は図 1 において `writeback` が発生した回数に相当する．これらの値を用い，(1) の関数が一定回数呼ばれた際に (2) が一定値に達しているか調べることでメモ化機構が有効に働いているかを検知することができる．

4.2　メモ化中断アルゴリズム

メモ化による再利用成功率が低い場合，メモ化を中断しメモ化機構への電力供給を遮断することで，無駄なエネルギー消費を抑えることができる．そのメモ化中断を決定する判定方法を以下に定める．以下，本稿ではこれを**メモ化中断**アルゴリズムと呼ぶ．

中断判定を行うために関数呼び出し発生回数に対する N_f，および計算再利用成功回数に対する N_r の 2 つの閾値を定義する．

関数呼び出しが N_f に達する前に計算再利用が N_r 回発生した場合には，メモ化が有効に働いていると判断し，現在の関数呼び出しおよび計算再利用の回数をリセットして引き続きメモ化及びメモ化の閾値判定を行っていく．一方，計算再利用が N_r 回発生する前に呼出し回数が N_f に達した場合にはメモ化機構を停止し，計算再利用を行わない通常のプロセッサとして動作する．このときメモ化機構への電力供給をシャットダウンするため，MemoBuf および MemoTbl の内容は揮発する．

なお，ひとたびメモ化が中断してしまうと，それ以降計算再利用による高速化は一切望めなくなるため，メモ化有効性チェックのためのパラメータは比較的中断が起こりにくいよう設定した．メモ化中断の閾値でパラメータ N_f, N_r の値については，本稿ではメモ化の効果を確認できる十分な時間として $N_f = 1024$ $(= 2^{10})$ とした．また，一般にメモ化によるサイクル数削減効果のみられるプログラムでは，関数呼び出し回数のうちの数%以上の再利用が起きているため，計算再利用回数 N_r を関数呼び出し回数 N_f のうちの約 1.5%にあたる $N_r = 16$ $(= 2^4)$ とした．これに加えて，メモ化効果のチェック履歴を記憶し，4 回連続してチェックをパスした場合それ以降のチェックにおいて 1 度だけ閾値を下回ることを容認することで，本来メモ化効果の得られるプログラムで中断が起こりにくくなるよう配慮した．

4.3　メモ化再開アルゴリズム

前節アルゴリズムはメモ化の中断のみを考慮したものであるが，プログラムにおいてはメモ化効果に時間的局所性のあるものがある．例えばメモ化の効果がプログラム実行開始直後には現れないが後のカーネルループで現れてくるようなプログラムに対しては，前節で述べたアルゴリズムを用いてメモ化を中断した場合，本来得られるべきメモ化による高速化の恩恵が受けられないことがある．

しかし，メモ化の有効性は実際にメモ化機構を動作させている状態で初めて判定することが可能なため，メモ化を中断した以降も適宜メモ化を再開し，プログラム中におけるメモ化効果を判定できる状態にしておく必要がある．よって次に，メモ化を中断した状態からメモ化を再開するアルゴリズムを提案する．以下，本稿ではこれを**メモ化再開**アルゴリズムと呼ぶ．

具体的には，メモ化中断状態から一定期間経過すると再びメモ化機構を作動させ計算再利用を試みるモデルである．ここで中断状態を維持する一定期間は，関数呼出しが $2^k \times N_r$（k は整数）回起こるまでとする．再開後は前項で述べたメモ化中断アルゴリズムに従う．もちろん，再開時には MemoBuf および MemoTbl には中断以前の情報は保存されていない．

なおパラメータに関しては，再開アルゴリズムを中断アルゴリズムと組み合わせて使用する場合，あえて中断を起こりにくくする必要はない．このため N_r は前節より大きく 32 $(= 2^5)$ とした．また前節で述べたメモ化のチェック履歴を用いて動作を変化させるアルゴリズムを用いない代わりに，中断期間の長さを決定する k を動的に変化させることとした．具体的には初期値を $k = 1$ からメモ化中断と判断されるごとに最大値 $k = 4$ までインクリメントし，メモ化再開条件を満たすと初期値 $k = 1$ にリセットする．これにより，メモ化の効果がほとんど得られないプログラムにおいて頻繁なメモ化再開によるエネルギー消費を抑えることができる．

