スクラッチパッドメモリを考慮したコード配置最適化による組込みシステムの低消費エネルギー化

石飛, 百合子
九州大学大学院システム情報科学府

石原, 亨
九州大学システムLSI 研究センター

安浦, 寛人
九州大学大学院システム情報科学研究院

出版情報：IPSJ Symposium Series, pp.85-90, 2007-08-29. 情報処理学会システムLSI設計技術研究会
バージョン：
権利関係：

スクラッチパッドメモリを考慮したコード配置最適化による組込みシステムの低消費エネルギー化

石飛百合子[1]　　石原亨[2]　　安浦寛人[3]

[1] 九州大学　大学院　システム情報科学府
[2] 九州大学　システム LSI 研究センター
[3] 九州大学　大学院　システム情報科学研究院

本稿では CPU コア，オンチップメモリおよびオフチップメモリを含む総消費エネルギーを削減するコード配置アルゴリズムの提案を行う．提案するアルゴリズムでは，cacheable 領域および scratchpad 領域を含む non-cacheable 領域へのコード配置を一括して行い，消費エネルギーを最小にするコード配置を求める．商用のプロセッサおよび SDRAM を用いた評価実験の結果，従来手法に比べ 23%の消費エネルギー削減効果を確認した．

Code Placement for Reducing the Energy Consumption of Embedded Processor with Scratchpad and Cache Memories

Yuriko Ishitobi[1]　　Tohru Ishihara[2]　　Hiroto Yasuura[3]

[1] Graduate School of Inf. Sci. & EE, Kyushu University
[2] System LSI Research Center, Kyushu University
[3] Faculty of Inf. Sci. & EE, Kyushu University

This paper proposes a code placement algorithm for reducing the total energy consumption of embedded processor systems including a CPU core, on-chip and off-chip memories. Our algorithm simultaneously finds code layouts for a cacheable region, a scratchpad region, and the other non-cacheable region of the address space so as to minimize the total energy consumption of the processor system. Experiments using a commercial embedded processor and an off-chip SDRAM demonstrate that our algorithm reduces the energy consumption of the processor system by 23% without any performance degradation compared to the best result achieved by the conventional approach

1　はじめに

近年マイクロプロセッサの消費エネルギーにおいて，オンチップメモリの消費エネルギーの割合が支配的になってきている．ARM920T^{TM} マイクロプロセッサでは消費電力の 43%，低消費電力プロセッサである StrongARM SA-110 においても 27%の電力がキャッシュメモリで消費されているとの報告がある [1–3]．キャッシュメモリの消費エネルギーについて，オフチップメモリの消費エネルギーとトレードオフの関係を考慮したキャッシュメモリサイズの最適化手法が多く提案されている [4–8]．これらの手法は，キャッシュメモリのサイズが大きくなると，アクセスあたりに必要なエネルギーは増加する一方でキャッシュミス数は減少するため，オフチップメモリでの消費エネルギーが減少する事実を利用している．キャッシュメモリのサイズを最適化することで，消費エネルギーを最適化することは可能である．しかし組込みシステム開発では，市販のマイクロプロセッサを用いてシステムを構築する場合が多く，キャッシュメモリサイズの選択肢が限られている状況が多い．本稿では，キャッシュメモリサイズが決定している場合でも適用可能な低消費エネルギー化手法であるコード配置手法に着目し，従来のコード配置手法を拡張した手法を提案する．

提案する手法は CPU ロジック部，キャッシュメモリ (以下キャッシュ)，SPM(Scratchpad Memory) およびオフチップメモリを含んだ組込みシステムの総消費エネルギーの削減を行うコード配置手法である．提案手法は，キャッシュを使用したアクセスを行う領域 (以下 cacheable 領域) とキャッシュを使用せずにアクセスを行う領域 (以下 non-cacheable 領域) の考慮を行う．non-cacheable 領域には，SPM に割り当てられたアドレス領域 (以下 scratchpad 領域) およびオフチップメモリに直接アクセスする領域の 2

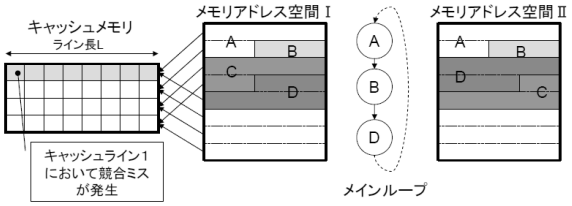

図 1: コード配置例

種類を想定している．non-cacheale 領域の存在を考慮した配置手法を用いることで，従来手法よりも低消費エネルギーなコード配置を実現する．本稿の構成を以下に示す．2 章にて既存の手法の紹介と提案手法の説明を行い，3 章で実行時消費エネルギーが最小となるコード配置を求めるアルゴリズムを示す．4 章で提案手法を商用のプロセッサとメモリに対し適用した際の消費エネルギーの評価を行い，5 章ではまとめを行う．

