社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SLDM－139 (4)
2009－ＥＭＢ－ 12 (4)
2009／3／5

シングルサイクルアクセス可能な二階層キャッシュアーキテクチャ

山口誠一朗† 石原 亨†† 安浦 寛人†††

† 九州大学大学院システム情報科学府
†† 九州大学システム LSI 研究センター
††† 九州大学

あらまし 組込みプロセッサのメモリサブシステムの消費エネルギーを削減するために，プロセッサコアと L1 キャッシュメモリ（以下，キャッシュメモリを単にキャッシュという）の間に小容量の L0 キャッシュを配置する技術が広く利用されている．L0 キャッシュは小容量であるためキャッシュヒットすれば消費エネルギーを削減できる．しかし，キャッシュミスした場合，L1 キャッシュへアクセスするために最低 1 サイクル必要となり，プロセッサの性能低下を引き起こす．この問題を解決するため，シングルサイクルアクセス可能な二階層キャッシュ（STC: Single-cycle-accessible Two-level Cache）アーキテクチャを本稿で提案する．STC アーキテクチャでは，プロセッサコアはシングルサイクルで小容量キャッシュまたは L1 キャッシュにアクセスできる．さらに，STC アーキテクチャを有効活用するコンパイル技術も本稿で提案する．ベンチマークを用いた実験では，L0 キャッシュを用いたアプローチと比較して，メモリサブシステムの消費エネルギーを最大で 64%，平均で 41%削減できた．
キーワード 組込みシステム，キャッシュメモリ，消費エネルギー

Single-Cycle-Accessible Two-Level Cache Architecture

Seiichiro YAMAGUCHI†, Tohru ISHIHARA††, and Hiroto YASUURA†††

† Graduate School of Information Science and Electrical Engineering, Kyushu University
†† System LSI Research Center, Kyushu University
††† Kyushu University

Abstract A small L0-cache located between an MPU core and an L1-cache is widely used in embedded processors for reducing the energy consumption of memory subsystems. Since the L0-cache is small, if there is a hit, the energy consumption will be reduced. On the other hand, if there is a miss, at least one extra cycle is needed to access the L1-cache. This degrades the processor performance. Single-cycle-accessible Two-level Cache (STC) architecture proposed in this paper can resolve the problem in the conventional L0-cache based approach. Both a small L0 and a large L1 caches in our STC architecture can be accessed from an MPU core within a single cycle. A compilation technique for effectively utilizing the STC architecture is also presented in this paper. Experiments using several benchmark programs demonstrate that our approach reduces the energy consumption of memory subsystems by 64% in the best case and by 41% on an average without any performance degradation compared to the conventional L0-cache based approach.
Key words Embedded system, cache memory, energy consumption

1. はじめに

携帯電話に代表されるバッテリー駆動の組込みシステムのみならず，あらゆる組込みシステムにおいて消費エネルギーの削減は重要な課題である．また，多くの機能を一つのシステムで実現するために，組込みシステム向けプロセッサの性能向上も求められている．これらの要求を満たすためにキャッシュメモリ（以下，キャッシュという）がプロセッサには搭載されている．キャッシュを利用することで，大容量であるがアクセス時間が遅く，アクセスエネルギーが大きいオフチップメモリへのアクセス回数を削減している．しかし，キャッシュで消費されるエネルギーはプロセッサで消費されるエネルギーの 40%以上を占めている [1], [2]．したがって，キャッシュ，さらにはオフチップメモリを含めたメモリサブシステムで消費されるエネル

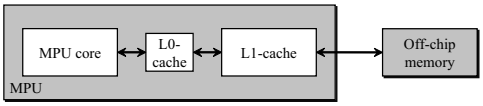

図 1　L0 キャッシュを搭載したメモリサブシステムの構成.

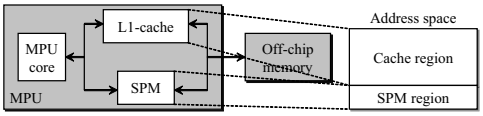

図 2　SPM を搭載したメモリサブシステムの構成およびアドレス空間の割当て.