なおメモ化再開時においては MemoBuf および MemoTbl のウォームアップ時間を考慮する必要があるが，これはゲーティッド V_{dd}[7] などの他の供給電力制御手法と違い，その性能に与える影響はほとんどないと言える．本手法では再開条件を満たしたと判断されるのはある関数呼出しの発生時であるが，当該関数はまだメモ化対象ではないためメモ化機構は機能を提供する必要がない．メモ化が再開されるのは次の関数呼出しからであって，それが発生するまでに準備ができていれば十分である．汎用 GA プログラムである GENEsYs を用いてこの影響を予備評価したところ，再開判定後 MemoBuf/MemoTbl を使用できない期間が数万クロック程度以下であればほとんど影響がないことを確認した．

5. 評　　価

5.1　評 価 環 境

以上で述べたように，自動メモ化プロセッサシミュレータに消費電力評価の機能を追加し，更に計算再利用の成功率に基づき動的にメモ化の中断・再開を行う機構を実装し，評価を行った．シミュレータは単命令

表 1 シミュレータ諸元

Register 幅	32 bits
Register 本数	72 entries
FloatingPoint Register 幅	64 bits
FloatingPoint Register 本数	32 entries
プロセスルール	0.18 μm
D1 Cache 容量	32 KBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	2 cycles
ミス ペナルティ	10 cycles
D2 Cache 容量	2 MBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	10 cycles
ミス ペナルティ	100 cycles
Register Window 数	4 sets
Window ミス ペナルティ	20 cycles/set
MemoBuf. サイズ	19 kB
MemoTbl. サイズ (CAM 部)	36 kB
MemoTbl. サイズ (RAM 部)	112 kB
MemoTbl. ⇔ レジスタ 比較	32 Byte/cycle
MemoTbl. ⇔ キャッシュ 比較	32 Byte/2cycles

発行の SPARC-V8 をベースとし，命令レイテンシは SPARC64-III[8] を参考にした．評価に用いたパラメータを**表 1** に示す．

ベンチマークプログラムには，SPEC CPU95 及び GENEsYs を用い，以下の 4 種類の場合に対しシミュレーション及び評価を行った．

(O) オリジナル（メモ化なし）
(M) 自動メモ化プロセッサ（従来手法）
(E_A) 省エネルギーモデル（メモ化中断）
(E_{AR}) 省エネルギーモデル（メモ化中断＋再開）

5.2 SPEC CPU95

まず SPEC CPU95 (train) を gcc 3.0.2 (-O2 -msupersparc) によりコンパイルし，スタティックリンクにより生成したロードモジュールを用いて評価を行った．**図 2**，**図 3** および**表 2** に結果を示す．

図 2 は各ベンチマークプログラムが要した実行サイクル数を表しており，左から順に (O) オリジナル，(M) 従来手法，(E_A) 中断手法，(E_{AR}) 中断＋再開手法における実行サイクル数である．図 3 は図 2 同様消費エネルギーを表している．図 3 の凡例は消費エネルギーの内訳であり，順にクロック (Clock)，一次キャッシュ (D$1)，二次キャッシュ (D$2)，演算器 (ALU)，レジスタファイル (Regfiles)，およびメモ化のための機構の各構成要素 (MemoBuf, RF, RB, RA, W1) における消費エネルギーを示している．なお，図 2, 図 3 では (O) の値で全て正規化してある．また表 2 は SPEC CPU95 全体を平均した結果であり，各値はやはり (O) を基準とした増減で表している．

従来手法 (M) では，124.m88ksim や 147.vortex のようにおよそ 20%以上のサイクル数が削減可能な場合に，消費エネルギーをオリジナルと同程度もしくはそれ以下に抑えることができると分かる．しかし，SPEC CPU95 は平均削減サイクル数 6.8%と，メモ化による効果が得にくいプログラムが多い．その結果平均消費エネルギーは 15.4%増となり，比較的現実的な値に抑えられているとはいえ，101.ijpeg や 129.compress 等，メモ化の効果が現れないプログラムにおいて無駄なエネルギー消費が目立っている．

メモ化中断手法 (E_A) では 099.go や 126.gcc 等，計 8 つのプログラムにおいてメモ化の中断が起きているが，101.tomcatv 124.m88ksim 及び 147.vortex では中断が起こらず (M) と同じエネルギー消費となっている．平均消費エネルギーは 1.7%増となり，(M) の

