組込みシステムの消費エネルギー削減を目的とした不均一キャッシュアーキテクチャ

山口, 誠一朗
九州大学大学院システム情報科学府

石飛, 百合子
九州大学大学院システム情報科学府

室山, 真徳
九州大学システムLSI 研究センター

石原, 亨
九州大学システムLSI 研究センター

他

出版情報：DAシンポジウム. 2008, pp.25-30, 2008-08-26
バージョン：
権利関係：

組込みシステムの消費エネルギー削減を目的とした不均一キャッシュアーキテクチャ

山口 誠一朗 † 石飛 百合子 † 室山 真徳 ‡ 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 大学院システム情報科学研究院

概要 本稿では，組込みシステムにおけるメモリシステムの消費エネルギー削減を目的として，一回あたりのアクセスエネルギーが不均一な二種類のキャッシュメモリを持つキャッシュアーキテクチャを提案する．提案アーキテクチャはアドレス空間を二種類の領域に分割する．一つの領域を Multi-Cache 領域といい，両方のキャッシュメモリを利用できる．もう一つの領域を Single-Cache 領域といい，アクセスエネルギーが大きいキャッシュメモリのみ利用できる．アクセス頻度が高いコード／データを Multi-Cache 領域に配置することでメモリシステムの消費エネルギーを削減する．例を用いた評価では実行サイクル数の悪化なく約 35%のエネルギーを削減できた．

A Non-Uniform Cache Architecture for Reducing the Energy Consumption of Embedded Systems

Seiichiro YAMAGUCHI†, Yuriko ISHITOBI†, Masanori MUROYAMA‡, Tohru ISHIHARA‡, and Hiroto YASUURA‡†

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

Abstract This paper proposes a cache architecture for energy efficient embedded systems. The cache architecture has two non-uniform cache memories, small cache and normal cache, in terms of the energy consumption per access. The cache architecture partitions address spaces into two regions. One is *Multi-Cache region* allowed to use the both of cache memories. The other is *Single-Cache region* allowed to use only normal cache memory which consumes larger energy per access. The energy of memory subsystems can be reduced through assigning frequently accessed code/data into *Multi-Cache region*. Evaluation using a simple example demonstrated that the proposed architecture reduced the energy consumption of memory subsystem approximately 35% with no performance overhead.

1 はじめに

近年では，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となっている．省エネ製品を発売することによりユーザの利便性を高めることができ，それが販売数増加に繋がり企業の利益となる．さらには，地球環境へ配慮することにもなる．

組込みシステムで利用されているマイクロプロセッサには，性能向上を主な目的としてキャッシュメモリが搭載されている．しかしながら，キャッシュメモリで消費されるエネルギーは，マイクロプロセッサで消費されるエネルギーの半分近くを占めている．ARM920T マイクロプロセッサでは 44%（命令用：25%，データ用：19%）[10]，低消費電力システムを特にターゲットとしている StrongARM SA-110 マイクロプロセッサでは 43%（命令用：27%，データ用：16%）がキャッシュメモリによる消費である [8]．したがって，キャッシュメモリで消費されるエネルギーを削減することにより，組込みシステムの消費エネルギーを削減することに大きく貢献できる．本稿ではキャッシュメモリの消費エネルギーを削減するキャッシュアーキテクチャを提案する．

提案するキャッシュアーキテクチャは，消費エネルギーが不均一な二種類のキャッシュメモリを持つ．一つは，従来の L1-Cache と同じキャッシュメモリであり，本稿では Inclusive-Cache と呼ぶ．もう一つは，キャッシュ容量が小さく Inclusive-Cache よりもアクセスエネルギーが小さい Tiny-Cache である．提案アーキテクチャでは，アドレス空間を Multi-Cache 領域と Single-Cache 領域の二種類の領域に分割する．Multi-Cache 領域に割り当てられたコード／データは Tiny-Cache を優先的に利用し，Tiny-Cache の利用状況に応じて Inclusive-Cache も利用できる．一方，Single-Cache 領域に割り当てられたコード／データは，Inclusive-Cache のみ利用できる．アクセス頻度が高いコード／データを Multi-Cache 領域に割り当てることにより，一回あたりのアクセスエネルギーを削減し，かつアクセス頻度が高いコード／データのキャッシュミス回数を削減することができる．