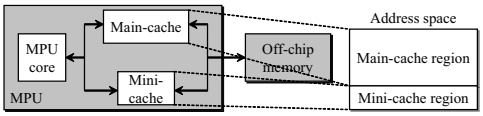

図 3　HPC アーキテクチャのメモリサブシステムの構成およびアドレス空間の割当て.

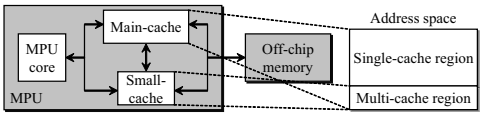

図 4　STC アーキテクチャのメモリサブシステムの構成およびアドレス空間の割当て.

ギーを削減することは重要である.

我々は文献 [3] で，メモリサブシステムで消費されるエネルギーを削減するためのキャッシュアーキテクチャ，シングルサイクルアクセス可能な二階層キャッシュ（STC: Single-cycle-accessible Two-level Cache）アーキテクチャを提案している．STC アーキテクチャは，小容量のキャッシュと通常サイズのキャッシュを二階層キャッシュとして搭載しながら，ともにシングルサイクルでアクセス可能なアーキテクチャである．アクセスエネルギーが小さい小容量のキャッシュへアクセスが集中すれば消費エネルギーを削減できる．本稿では，STC アーキテクチャについて説明し，それを有効活用するためのコード配置技術を提案する．これにより，小容量キャッシュへのアクセスを集中させ，結果としてメモリサブシステムの消費エネルギー削減を達成する.

2. 関 連 研 究

2.1 L0 キャッシュ

L0 キャッシュと呼ばれる小容量キャッシュをプロセッサコアと L1 キャッシュの間のメモリ階層に搭載する技術が多く提案されている [4]〜[8]．L0 キャッシュを搭載したメモリサブシステムの構成を図 1 に示す．L0 キャッシュは小容量であるため，アクセス当たりのエネルギーは L1 キャッシュよりも小さい．したがって，L0 キャッシュヒットすれば，アクセスエネルギーを削減できる．一方で，L0 キャッシュミスすれば，L1 キャッシュへアクセスするために少なくとも 1 サイクル必要となり，プロセッサの性能低下を引き起こす要因となる．L0 キャッシュを用いたメモリサブシステムでは，メモリアクセスの時間的局所性が高い場合にアクセスエネルギーの削減効果は大きい.

2.2 スクラッチパッドメモリ

L0 キャッシュを搭載したメモリサブシステムの問題点を解決する一つの方法として，ソフトウェア制御可能なスクラッチパッドメモリ（SPM: Scratch-Pad Memory）を L1 キャッシュと共に活用する方法がある．SPM を搭載したメモリサブシステムの構成およびアドレス空間の割当てを図 2 に示す．SPM と L1 キャッシュは同じメモリ階層にあるため，プロセッサコアはどちらのメモリへもシングルサイクルでアクセス可能である．アクセスアドレスの一部のビットからアドレス空間の領域を判定し，その結果により SPM と L1 キャッシュのどちらか片方のみがアクセスされる．アドレス空間の SPM 領域は静的に SPM に割り当てられており，SPM 領域のコード／データはシステム起動時に SPM へコピーされる．つまり，SPM は割り当てられたアドレスのコード／データを常に保持しており，アクセスミスは発生しない．また，SPM はキャッシュに必要不可欠なタグの読出し／比較を行う必要がないため，アクセスエネルギーが小さい．したがって，プログラマやコンパイラが SPM 領域に適切なコード／データを配置できればエネルギーの削減が見込める．コード／データの配置に関する研究は文献 [9]〜[12] などで行われている．さらには，SPM をオーバーレイする技術についても研究されている [13]〜[16].