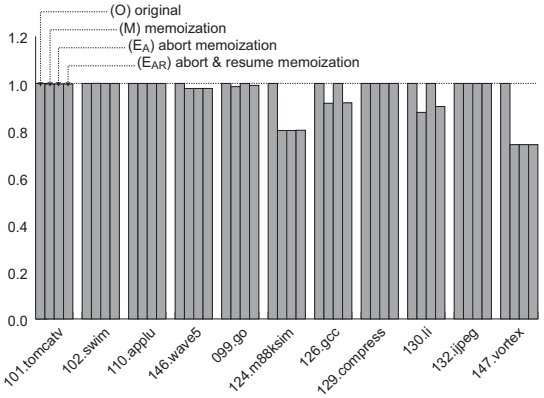

図 2 実行サイクル数比 (SPEC CPU95)

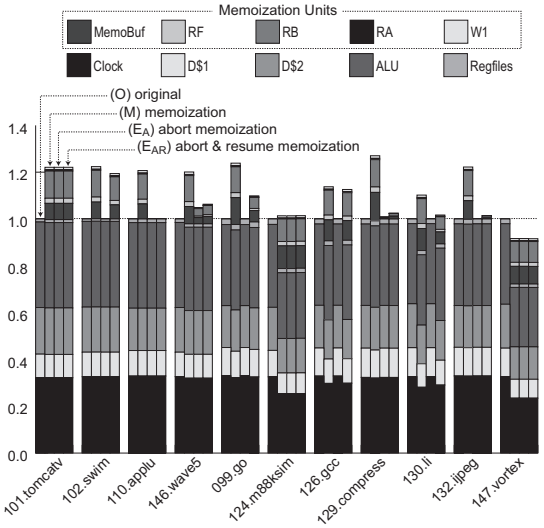

図 3 総消費エネルギー比 (SPEC CPU95)

表 2 SPEC CPU95 の結果（平均）

	サイクル数	消費エネルギー
(M) 従来手法	−6.8%	+15.4%
(E_A) 中断手法	−4.9%	+1.7%
(E_{AR}) 中断＋再開手法	−6.5%	+5.9%

6.8%と比べかなりのエネルギー消費の抑制が実現できている．また，中断の起きた 8 つのプログラムの消費エネルギーに限っては，(M) では平均で 19.9%増だったものが，僅か 0.7%増にまで減少した．中断により消費エネルギーの抑制が効果的に行われたことが分かる．

一方で全体の平均削減サイクル数は 4.9%となり，従来手法の 6.8%と比べ削減サイクル数は有意に減少してしまった．これは本来メモ化の効果が得られるものまでメモ化中断してしまったことによると考えられる．従って，(E_A) はエネルギー消費を抑制するが，同時にメモ化による高速化も抑制してしまう可能性の高い手法だといえる．

最後に中断＋再開手法 (E_{AR}) では，メモ化の再開動作により，(E_A) で高速化されていた 124.m88ksim や 147.vortex だけでなく，126.gcc, 130.li においても高速化がみられた．特に 130.li に関しては，本来メモ化による高速化の恩恵を受けられる時間区間とメモ化の効果がない区間を上手く判定できていると考えられ，(M) で余分に消費していたエネルギーが削減され (O) と同程度のエネルギー消費に抑えられている．

(E_{AR}) による全体の平均削減サイクル数は 6.5%となり，(E_A) の 4.9%と比較すると，より従来手法 (M) の高速性を維持できた結果となった．また平均消費エネルギーは 5.9%増となり，(M) の 15.4%と比べかなりの消費エネルギーの抑制が実現できている．

以上の結果から，メモ化の再開動作を取り入れることで，中断アルゴリズムだけでは損なわれてしまう計算再利用による高速性を維持しつつ，消費エネルギーを削減できることが分かった．

5.3 GENEsYs

次に，汎用 GA ソフトウェア GENEsYs 1.0 を用いて評価を行った．GENEsYs には，GA のベンチマークや定量的評価に良く用いられる De Jong のテスト関数，巡回セールスマン問題，フラクタル関数などの標準的な関数をはじめとする，24 種の適合度関数が実装されている．メモ化効果はプログラムの記述方式に依存して変化するため，今回は，メモ化効果の高い適合度関数をメモ化効果が得られやすいように書き換えたものを用いて評価した．また，交叉アルゴリズムは 2 点交叉とした．

表 3 に GENEsYs の実行パラメータ，**図 4**，**図 5**，および**表 4** にその結果を示す．図 2，図 3 と同様に，24 種の各適合度関数の結果は全てオリジナル (O) の値により正規化した．使用した gcc のバージョン，コンパイルオプションなども SPEC CPU95 と同様である．