2 既存の研究と提案手法

2.1 既存研究

2.1.1 競合ミスを避けるコード配置

キャッシュのサイズを変更することなく，キャッシュおよびオフチップメモリの消費エネルギーを削減する手法に，競合ミスを避けるコード配置手法がある [9–14]．既存のコード配置手法では，関数，基本ブロックおよびデータのメモリアドレス空間内での配置をキャッシュの競合ミス数が最小となるよう最適化を行っている．

コード配置の基本となる考えを，サイズ $C(=2^m$word), ライン長 L のダイレクトマップ方式のキャッシュを例に説明する．キャッシュミス時のオフチップメモリとのデータの転送は，ライン長 L 単位で行う．アドレス空間内のあるデータ (アドレス M) のキャッシュ内での配置エントリは $\lfloor M/L \rfloor mod C/L$ により求められる．メモリアドレス M_i と M_j を持つ 2 つのデータが競合する条件は (1) 式により表される．

$$(\left\lfloor \frac{M_i}{L} \right\rfloor - \left\lfloor \frac{M_j}{L} \right\rfloor) mod \frac{C}{L} \qquad (1)$$

図 2.1.1 にコード配置手法を適用する例を示す．関数 A, B, C および D がメモリアドレス空間 I のように配置されている．関数へのアクセスが A, B, D の順にループすると，A と D の間で競合ミスが発生する．関数 C と D の配置をメモリアドレス空間 II のように入れ替えると，A と D の間で発生していた競合ミスは発生しない．既存のコード配置決定アルゴリズムでは，競合ミス数が最小になるようにコード配置を決定している．

2.1.2 SPM への静的なコード配置

キャッシュがデータ配置が動作時にハードウェアにより自動的に行われるメモリであることに対し，SPM はアドレスが割り振られており，プログラマおよびコンパイラのアドレス指示によりデータ配置が行われるメモリである．SPM はハードウェアによるデータ転送のサポートが行われないため，キャッシュで必要なタグアドレス比較等の処理が必要なく，アクセス時消費エネルギーが同容量のキャッシュと比較して小さい．これを利用し，コンパイル時に scratchpad 領域へのコード配置を最適化し，オンチップメモリの消費エネルギーを削減する研究が行われている．Banakar らは実験で，SPM のみを使用した場合でキャッシュのみを使用した場合の平均 40%の消費エネルギー削減を確認している [15]．Steinke らは scratchpad 領域への関数，基本ブロックおよびデータの配置問題を knapsack 問題と定義し，配置された関数，基本ブロックおよびデータへの総アクセス回数を最大化するように，配置する関数，基本ブロックおよびデータの選択を行うアルゴリズムを提案している [16].

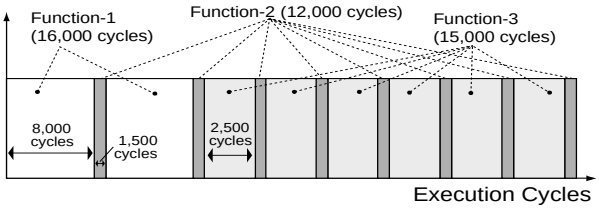

図 2: SPM 活用例

2.1.3 キャッシュバイパス

アクセスの時間的局所性が低いコードをキャッシュバイパスすることで，キャッシュにおける消費エネルギーを削減することができる．加えてバイパスを行うことでキャッシュの有効活用が可能となり，結果として総キャッシュミス数を削減することができる．Johnson らは，実行時にアクセスの時間的局所性の検知を行うハードウェア機構を提案している [17]．データのアクセスパターンを観察し，アクセスの時間的局所性が低いデータはキャッシュバイパスバッファを用いてバイパスを行う．Rivers らはストリームデータのバイパスを行う NTS(non-temporal stream) キャッシュを提案している [18]．提案しているキャッシュ構成は，ダイレクトマップ方式の L1 キャッシュに加えストリームデータ転送用にフルアソシアティブキャッシュを加えた構成である．データを配置するキャッシュの選択は，L2 キャッシュで行うものとしている．