2.3 Horizontally Partitioned Cache

Horizontally Partitioned Cache（HPC）アーキテクチャは同じメモリ階層に小容量のキャッシュ（Mini キャッシュ）と通常サイズのキャッシュ（Main キャッシュ）を持つ [17]〜[19]．HPC アーキテクチャのメモリサブシステムの構成およびアドレス空間の割当てを図 3 に示す．アドレス空間は二つの領域，Main キャッシュ領域および Mini キャッシュ領域に分割されており，それぞれのキャッシュに割り当てられている．SPM を搭載したメモリサブシステムと同様に，アクセスアドレスの一部のビットからアドレス空間の領域を判定し，その結果によりどちらか片方のキャッシュのみがアクセスされる．SPM を搭載したメモリサブシステムとの違いは，異なるアドレスのコード／データを Mini キャッシュは自動的に保持することができる点である.

3. STC アーキテクチャ

STC アーキテクチャは小容量のキャッシュ（Small キャッシュ）と通常サイズのキャッシュ（Main キャッシュ）を二階層のキャッシュとして搭載しながら，ともにシングルサイクルでアクセス可能なアーキテクチャである．STC アーキテクチャのメモリサブシステムの構成およびアドレス空間の割当てを図 4 に示す．STC アーキテクチャでは，HPC アーキテクチャと同様にアドレス空間は二つの領域，Single キャッシュ領域および Multi キャッシュ領域に分割されている．STC アーキテクチャと HPC アーキテクチャの違いは，Multi キャッシュ領域の割

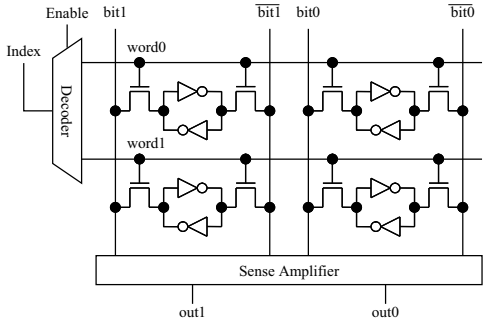

図 5　2 行× 2 列の SRAM アレイ.

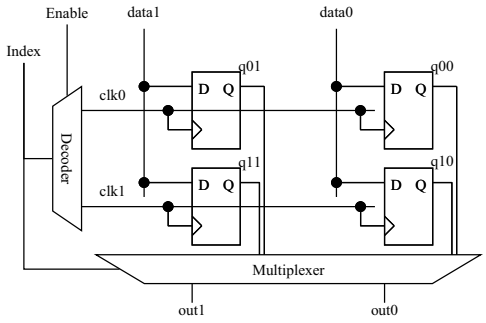

図 6　2 行× 2 列の FF アレイ.

当てである．Single キャッシュ領域に存在するコード／データは全て Main キャッシュを介してアクセスされる．これは HPC アーキテクチャにおいて，Main キャッシュ領域に存在するコード／データは全て Main キャッシュを介してアクセスされることと同じ動作である．一方，Multi キャッシュ領域に存在するコード／データをオフチップメモリからコピーする際は，優先的に Small キャッシュを上書き（データが更新されている場合は書き戻し操作も必要）する．STC アーキテクチャでは，このとき上書き（書き戻し）される対象である低優先度のラインを Main キャッシュに退避させる．退避先である Main キャッシュの低優先度のラインは Main キャッシュから追い出される．つまり，Multi キャッシュ領域は Main キャッシュおよび Small キャッシュの両方に割り当てられている．