GENEsYs では全般的に大きなメモ化効果が得られており，全 24 適合度関数のうち 6 関数で，総消費エネルギーがメモ化なし (O) の場合より抑えられていることが分かる．また GENEsYs では，特に全体に対して適合度計算量の占める割合の大きい関数，すなわち処理時間が長くかかる関数ほどメモ化の効果が得られることが分かっており GA は処理時間面においても消費エネルギー面においてもメモ化の恩恵をうけやすいプログラムであると言える．

表 3 GENEsYs パラメータ

交叉率	60.0 %
突然変異率	0.1 %
個体数	50
世代数	25 世代
他のパラメータ	default 値

従来手法 (M) の全体における平均削減サイクル数は 18.1%，平均消費エネルギーは 1.9%増となった．しかし SPEC CPU95 同様，やはりメモ化による高速化が起きていないものが存在し，いくつかの適合度関数で無駄にエネルギーを消費しているのがわかる．

中断手法 (E_A) では，f1 をはじめ計 15 の適合度関数でメモ化の中断が起きたが，f2, f3, f4, f5, f11, f12, f14, f16, f17 ではメモ化の中断が起こらず (M) と同じエネルギー消費となっている．平均消費エネルギーは 4.7%減となり，メモ化中断の導入によって消費エネルギーの削減が可能となった．

中断の起きた計 15 の評価関数の消費エネルギーに限れば，(M) では 12.4%増だったのが (E_A) では 1.0%増にまで抑えられており，効果的に消費エネルギーの抑制ができたことが分かる．一方，f18, f19, f20 の 3 関数に関しては，本来高速化が見込めるにもかかわらずプログラムの序盤でメモ化の中断が起きたため，高速化も消費エネルギー削減もできない結果となった．このため実行サイクル数の削減率は 13.7%となり，(M) の 18.1%と比較すると有意に性能低下している．

中断＋再開手法 (E_{AR}) では，メモ化の再開動作により，メモ化中断手法で大きく高速化できていた f5 や f16, f17 だけでなく，(M) で高速化できている全ての評価関数において，メモ化再開による高速化がみられた．特に f9 や f18, f19, f20 に関しては，SPEC CPU95 の 130.li 同様，定期的にメモ化を再開したことにより，メモ化なし更にはメモ化中断手法と比べても消費エネルギーが削減される結果となった．

全体の平均削減サイクル数は 17.3%となり，(E_A) では高速性が犠牲になっていたがより (M) に近い高速性が実現できた．また平均消費エネルギーは 3.7%減となり，1.9%増であった (M) と比べ消費エネルギーの削減が実現できている．

以上の結果から，(E_{AR}) ではメモ化効果の高い区間においてのみメモ化機構を動作させることをほぼ実現できており，従来手法 (M) の高速性をほとんど損なうことなく効果的に消費エネルギーの削減を行うことができることが分かった．

5.4 考　　察

オリジナル (O) および従来手法 (M) においては電力的な遮断は行っていない．また，(2) 式に示したように，消費電力に対し図 2 や図 4 に示している時間

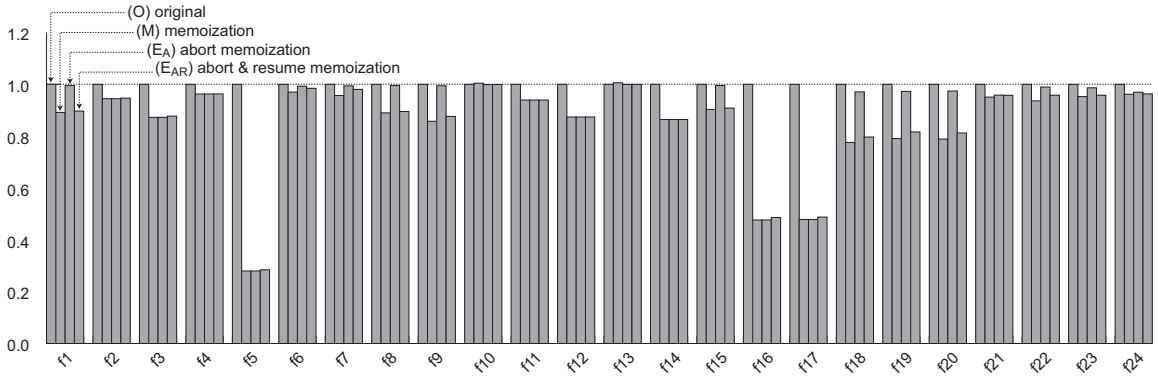

図 4 実行サイクル数比 (GENEsYs)