我々の提案手法においてもキャッシュバイパスによりキャッシュミス数を削減する手法を活用するが，提案手法ではキャッシュバイパスに専用のバッファ等を使用しない．キャッシュバイパスは時間的局所性の低いデータをメモリアドレス空間内の non-cacheable 領域にコードを配置することで実現する．non-cacheable 領域を使用することでキャッシュバイパスを行い，キャッシュミス数を削減し総消費エネルギーの改善を行う手法は本稿独自のものである．

2.2 提案手法

従来の scratchpad 領域へのコード配置決定アルゴリズムでは，キャッシュと SPM を両方搭載したプロセッサを想定していない．キャッシュと SPM が混載されたプロセッサを想定する場合，SPM を利用することによるキャッシュミス削減についても考慮する必要がある．例として 2KB のキャッシュおよび SPM を搭載したプロセッサを仮定する．関数 1，関数 2 および関数 3 がすべて同サイズで 2KB であるとする．図 2 に示すように，前半は関数 1 と関数 2 が交互に実行され，後半は関数 2 と関数 3 が交互に実行されるアドレストレースが得られたとする．従来手法を用いると，アクセス頻度は関数 1 が最も高くなるため，

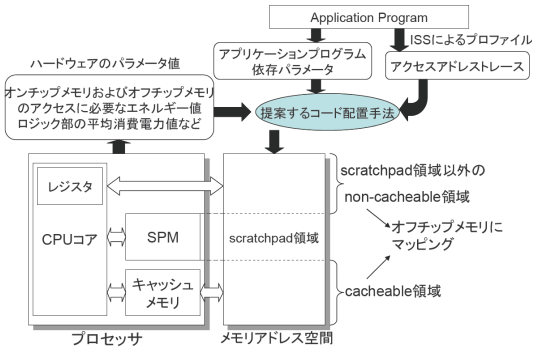

図 3: コード配置手法のフロー

scratchpad 領域には関数 1 が配置される．しかし，関数 1 を scratchpad 領域に配置する場合，関数 2 と関数 3 の実行が切り替わる際に多くのキャッシュミスを発生する．関数 2 が scratchpad 領域へ配置された場合，キャッシュミス数は効率よく削減できるため，キャッシュおよびオフチップメモリの消費エネルギー削減される．加えて SPM の領域が足りない場合でも，その他の non-cacheable 領域を使用することを考慮することができる．

提案するコード配置手法は，cacheable 領域および，scratchpad 領域を含む non-cacheable 領域へのコード配置を総消費エネルギーが最小になるように決定する．提案手法では scratchpad 領域を利用することによるキャッシュミス削減効果の考慮も行う．キャッシュミス数は cacheable 領域の配置により異なってくるため，scratchpad 領域への最適な配置は cacheable 領域の配置を無視して決めることはできない．提案手法ではキャッシュミス数を考慮した消費エネルギー見積り関数によりコード配置の評価を行うため，キャッシュミス削減効果を考慮した scratchpad 領域への配置が実現できる．SPM に加えて，提案手法では non-cacheable 領域としてキャッシュを使用せず直接オフチップメモリへアクセスする領域を使用する．SPM はサイズに制限があるため，静的にコード配置を行う際はサイズ以上のコードを配置することはできない．ストリームデータのようにサイズが大きく，アクセスの時間的局所性が低いデータは scratchpad 領域以外の non-cacheable 領域に配置し，キャッシュバイパスを行うことが可能である．

コード配置は図 3 に示すフローで行う．アプリケーションに依存した情報，命令セットシミュレータ (ISS) によって得たアプリケーションプログラムのアクセスアドレストレースおよび測定して得たハードウェアのパラメータをコード配置探索を行うプログラムに入力し，最適なコード配置を求める．求めたコード配置はコンパイラのプリプロセスにより，メモリアドレス空間に割当てを行う．

3 コード配置決定アルゴリズム

3.1 問題定義

3.1.1 コード配置問題

総消費エネルギーを最小にするようなメモリアドレス空間内のメモリオブジェクトの配置を求める問題をコード配置決定問題と定義する．メモリオブジェクトは関数，グローバル変数および定数データであり，配置換えはメモリオブジェクト単位で実行する．メモリオブジェクトは cacheable 領域，scratchpad 領域および他の non-cacheable 領域のいずれかに配置される．cacheable 領域内は配置された各メモリオブジェクトの位置がキャッシュミス数に影響を与えるため，領域内でのメモリオブジェクト配置位置も消費エネルギーを最小にするよう決定する．non-cacheable 領域では，領域内でのメモリオブジェクトの配置位置は消費エネルギーへの影響は無いものとし，配置するメモリオブジェクトの選択のみを行う．