SPM を搭載したメモリサブシステムや HPC アーキテクチャのメモリサブシステムと同様に，STC アーキテクチャのメモリサブシステムでもアクセスされるのはどちらか片方のメモリのみである．SPM 搭載および HPC アーキテクチャのメモリサブシステムの場合，アクセスアドレスの一部のビットからアドレス空間の領域を判定し，その領域に割り当てられた唯一のメモリをアクセスすればよい．STC アーキテクチャのメモリサブシステム場合，判定された領域が Single キャッシュ領域の場合に限りアクセスすべきメモリが Main キャッシュであると一意に決まる．一方，判定された領域が Multi キャッシュ領域の場合はどちらのキャッシュにコード／データがある（もしくはどちらにもない）かわからない．まず Small キャッシュへアクセスし，ミスした場合に Main キャッシュをアクセスする方法と取れば，L0 キャッシュを搭載したメモリサブシステムと同様にプロセッサの性能低下を引き起こす．両キャッシュへ同時にアクセスすれば，プロセッサの性能低下は発生しないが無駄なエネルギーを消費することにつながる．STC アーキテクチャでは，Small キャッシュのタグを SRAM アレイではなく D 型フリップフロップ（FF）アレイを用いて構成することにより，データフィールドへアクセスする前に Small キャッシュのヒット／ミスを判定し，この問題を解決する．

図 5 および図 6 に 2 行× 2 列の一般的な SRAM アレイおよび FF アレイを示す． SRAM アレイが保持している値は通常以下のような手順で読み出す．

（1） アクセスアドレスのインデックスをデコードし，活性化させるワードラインを決定する．デコードと同時にビットラインをプリチャージする．

（2） Enable 信号によりワードラインを活性化させる．

（3） センスアンプ回路を用いて，データを読み出す．

セット・アソシアティブ・キャッシュからコード／データを読み出す場合はさらに上記手順に加え，アクセスアドレスのタグと SRAM アレイから読み出したタグの値を比較し，ヒット／ミス判定を行う．ヒットした場合は，タグの読み出しと同時に読み出していた全てのウェイのデータフィールドの値のうちヒットしたウェイの値をプロセッサコアへ渡す．SRAM アレイで構成されるキャッシュではこれらの動作が通常 1 サイクルで行われている．一方，FF が保持している値は SRAM セルとは異なり常にアクセス可能である．SRAM アレイの読み出し手順（2）のワードライン活性化よりも先に Small キャッシュのヒット／ミス情報がわかれば，どちらのキャッシュへアクセスすべきかが決定する．

FF アレイを用いた時のヒット／ミス情報の判定に要する時間は，マルチプレクサおよびタグ比較器の遅延時間を足し合わせたものに等しい．Multi キャッシュ領域のサイズを Small キャッシュのサイズの N 倍とすると，$\lceil log_2(N) \rceil$ ビットのタグメモリを実装する必要がある．Multi キャッシュ領域のサイズを 4MB，Small キャッシュのサイズを 1KB，ラインサイズを 32B と仮定すると，12 ビット× 32 ライン分のタグメモリが必要である．商用の 65nm CMOS プロセスを用いてこのタグメモリ用のマルチプレクサおよびタグ比較器を設計し遅延時間を見積もった結果，それぞれ約 550 ps，約 420 ps であった．SRAM アレイの読み出し手順（2）がシングルサイクルの半分のタイミングで実行されるとすると，動作周波数が 500 MHz 以下の組込みプロセッサにおいてはキャッシュのワードラインを活性化させる前に Small キャッシュのヒット／ミスを判定可能である．

図 7 に 2 ウェイ・セット・アソシアティブの Small キャッシュを示す．Region タグはアドレス空間のどちらの領域に含まれるかを判定するために使用される．Small タグは Small キャッシュにおけるタグである．Region タグおよび Small タグを並べた

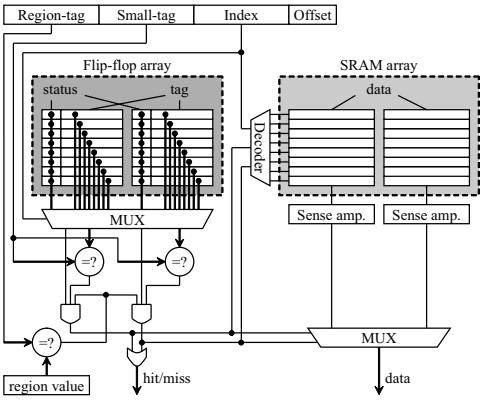

図 7　2 ウェイ・セット・アソシアティブ Small キャッシュ.