本稿の構成は以下のとおりである．2 章ではキャッシュメモリの消費エネルギー削減に関連する既存研究を紹介し，我々のアプローチを述べる．また，関連研究および本研究に簡単な例を適用した場合の評価結果も示す．3 章では提案するキャッシュアーキテクチャの実装および動作について詳しく説明する．最後に 4 章で本稿をまとめ，今後の方針について述べる．

2 関連研究と提案手法

2.1 関連研究

キャッシュメモリと同じく SRAM で構成されるオンチップメモリにスクラッチパッドメモリ（以下，SPM）がある．図 1 に SPM を持つ MPU のメモリシステムおよびアドレス空間の割り当てを示す．SPM はアドレス空間の一部（SPM 領域）が静的に割り当てられており，常に定められたアドレスのコード／データを保持している．キャッシュメモリのようにタグを保持し比較する必要がないため，アクセスエネルギーが小さく参照ミスも発生しない．したがって，アクセス頻度が高いコード／データを SPM に配置することで消費エネルギーを削減することができ，そのような配置手法は文献 [1, 13, 16] などで提案されている．SPM をオーバーレイすることにより SPM アクセス回数を増やし，さらに消費エネルギーを削減する手法もある [2, 6, 15, 17].

アクセスエネルギーが小さい小容量のキャッシュメモリを追加する手法もある．L1-Cache と MPU core の間のメモリ階層に追加する手法と，L1-Cache と同じメモリ階層に追加する手法がある．前者を一般に L0-Cache と呼び，S-Cache[9]，Block Buffering[3, 14]，Filter Cache[7]，および Loop Cache[5] などが提案されている．図 2 に L0-Cache を持つ MPU のメモリシステムおよびアドレス空間の割り当てを示す．L0-Cache を利用したシステムでは，メモリアクセスの時間的局所性が高い場合に消費エネルギーの削減効果が大きい．

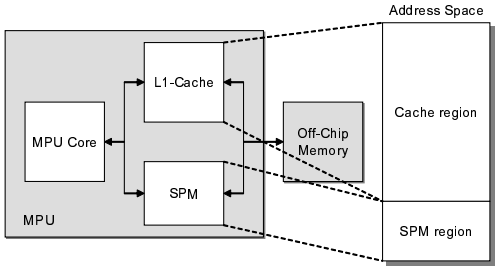

図 1: SPM を持つ MPU のメモリシステムおよびアドレス空間の割り当て

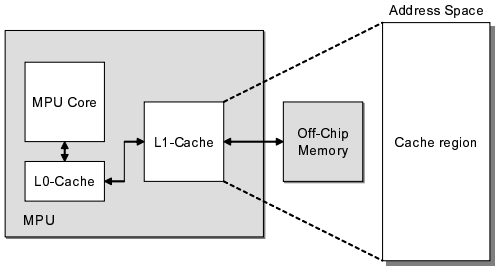

図 2: L0-Cache を持つ MPU のメモリシステムおよびアドレス空間の割り当て

一方，L1-Cache と同じメモリ階層に追加する手法としては Horizontally Partitioned Cache（HPC）アーキテクチャがある [4, 11, 12]．図 3 に HPC アーキテクチャの MPU のメモリシステムおよびアドレス空間の割り当てを示す．HPC アーキテクチャでは，Main-Cache と呼ばれる従来の L1-Cache と同じキャッシュメモリと小容量の Mini-Cache を同じメモリ階層に持つ．アドレス空間も Main-Cache 領域と Mini-Cache 領域の二種類に分割されており，それぞれのキャッシュメモリに対応付けられている．MPU core がアクセスするアドレスを判定し，どちらか一方のキャッシュメモリのみアクティブにする．SPM を用いた手法と同様にアクセス頻度が高いコード／データを Mini-Cache 領域に配置することでアクセスエネルギーを削減することができる．