3.1.2 目的関数

コード配置問題の目的関数を (2) 式に示す．提案するアルゴリズムでは，メモリオブジェクトの位置を入れ替えることで TE_{all} を最小化することを目的としている．TE_{all} は CPU とオフチップメモリの総消費エネルギーを表す．E_{cacher} および E_{cachew} はキャッシュのリードおよびライトアクセスあたりに必要なエネルギーを表し，E_{miss} および E_{dmiss} はキャッシュミスおよび書戻しを伴うキャッシュミスあたりに消費するエネルギーを表す．N_{cacher}，N_{cachew}，N_{miss} および N_{dmiss} は各々キャッシュリード数，キャッシュライト数，キャッシュミス数および書戻しを伴うキャッシュミス数を表す．E_{spmr} および E_{spmw} は SPM へのリードおよびライトアクセスあたりの消費エネルギーを表し，N_{spmr} および N_{spmw} は SPM へのリードおよびライトアクセス数を表す．P_{logic} および t_{all} はそれぞれロジック部の平均消費電力と総実行時間を示す．E_{offr} および E_{offw} はオフチップメモリへのリードおよびライトアクセスあたりの消費エネルギーを示し，P_{off} はオフチップメモリが静的に消費する電力値を示す．N_{offbr} および N_{offbw} はバースト転送でのオフチップリードおよびライトアクセス数を示し，N_{offsr} および N_{offsw} は non-cacheable 領域アクセス時のシングル転送におけるオフチップリードおよびライトアクセス数を示す．CN_{inst}，CN_{offbr}，CN_{offbw}，CN_{offsr} および CN_{offsw} はそれぞれ 1 命令実行，オフチップバーストリード，オフチップバーストライト，オフチップシングルリードおよびオフチップシングルライトに必要な実行サイクル数の平均値を示す．N_{inst} は実行命令数を示し，CT はクロックサイクル時間を示す．

$$
\begin{aligned}
TE_{total} =& TE_{cache} + TE_{spm} + TE_{logic} + TE_{off} \qquad (2)\\
TE_{cache} =& N_{cacher} \cdot E_{cacher} + N_{cachew} \cdot E_{cachew}\\
&+ N_{miss} \cdot E_{miss} + N_{dmiss} \cdot E_{dmiss}\\
TE_{spm} =& N_{spmr} \cdot E_{spmr} + N_{spmw} \cdot E_{spmw}\\
TE_{logic} =& P_{logic} \cdot t_{all}\\
TE_{off} =& (N_{offbr} + N_{offsr}) E_{offr} + (N_{offbw}\\
&+ N_{offsw}) E_{offw} + P_{off} \cdot t_{all}\\
t_{all} =& CT(N_{inst} \cdot CN_{inst} + N_{offbr} \cdot CN_{offbr}\\
&+ N_{offsr} \cdot CN_{offsr} + N_{offbw} \cdot CN_{offbw}\\
&+ N_{offsw} \cdot CN_{offsw}) \qquad (3)
\end{aligned}
$$

E_{cacher}，E_{cachew}，E_{spmr}，E_{spmw}，E_{miss}，E_{dmiss}，E_{offbr}，E_{offbw}，E_{offsr}，E_{offsw}，P_{logic}，P_{off}，CT，CN_{inst}，CN_{offbr}，CN_{offbw}，CN_{offsr} および CN_{offsw} はハードウェアに依存した値であり，コード配置に依存しない．これらの値は回路シミュレータ，ゲートレベルシミュレータおよび電力解析ツールを用いて測定を行った．N_{cacher}，N_{cachew}，N_{miss}，N_{dmiss}，N_{spmr}，N_{spmw}，N_{inst}，N_{offbr}，N_{offbw}，N_{offsr} および N_{offsw} はコード配置に依存した値であり，アクセスアドレストレース (TR) を用いて，キャッシュシミュレータにより値を求めた．