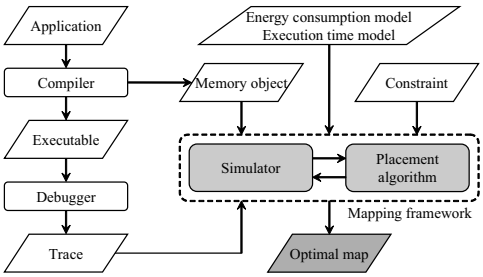

図 8　コンパイラフレームワーク.

ものが Main キャッシュ用のタグ（Main タグ）である．Small キャッシュのヒット／ミス判定は，Region タグを用いた領域判定と同時に行われる．

4. コードおよびデータ配置

4.1 コンパイラフレームワーク

メモリサブシステムの消費エネルギーはコードおよびデータ配置に強く依存しているため，メモリサブシステムの消費エネルギー最適化問題は，メモリオブジェクトの配置問題と捉えることができる．STC アーキテクチャを有効活用するためのコンパイラフレームワークを図 8 に示す．ターゲットアプリケーションをコンパイルし，実行ファイルおよび初期配置アドレスが決まったメモリオブジェクトのリストを得る．デバッガを用いることでアドレストレースを取得する．最適なコードおよびデータ配置を求めるために，メモリオブジェクトリスト，アドレストレースおよびいくつかの制約条件をマッピングフレームワークに入力する．マッピングフレームワーク中のシミュレータは次節で示すエネルギーモデルおよびタイミングモデルを使用して消費エネルギーおよび実行サイクル数を見積もる．コード配置のアルゴリズムは次々節で説明する．

4.2 エネルギーおよびタイミングモデル

メモリサブシステムの消費エネルギー E_{total} およびターゲットアプリケーションの実行時間 T_{total} は以下のような式で表す

Code_and_Data_Placement
Input: TR,F,t_{const}
Output: F'
E_{min}=infinity;
repeat
  for$(t = 0; t < |F|; t++)$**do**
    $o = F[t]$;
    tentatively place memory object o to the multi-cache region;
    TR' = modified TR according to the relocation of o;
    Calculate $E_{tmp} = Energy(TR')$;
    Calculate $t_{tmp} = time(TR')$;
    if($E_{tmp} \leqq E_{min}$ and $t_{tmp} \leqq t_{const}$)
      $E_{min} = E_{tmp}$;
      $o_{best} = o$;
    end if
  end for
  Remove o_{best} from F;
  Append o_{best} into F';
  Update TR according to F and F';
until E_{min} **stops decreasing**
Output F'

図 9　コードおよびデータ配置アルゴリズム.

ことができる．

$$\begin{aligned} E_{total} &= N_{RStag} \cdot E_{RStag} + N_{RSdata} \cdot E_{RSdata} \\ &\quad + N_{RM} \cdot E_{RM} + N_{Roff} \cdot E_{Roff} \\ &\quad + N_{WStag} \cdot E_{WStag} + N_{WSdata} \cdot E_{WSdata} \\ &\quad + N_{WM} \cdot E_{WM} + N_{Woff} \cdot E_{Woff} \qquad (1) \\ T_{total} &= N_{RSdata} \cdot T_{RSdata} + N_{RM} \cdot T_{RM} \\ &\quad + N_{Roff} \cdot T_{Roff} + N_{WSdata} \cdot T_{WSdata} \\ &\quad + N_{WM} \cdot T_{WM} + N_{Woff} \cdot T_{Woff} \qquad (2) \end{aligned}$$