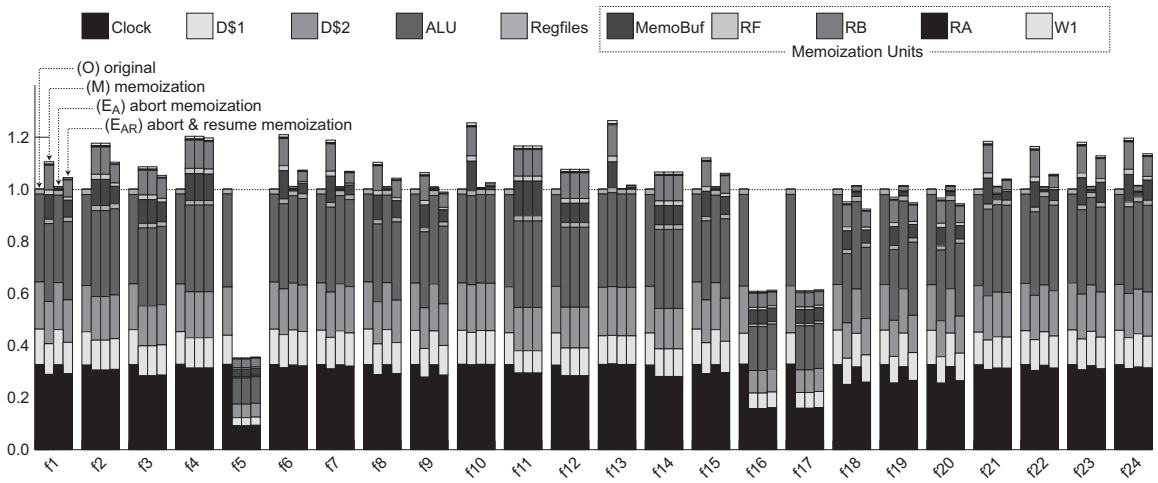

図 5 総消費エネルギー比 (GENEsYs)

表 4 GENEsYs の結果

	サイクル数	消費エネルギー
(M) 従来手法	−18.1%	+1.9%
(E_A) 中断手法	−13.7%	−4.7%
(E_{AR}) 中断＋再開手法	−17.3%	−3.7%

を乗算したものが消費エネルギーであり，これが図 3 や図 5 の結果である．従って (O) や (M) においては，図 3 で示す消費エネルギーの割合がそのまま消費電力の示す割合となる．

メモ化のための機構により消費電力は 25%弱ほど増加しているが，メモ化機構の消費電力の大部分は MemoBuf および RB (CAM) によって占められていることが分かる．MemoBuf は一次キャッシュと同様の構成を想定しており容量も小さいが，参照頻度が高いため一次キャッシュ以上の電力を消費している．また RB はサイズおよび参照頻度はそれほど大きくないが，CAM で構成されているためやはり消費電力が大きい．

実行サイクル数と，メモ化機構を含まない部分の消費電力を比較すると，サイクル数の減少以上に消費電力の方が多くのプログラムで減少していることが分かる．これは，計算結果の再利用によりキャッシュミスやキャッシュアクセス自体が減少したことと，入出力登録時，命令区間内で書込みが発生した変数はその後キャッシュではなく MemoBuf から参照されることに起因していると考えられる．

なお SPEC CPU95 において，従来手法 (M) では高速化が起きておらずメモ化がそもそも必要ないにもかかわらず，中断手法 (E_A) や中断＋再開手法 (E_{AR}) を用いた場合でもメモ化の中断が起こらず結果的に無駄にエネルギーを消費してしまっているプログラムが存在する．一般にメモ化の中断が効果的に行えない理由として次の 3 つが考えられる．

(1) 値の局所性がプログラム全体を通じて存在
(2) プログラムが小さく，閾値判定の前に終了
(3) 実行サイクル数の非常に大きな関数が存在

SPEC CPU95 では，124.m88ksim や 147.vortex

が（1）に該当しており，メモ化により実行サイクル数が 20%以上削減できている．

一方，126.gcc や 130.li では本来実行サイクル数を 10%ほど削減できるところ，プログラムの前半に値の局所性が無かったため (E_A) ではメモ化が中断されてしまっている．一方 (E_{AR}) では，中断後も定期的に値の局所性の有無を確認したことで，このようなプログラムに対しても高速化を図ることが可能となった．