3.2 コード配置探索プログラム

Input: TR,F,SPS,E_{cacher},E_{cachew},E_{spmr},
E_{spmw},E_{offr},E_{offw},E_{miss},E_{dmiss}
P_{logic},P_{off},CT,CN_{inst},CN_{offbr},CN_{offbw}
CN_{offsr} *and* CN_{offsw}
Output: *location of memory objects* in optimized object code
$t_{const} = t_{all}$; TE_{min}=t_{min}=infinity;
$SPS_{rest} = SPS$;
repert
  for($t = 0; t < |F|; t++$)**do**
    $p = F[t]$; $BEST_{locate} = p$;
    $TE_{org} = TE_{min}$; $t_{org} = t_{min}$;
    if($SPS_{rest} \leq SIZE[p]$))**then**
      put Place memory object p to scratchpad region;
      $SPS_{rest} = SPS_{rest} - SIZE[p]$
      next;
    end if
    for each $p' \in F$ **and** $p' \neq p$ **do**
      if(p' resides in scratchpad &&
        $SIZE[p] - SIZE[p'] \leq SPS_{rest}$)**then**
        Evict p' from scratchpad region to cacheable region and place p to scratchpad region;
        Update TR according to new location;
        Calculate TE_{all} and t_{all};
        if($TE_{all} < TE_{min}$)
          $TE_{min} = TE_{all}$; $t_{min} = t_{all}$;
          save present placement of p as $BEST_{locate}$;
        end if
      end if
      Insert memory object p in place of p';
      Update TR according to new location;
      Calculate TE_{all} and t_{all};
      if($TE_{all} < TE_{min}$)
        $TE_{min} = TE_{all}$;$t_{min} = t_{all}$;
        save present placement of p as $BEST_{locate}$;
      end if
    end for
    Place p to non-cacheable region;
    Update TR according to new locations;
    Calculate TE_{all} and t_{all};
    if($TE_{all} < TE_{min}$)**then**
      $TE_{min} = TE_{all}$; $t_{min} = t_{all}$;
      save present placement of p as $BEST_{locate}$;
    end if
    if($t_{min} \leq t_{org}$)**then**
      Fix the location of p as the $BEST_{locate}$;
    end if
  end for
until TE_{min} **stops decreasing**
Output *locations of memory objects*

上記はコード配置を探索するために用いたプログラムの擬似コードである．初期コード配置は，メモリオブジェクトがすべて cacheable 領域に配置されている状態である．メモリオブジェクトをアクセス頻度の降順に並べたリスト F の先頭から 1 つずつメモリオブジェクトを選択し，配置する領域および cacheable 領域であれば領域内での配置位置を決定する．擬似コード中の SPS は SPM のサイズ，SPS_{rest} はコードが配置されていない scratchpad 領域のサイズを示す．TE_{min}，t_{min} および $BEST_{locate}$ は最小なエネルギー値となったときのエネルギー値，実行時間および配置を一時的に保持する変数である．t_{const} は初期配置での実行時間を示す．TE_{all} および t_{all} の計算には (2) 式および (3) 式を用いる．

F から選ばれたメモリオブジェクト (以下 p) は，最初に scratchpad 領域に配置可能かのチェックされる．scratchpad 領域の空き領域が p のサイズ以上ならば配置可能であるので，scratchpad 領域へ配置する．scratchpad 領域に空きがない場合は，他のメモリオブジェクト (以下 p':$p' \neq p$) を対象として配置位置の変更を行う．p' が scratchpad 領域に配置されている場合は，p' を追い出して cacheable 領域に配置し，scratchpad 領域の空きスペースが p のサイズ以上であれば配置を実施し消費エネルギー TE_{all} と実行時間 t_{all} を見積もる．p' が cachable 領域に配置されていた場合は，配置されている位置に p の挿入を行い，同様に見積りを行う．配置変更後に見積りを行った結果として TE_{all} がその時点までの最小エネルギー値 TE_{min} よりも低い場合は，配置を $BEST_{locate}$ として記憶し，TE_{min} および t_{min} を更新する．すべての p' を対象として配置位置の変更とエネルギーの見積りを行った後，p を他の non-cacheable 領域に配置した際の TE_{all} および t_{all} の見積りを行い，その時点までの TE_{min} よりも TE_{all} が低ければ，$BEST_{locate}$ をその他の non-cacheable 領域に更新し TE_{min} および t_{min} を更新する．最終的に配置変更後の実行時間 t_{min} が初期配置の実行時間 t_{const} より長くならないときのみ，配置を $BEST_{locate}$ に変更する．

p を選んで配置を求める試行を繰り返し，リスト F の最後尾に到達すると，F の先頭に戻りさらに配置換えを続ける．同じ p に関する配置を求める場合でも，n 回目と m 回目の試行では，他のメモリオブジェクトの配置状況が異なる可能性があるため，リスト F の試行は複数回行われる．配置換えはリスト F の先頭のメモリオブジェクトに対しての配置換え前の消費エネルギーと，すべての F の配置換え終了後の消費エネルギー結果が変わらなくなるまで続ける．