ここで，E_{RStag}，E_{RSdata}，E_{RM}，E_{Roff}，E_{WStag}，E_{WSdata}，E_{WM} および E_{Woff} はそれぞれ，Small キャッシュのタグ読み出し，Small キャッシュのデータ読み出し，Mail キャッシュの読み出し，オフチップメモリの読み出し，Small キャッシュのタグ書き込み，Small キャッシュのデータ書き込み，Main キャッシュの書き込みおよびオフチップメモリの書き込みの操作に関する消費エネルギーを表している．N_{RStag}，N_{RSdata}，N_{RM}，N_{Roff}，N_{WStag}，N_{WSdata}，N_{WM} および N_{Woff} は各操作の実行回数を表す．T_{RSdata}，T_{RM}，T_{Roff}，T_{WSdata}，T_{WM} および T_{Woff} は各操作の実行時間を表している．変数 N_x の値はマッピングフレームワーク中のシミュレータにアドレストレースを入力し求める．変数 E_x および T_x の値は各メモリモジュールの値を用いる．

4.3 アルゴリズム

図 9 に示すアルゴリズムの入力は，メモリオブジェクトリスト F である．メモリオブジェクトには関数，グローバル変数，および定数が含まれる．アプリケーションプログラムのアドレストレース TR もまたアルゴリズムの入力である．実行時間制

約 t_{const} は問題の制約として与えられる．メモリサブシステムの消費エネルギーおよびターゲットアプリケーションの実行時間は TR および前節で示したエネルギーモデルおよびタイミングモデルを用いて見積もる．アルゴリズムのメインループは全てのメモリオブジェクトが Single キャッシュ領域に配置された状態で始まる．そして，Multi キャッシュ領域に配置すべきメモリオブジェクトの最適なセット F' が見つかる．この操作は一つのメモリオブジェクト o を F の先頭から一つ選び，o を Multi キャッシュ領域へ配置したときのエネルギー削減量を計算することで行われる．この計算は F の全てのメモリオブジェクトに対して行われる．全てのメモリオブジェクトに対して計算を行ったのち，最もエネルギーが削減されたメモリオブジェクト o_{best} を F から F' へ移動させる．この操作をエネルギーが削減されなくなるまで続ける．最終的にアルゴリズムは最適な F' を出力する．

5. 実験結果

我々は実験に EEMBC DENBench 1.0 [20] をベンチマークとして利用した．利用したベンチマークを表 1 に示す．東芝 MeP アーキテクチャ用の GNU C コンパイラおよびデバッガをメモリオブジェクトリストの作成およびアドレストレースの作成に使用した．各シングルタスクベンチマークのアドレストレースの長さは，最初の 100 万命令をスキップしたあとの 1000 万命令とした．表 1 に示したアクティブコードサイズとはアドレストレース中に現れるコードサイズのことである．我々は実験でオリジナルのアドレストレース *taskset*A，*taskset*B および *taskset*C をシングルタスクアプリケーションのアドレストレースを複数混成して作成した．例えば *taskset*A は *aes*，*cjpeg*，*des*，*djpeg* および $huffde$ からなる．*taskset*A では，各シングルタスクのアプリケーションが 100 万命令ごとに切り替わりながら実行されていくことを仮定した．*taskset*A で $huffde$ が実行されたあとは *aes* の続きに戻る．これらマルチタスクを模擬したアドレストレースの長さは最初の 500 万命令をスキップしたあとの 3000 万命令とした．

我々は図 8 に示したコードおよびデータのマッピングフレームワークを開発した．コードおよびデータ配置アルゴリズムも開発したフレームワークに実装している．実行時間の制約は L0 キャッシュを搭載したメモリサブシステムの実行時間と設定した．各メモリモジュールの消費エネルギーモデルおよび実行時間は商用の 65nm CMOS テクノロジを用いて見積もった値を使用した．また，オフチップメモリとして Micron mobile SDRAM MT48H16M32LFCM-75 IT [21] のモデルを使用した．実験では表 2 に示すオンチップメモリのコンフィグレーションを想定した．実験で想定するメモリサブシステムは命令に関するメモリサブシステムとし，8 バイトの命令バッファを持つと仮定した．HPC アーキテクチャにおける Mini キャッシュ領域および STC アーキテクチャにおける Multi キャッシュ領域のサイズは 4MB と仮定した．