（2）については，閾値判定の間隔を狭めることでいくぶん対処はできるが，メモ化による MemoTbl への登録自体があまり起きていない可能性が高く，そもそもメモ化には適していないと考えられる．

（3）には 101.tomcatv や 102.swim が該当する．これらのプログラム中には，比較的短く，call 回数および再利用成功率が高い関数と，長大で call 回数および再利用成功率が低い関数が混在している．このような場合，短い関数のメモ化効果により全体としても再利用率が高いと判断され中断が起こらない．しかし実際に実行時間やメモ化オーバヘッドの多くは長大な関数が占めている．よってメモ化中断が起こらないまま，無駄なエネルギー消費を計上し続けてしまう．

このような場合にも中断を行わせるには，関数呼出し回数ではなくクロックサイクルやレジスタアクセス回数による閾値判定が有効であると考えられる．しかしそうした場合，長大な関数の実行中に中断判定が行われることとなり，機構が複雑化してしまう．それに加えて，長大な関数が計算再利用可能な場合であってもメモ化による高速化が得られなくなってしまう．以上のことより，(3）は高速化を維持しつつ消費エネルギーを抑制するのが困難なプログラムであると言える．

6. おわりに

本稿では，計算再利用技術に基づく自動メモ化プロセッサに対し，その消費電力をアーキテクチャレベルで評価した．命令区間の入出力を記憶するための CAM 等による消費電力の大きな増加が懸念されたが，36kB の CAM を含むメモ化機構による消費電力増加は約 25% であった．また消費エネルギーでは，SPEC CPU95 で平均 15.4%の増加，GENEsYs で平均 1.9%程度の増加となることが確認できた．このことから，メモ化の効果が得られるプログラムに対しては僅かなエネルギー増加でサイクル数を削減できることが分かった．

次に，計算再利用率に応じてメモ化機構を停止・動作させるモデルをシミュレーションにより評価したところ消費エネルギーは，SPEC CPU95 で平均 5.9%の増加，GENEsYs で平均 3.7%の減少となり，更なる消費エネルギーの削減が可能となった．また削減サイクル率は，メモ化機構を動作し続けた場合は SPEC CPU95 で 6.8%，GENEsYs で 18.1%だったのに対し，メモ化機構を中断＋再開させる機能を取り入れた場合でもそれぞれ 6.5%，17.3%となり，メモ化による高速性をほとんど損なうことなく省エネルギー化を実現できることが分かった．

今後の課題としては，本稿で用いた SPEC CPU95 や GENEsYs 以外の様々なベンチマークプログラムを用いたシミュレーションを通じて，より汎用的な閾値パラメータ，およびより効果的な動的アルゴリズムを検討していく予定である．特に，101.tomcatv や 102.swim のような特徴を持つプログラムに対していかに無駄な消費エネルギーを削減するかを検討する必要がある．

また，本稿では比較的単純なアーキテクチャを想定して評価を行ったが，より複雑な環境を想定したシミュレーションを行うとともに，命令レベル並列性に基づく高速化手法とメモ化とを組み合わせた手法を探ってゆくことも今後の課題である．

謝辞 本研究の一部は，文部科学省科学研究費補助金（萌芽研究 18650005，若手研究 (B) 19700041），（財）栢森情報科学振興財団研究助成金，（財）堀情報科学振興財団研究助成金による．

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. of Parallel and Distributed Computing and Networks*, pp.245–250 (2007).
2) Norvig, P.: *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann (1992).
3) Yang, J. and Gupta, R.: Energy-Efficient Load and Store Reuse, *Intl. Symp. on Low Power Electronics and Design*, pp.72–75 (2001).
4) Brooks, D., Tiwari, V. and Martonosi, M.: Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, *Proc. of the 27th Annual Intl. Symp. on Computer Architecture*, pp.83–94 (2000).
5) 島崎裕介，池内康樹，津邑公暁，中島 浩，松尾啓志，中島康彦：自動メモ化プロセッサの消費エネルギー評価，情報科学技術レターズ (FIT2007), pp.51–54 (2007).
6) Wilton, S.J. and Jouppi, N.P.: An Enhanced Access and Cycle Time Model for On-Chip Caches, Technical report, DEC Western Research Laboratory (1993).
7) Powell, M., Yang, S.-H., Falsafi, B., Roy, K. and Vijaykumar, T. N.: Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories, *Proc. of Intl. Symp. on Low Power Electronics and Design (ISLPED)*, pp.90–95 (2000).
8) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).