4 評価実験

提案するコード配置手法を用いた場合の消費エネルギー削減効果と，実行時間の評価を行う実験を行った．

4.1 ターゲットシステム

実験にはプロセッサにルネサステクノロジ社製の SH3-DSP，オフチップメモリに Micron の SDRAM DDR-II を用いた．実験に用いた SH3-DSP は命令およびデータ混在のユニファイドキャッシュおよび XY メモリと呼ばれる SPM を搭載している．SH3-DSP プロセッサは 0.18 μm プロセステクノロジのスタンダートセルライブラリおよび SRAM モジュールを用いて論理合成を行っている．

ロジック部の平均消費電力 P_{logic} は，CADENCE 社製の $NC-Verilog^{TM}$ によるゲートレベルシミュレーションで得たトランジスタのトグル情報を用いて，SYNOPSYS 社製の $PowerCompiler^{TM}$ により算出を行った．SRAM モジュールのアクセスあたりの消費エネルギーの算出には，SYNOPSYS 社製の回路シミュレータである $NanoSim$ を使用した．SRAM モジュールのアクセスあたりの消費エネルギーを表 1 に示す。各モジュールのリードアクセス時間が 4KB の SPM のリードアクセス時間である 954[p sec] 以下に統一されるように電源電圧値を選択した．

文献 [20] において Panwer らは，すべての命令フェッ

表 1: オンチップメモリアクセスエネルギー

Memory size	Supply Voltage	Single-way access energy	Full-way access energy
8KB4way 128-set cache	1.70V	420.308 pJ	2209.34 pJ
16KB4way 256-set cache	1.80V	573.696 pJ	2946.67 pJ
4KB SPM	1.75V	520.896 pJ	
8KB SPM	2.25V	1381.440 pJ	
16KB SPM	2.50V	2382.240 pJ	

表 2: ベンチマークプログラム

Benchmark	Code size	# of Memory objects
compress	36,778 byte	130
JPEG encoder	138,334 byte	427
MPEG2 encoder	133,998 byte	477

チにおいてキャッシュのタグへのアクセスおよびタグ比較が必要なわけではないことを示している．アクセスアドレストレース上で命令 i の直後に命令 j が実行される場合を考える．i が分岐命令でなくかつ，キャッシュラインの最後尾に配置されていないのであれば，j は i と同様のキャッシュラインに配置されていることが分かるため，j はタグの確認を行う必要がなくなる [20,21]．この場合，キャッシュアクセス時は j が配置されていると分かっているウェイのみ活性化を行えばよいことになる．1つのウェイのみをアクセスするために必要なエネルギーを *single-way access energy* とし，表 1 の 3 列目に示す．i が分岐命令あるいは，キャッシュラインの最後尾に配置されている場合はタグ確認の作業が必要になり，ウェイへのアクセスもすべてのウェイに対して行う必要が出てくる．この場合のキャッシュメモリへのアクセスエネルギーを *full-way access energy* とし，表 1 の 4 列目に示す．

4.2 ベンチマークプログラム

ベンチマークプログラムには，compress，JPEG エンコーダおよび MPEG2 エンコーダを利用した．各ベンチマークプログラムはすべて“ -O3 ”オプションを用いてコンパイルしている．SH3-DSP 用の GMU コンパイラおよびデバッカを使用し，初期配置におけるアドレストレースを生成した．アドレストレースは main 関数の開始から 100 万命令分を使用している．表 2 に各ベンチマークプログラムのコードサイズと，メモリオブジェクト数を示す．

4.3 実験結果

実験結果を図 4，5 および 6 に示す．各グラフにおいて，左側が 16KB の 4 ウェイセットアソシアティブキャッシュを使用した結果，右側が 8KB の 4 ウェイセットアソシアティブキャッシュを使用した結果である．SPM のサイズは 4KB，8KB および 16KB を用いている．消費エネルギーは棒グラフの値で示され，実行サイクル時間は折れ線グラフの値で示される．実験は以下の 6 つの配置の比較を行った．

ORG: 初期配置

CHE: cacheable 領域内でのメモリオブジェクトの配置換えのみを行い，競合ミスを避ける配置とした場合

SPM_G: scratchpad 領域への配置のみ，欲張り法を用いて行った場合

SPM_O: scratchpad 領域への配置のみ，knapsak 問題を厳密に解き SPM へのアクセス数が最大になるように配置した場合

CBN: **SPM_G** で用いた方法に従って scratchpad 領域への配置を行った後，**CHE** で用いた手法で，cacheable 領域の配置を行った場合