図 10 に関連研究および STC アーキテクチャのメモリサブシステムの消費エネルギーの結果を示す．消費エネルギーの値は

表 2　オンチップメモリのコンフィグレーション

On-chip Memory	Configurations
L1-cache Main-cache	4way 8KB or 2way 8KB or 2way 4KB line size:32byte
L0-cache, Mini-cache Small-cache	2way 1KB or Direct map 1KB or Direct map 512B, line size:32byte
SPM	1KB or 512B

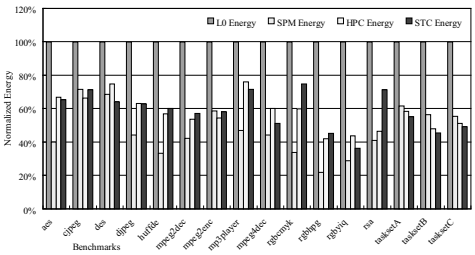

図 10　正規化した消費エネルギー．

L0 キャッシュを搭載したメモリサブシステムの消費エネルギーで正規化している．HPC および STC アーキテクチャのメモリサブシステムでは二種類の領域を使いこなしエネルギーを削減できている．これら HPC および STC アーキテクチャに対しては開発したフレームワークを使用してコード配置を行った．また，SPM に関しては最も実行回数が多い関数から順に SPM 領域へ割り当てていき，SPM 領域に配置するコードがなくなる順に割当てを行った．STC アーキテクチャでは，L0 キャッシュ搭載のメモリサブシステムと比較して性能のオーバーヘッドなく最大で約 64%，平均して約 41%の消費エネルギーを削減できている．図 10 からわかるように多くのシングルタスクのアプリケーションでは SPM を搭載したメモリサブシステムが最もよい結果を示している．この原因は各ベンチマークのアクティブコードサイズが小さいことに起因していると考えられる．一方で，STC アーキテクチャはマルチタスクを模擬したベンチマークにおいて効果がある．

6. おわりに

本稿では，シングルサイクルアクセス可能な二階層キャッシュ（STC）アーキテクチャ，および STC アーキテクチャを有効活用するためのコンパイラフレームワークを提案した．EEMBC DENBench 1.0 ベンチマークを使用した実験では，STC アーキテクチャのメモリサブシステムの消費エネルギーは L0 キャッシュ搭載のメモリサブシステムと比較して性能低下なく最大で約 64%，平均して約 41%の消費エネルギーを削減できた．

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，株式会社半導体理工学研究センター，(株) イー・シャトルおよび富士通株式会社の協力で行われたものである．本研究の一部は，科学技術振興事業団（JST）の戦略的創造研究推進事業（CREST）「情報システムの超低消費電力化を目指した技術革新と総合化技術」の支援によるものである．

表 1　ベンチマークアプリケーション

Benchmarks	Description	Code size	Active code size
aes	AES	55.10 KB	3.47 KB
cjpeg	JPEG Compression	71.34 KB	10.53 KB
des	DES	56.76 KB	7.91 KB
djpeg	JPEG Decompression	75.95 KB	6.50 KB
huffde	Huffman Decoder	49.23 KB	0.88 KB
mpeg2dec	MPEG-2 Decoder	84.76 KB	9.66 KB
mpeg2enc	MPEG-2 Encoder	112.30 KB	33.38 KB
mp3player	MP3 Player	64.64 KB	7.31 KB
mpeg4dec	MPEG-4 Decoder	256.05 KB	27.47 KB
rgbcmyk	RGB to CMYK Converter	49.18 KB	2.03 KB
rgbhpg	High-Pass Gray-Scale Filter	49.45 KB	2.19 KB
rgbyiq	RGB to YIQ Converter	49.38 KB	2.09 KB
rsa	RSA	100.23 KB	10.91 KB
tasksetA	aes, cjpeg, des, djpeg, huffde	308.38 KB	27.34 KB
tasksetB	mpeg2dec, mp3player, rgbcmyk, rgbhpg	248.02 KB	21.19 KB
tasksetC	mpeg2enc, mpeg4dec, rgbyiq, rsa	517.96 KB	62.66 KB

文　献

[1] S. Segars, "Low-power design techniques for microprocessor," *International Solid-State Circuits Conference Tutorial*, Feb. 2001.

[2] J. Montanaro *et al.*, "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1703–1714, Nov. 1996.

[3] S. Yamaguchi, T. Ishihara, and H. Yasuura, "A single cycle accessible two-level cache architecture for reducing the energy consumption of embedded systems," in *Proc. of International SoC Design Conference*, pp. 188–191, Nov. 2008.

[4] C.-L. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: A case study," in *Proc. of International Symposium on Low Power Design*, pp. 63–68, Apr. 1995.

[5] M. B. Kamble and K. Ghose, "Analytical energy dissipation models for low power caches," in *Proc. of International Symposium on Low Power Electronics and Design*, pp. 143–148, Aug. 1997.

[6] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis, "Architectural and compiler support for energy reduction in the memory hierarchy of high performance microprocessors," in *Proc. of International Symposium on Low Power Electronics and Design*, pp. 70–75, Aug. 1998.

[7] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The filter cache: An energy efficient memory structure," in *Proc. of International Symposium on Microarchitecture*, pp. 184–193, Dec. 1997.

[8] R. Panwar and D. Rennels, "Reducing the frequency of tag compares for low power I-cache design," in *Proc. of International Symposium on Low Power Design*, pp. 57–62, Apr. 1995.

[9] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," *ACM Trans. on Embedded Computing Systems*, vol. 1, no. 1, pp. 6–26, Nov. 2002.

[10] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," in *Proc. of Design, Automation, and Test in Europe*, pp. 409–415, Mar. 2002.

[11] M. Verma, L. Wehmeyer, and P. Marwedel, "Cache-aware scratchpad allocation algorithm," in *Proc. of Design, Automation, and Test in Europe*, vol. 2, pp. 1264–1269, Feb. 2004.

[12] Y. Ishitobi, T. Ishihara, and H. Yasuura, "Code placement for reducing the energy consumption of embedded processors with scratchpad and cache memories," in *Proc. of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*, pp. 13–18, Oct. 2007.

[13] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and J. M. Mendias, "An integrated hardware/software approach for run-time scratchpad manegement," in *Proc. of Design Automation Conference*, pp. 238–243, Jun. 2004.

[14] M. Kandemir, J. Ramanujam, M. J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, "Dynamic management of scratch-pad memory space," in *Proc. of Design Automation Conference*, pp. 690–695, Jun. 2001.

[15] S. Udayakumaran, A. Dominguez, and R. Barua, "Dynamic allocation for scratch-pad memory using compile-time decisions," *ACM Trans. on Embedded Computing Systems*, vol. 5, no. 2, pp. 472–511, May 2006.

[16] M. Verma, L. Wehmeyer, and P. Marwedel, "Dynamic overlay of scratchpad memory for energy minimization," in *Proc. of International Conference on Hardware/Software Codesign and System Synthesis*, pp. 104–109, Sep. 2004.

[17] A. González, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in *Proc. of International Conference on Supercomputing*, pp. 338–347, Jul. 1995.

[18] A. Shrivastava, I. Issenin, and N. Dutt, "Compilation techniques for energy reduction in horizontally partitioned cache architectures," in *Proc. of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 90–96, Sep. 2005.

[19] A. Shrivastava, I. Issenin, and N. Dutt, "A compiler-in-the-loop framework to explore horizontally partitioned cache architectures," in *Proc. of Asia and South Pacific Design Automation Conference*, pp. 328–333, Jan. 2008.

[20] EEMBC, *DENBench 1.0*, http://www.eembc.org/benchmark/digital_entertainment_sl.php.

[21] Micron Technology, Inc., *MICRON 512Mb Mobile SDRAM : MT48H16M32LFCM-75 IT Data Sheet*, http://www.micron.com/products/dram/mobilesdram/.