OUR: 提案手法を用いて配置を行った場合

図 4，図 5 および図 6 おいて，提案手法である **OUR** が常に最小エネルギー値となっている．大きいサイズの SPM を用いた場合，既存手法 **SPM_G** および **CBN** の消費エネルギーが **CHE** よりも大きくなることが分かる．これは SPM へのアクセスあたりに必要なエネルギーが，キャッシュの *single-way access energy* よりも大きくなるためである．一方で **CHE** は **SPM_G** および **CSN** と比較して，多くの実行サイクルが必要であることが分かる．**OUR** では実行サイクルと消費エネルギーに関して，多くの場合他の配置手法よりもよい結果となっている．scratchpad 領域へのコード配置を SPM へのアクセス回数を最大にする knapsack 問題とし，欲張り法によって近似解を得る **SPM_G** と厳密に解いた場合の **SPM_O** では大きな差は確認されなかった．厳密に解くことで SPM へのアクセス数は最大になる場合でも，キャッシュミス数の変化の影響により，総消費エネルギーが増加するケースも存在した．

compress を用いた実験では，すべてのメモリ構成において提案手法の有効性が確認できた．8KB のキャッシュおよび 16KB の SPM を用いた場合では，**CBN** に比べ **OUR** は 23%の消費エネルギー改善効果と，5%の実行サイクル数削減が確認できた．8KB のキャッシュおよび 4KB の SPM を使用した場合においても，10%の消費エネルギー削減と 6%の実行サイクル数削減が可能であった．

JPEG encoder における実験では，提案手法は 16KB のキャッシュを用いた場合に有効であった．16KB のキャッシュおよび 16KB の SPM を用いたとき，**CBN** に比べ，**OUR** は 18%の消費エネルギー削減と，1%の実行サイクル数削減を確認した．

MPEG2 encoder を用いた実験では，他のベンチマークプログラムの結果に比べ，大きな効果は得られなかった．この原因として，*MPEG2 encoder* ではアクセスが非常に少ないコードに集中しており，アクセスのあるコードの大部分がキャッシュおよび SPM 上に配置され，キャッシュミスの発生が少なかったためと考えられる．しかし，注

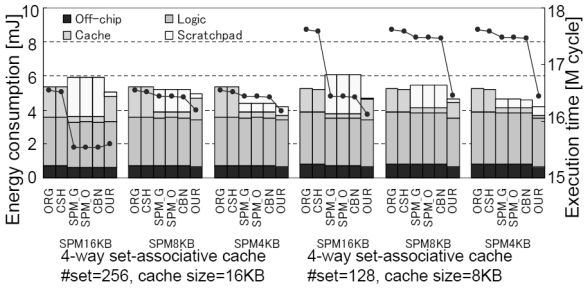

図 4: COMPRESS 実験結果

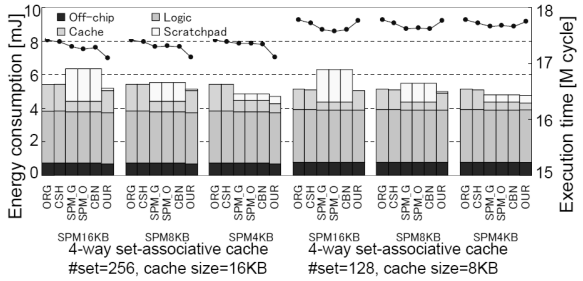

図 5: JPEG 実験結果

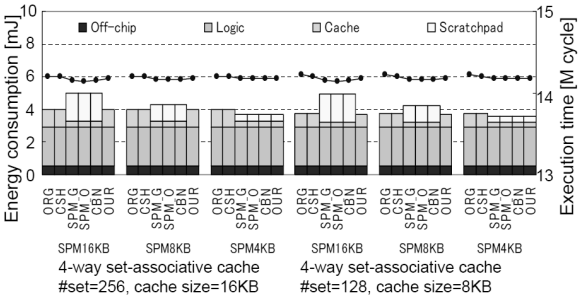

図 6: MPEG2 実験結果

目すべきは提案手法がどのケースにおいても，0.1%未満のパフォーマンスのロスのみで最もよい消費エネルギー値を得られる点である．

5 まとめ

本稿では，組込みシステムの消費エネルギーを削減するコード配置手法の提案を行った．提案本稿では，組込みシステムの消費エネルギーを削減するコード配置手法の提案を行った．提案手法ではコード配置に non-cacheable 領域を考慮に入れ，キャッシュヒット率を向上させることにより，システムの総消費エネルギーの低減を実現した．コード配置を求めるアルゴリズムにおいて総消費エネルギーを最小化する目的で，メモリアドレス空間内の cacheable 領域，scratchpad 領域および non-cacheable 領域の 3 領域への配置を同時に考慮した．市販のプロセッサおよびオフチップ SDRAM を使用した実験において，提案手法が実行時間の増加なしに消費エネルギーの削減が可能であることを確認した．

6 謝辞

本研究は東京大学大規模集積システム設計教育研究センターを通し，株式会社ルネサステクノロジ，ローム株式会社，凸版印刷株式会社，シノプシス株式会社，日本ケイデンス株式会社の協力で行われたものである．本研究の一部は，科学技術振興機構 (JST) の戦略的創造研究推進事業 (CREST) によるものである．

参考文献

[1] S.Seger, “ Low Power Design Techniques for Microprocessors ”, ISSCC Tutorial note, February 2001.

[2] ARM Ltd., “ ARM Processor Core Overview ” , *http://www.arm.com/products/CPUs/*

[3] J. Montanaro et al., “ A 160 MHz, 32b 0.5W CMOS RISC Microprocessor ”, In Proc. of ISSCC, February 1996.

[4] C. Su and A. Despain, “ Cache Design Trade-offs for Power and Performance Optimization: A Case Study ”, In Proc. of ISLPED, pp.63-68, August 1995.

[5] P. Hicks, M. Walnock, and R. M. Owens, “ Analysis of Power Consumption in Memory Hierarchies ”, In Proc. of ISLPED, pp.239-242, August 1997.

[6] Y. Li, J. Henkei, “ A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems ”, In Proc. of DAC, pp.188-193, June, 1998.

[7] W. T. Shine, and C. Chacrabarti, “ Memory Exploration for Low Power, Embedded Systems ”, In Proc. of DAC, pp.140-145, June, 1999.

[8] A. Malik, B. Moyer and D. Cermak, “ A Low Power Unified Cache Architecture Providing Power and Performance Flexibility ”, In Proc. of ISLPED, pp.241-243, July 2000.

[9] S. McFarling, “ Program Optimization for Instruction Caches ”, In Proc. of Int’l Conference on Architecture Support for Programming Languages and Operation Systems, pp.183-191, April 1989.

[10] W. W. Hwu and P. P. Chang, “ Achieving High Instruction Cache Performance with an Optimizing Compiler ”, In Proc. of ISCA, pp.242-251, May 1989.

[11] H. Tomiyama and H. Yasuura, “ Optimal Code Placement of Embedded Software for Instruction Cachea ”, In Proc. of European Design and Test Conference, pp.96-101, March 1996.

[12] P. Panda, N. Dutt, and A. Nicolau, “ Memory Organization for Improved Data Cache Performance in Embedded Processors ”, In Proc. of ISSS, pp.90-95, November 1996.

[13] A. H. Hashemi, D. R. Kaeli, and B. Calder, “ Efficient Procedure Mapping Using Cache Line Coloring ”, In Proc. of Programming Language Design and Implementation, pp.171-182, June 1997.

[14] S.Ghosh, M. Martonosi, and S. Malik, “ Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior ”, In Proc. of ACM Trans on Programming Languages and Systems, vol.21, no.4, pp.703-746, July 1999.

[15] R. Banakar, S. Steinke, B. S. Lee, M. Balakrishnan, and P. Marwedel, “ Scratchpad Memory : A Design Alternative for Cache On-Chip Memory in Embedded Systems ”, In Proc. of CODES, pp.73-78, May 2002.

[16] S. Steinke, L. Wehmeyer, B. Lee, P. Marwedel, “ Assigning Program and Data Objects to Scratchpad for Energy Reduction ”, In Proc. of DATE, pp.409-415, March 2002.

[17] T. L. Johnson, M. C. Merten, and W. W. Hwu, “ Run-Time Spatial Locality Detection and Optimization ”, In Proc. of the 30^{th} Int’l Symposium on Microarchitecture, pp.57-64, December 1997.

[18] J. A. Rivers and E. S. Davidson, “ Reducing Conflicts n Direct-Mapped Caches with a Temporality-Based Design ”, In Proc. of the 25^{th} Int’l Conference on Parallel Processing, pp.154-163, August 1996.

[19] “ The Micron System Power Calculator ” , *http://www.micron.com/support/designsupport/tools/powercalc/powercalc*

[20] R. Panwar, and D. Rennels, “ Reducing the Frequency of Tag Compares for Low Power I-Cache Design ”, In Proc. of ISLPED, pp.57-62, August 1995.

[21] M. Mullar, “ Power Efficiency & Low Cost: The ARM6 Family ”, In Proc. of Hot Chips IV, August 1992.