2.2 提案手法

提案するアーキテクチャでも L0-Cache を持つアーキテクチャや HPC アーキテクチャと同様に小容量のキャッシュメモリを持つ．本稿では小容量のキャ

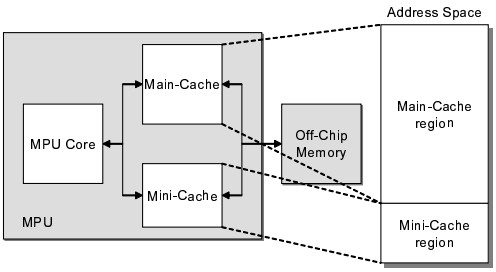

図 3: HPC アーキテクチャの MPU のメモリシステムおよびアドレス空間の割り当て

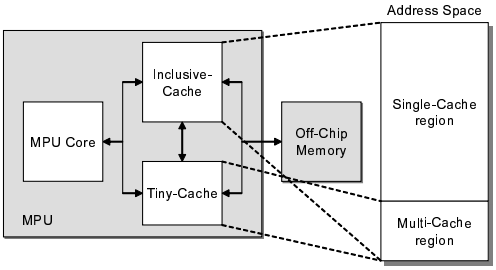

図 4: IHPC アーキテクチャの MPU のメモリシステムおよびアドレス空間の割り当て

ッシュメモリを Tiny-Cache，従来のキャッシュメモリを Inclusive-Cache と呼び，本アーキテクチャを Inclusive Horizontally Partitioned Cache（IHPC）アーキテクチャと呼ぶ．図 4 に IHPC アーキテクチャの MPU のメモリシステムおよびアドレス空間の割り当てを示す．IHPC 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を二種類の領域に分割する．HPC アーキテクチャとの違いは，それぞれの領域に対して一つのキャッシュメモリを割り当てるのではなく，二つのうち一つの領域は両方のキャッシュメモリを利用することができる点である．両方のキャッシュメモリを利用できる領域を Multi-Cache 領域，もう一方の領域を Single-Cache 領域といい，Single-Cache 領域は Inclusive-Cache のみ利用可能である．

IHPC アーキテクチャは，アクセス頻度が高いコード／データを Multi-Cache 領域に配置することで消費エネルギーの削減を実現する．Multi-Cache 領域に配置されたコード／データは優先的に Tiny-Cache を利用し，Tiny-Cache に保持されているコード／データの Tiny-Cache 利用優先度に応じて Inclusive-Cache を利用する．Tiny-Cache 利用優先度はプログラマが指定し，例えばコードの場合，アプリケーション単位，タスク単位，関数単位，あるいは基本ブロック単位等で指定する．また，Tiny-Cache が長期間アクセスされない Tiny-Cache 利用優先度の高いコード／データを保持し続けないための機構も持つ．

HPC アーキテクチャと同様に IHPC アーキテクチャでも MPU core がアクセスするキャッシュメモリはどちらか一方である．HPC アーキテクチャでは一つの領域に対して一つのキャッシュメモリが割り当てられているため，MPU core がアクセスするアドレスの一部のビットを判定するだけで，どちらのキャッシュメモリをアクティブにするかを決定することができる．IHPC アーキテクチャにおいても Single-Cache 領域に配置されたコード／データへアクセスする際は，アドレスの一部のビットを判定するだけで Tiny-Cache を非アクティブ状態（ワードラインを活性化させない）にすることが可能である．しかしながら，Multi-Cache 領域は両方のキャッシュメモリを利用可能なため，アドレスの一部のビットを判定するだけではどちらのキャッシュメモリをアクティブにするか決定することができない．したがって，IHPC アーキテクチャでは，キャッシュメモリをアクティブにする前（ワードラインを活性化させる前）に Tiny-Cache のヒット／ミスを判定する．言い換えれば，0.5 サイクルで Tiny-Cache ヒット／ミスを判定する機構を持つ．

Multi-Cache 領域かつ Tiny-Cache ヒットであれば Tiny-Cache をアクティブにし，それ以外は Inclusive-Cache をアクティブにする．また，Multi-Cache 領域のコード／データが Inclusive-Cache に保持されている場合，Tiny-Cache 利用優先度やコード／データの非アクセス期間などに応じてコード／データを入れ替える．IHPC アーキテクチャの具体的な実現法や動作は 3 章で説明する．

2.3 適用例

従来の L1-Cache のみ持つシステム，L0-Cache を持つシステム，HPC アーキテクチャのシステム，および提案する IHPC アーキテクチャのシステムにおいて，表 1 に示すパラメータを持つタスクを図 5 に示す順序で実行した場合の評価を行う．ただし，評価対象は命令キャッシュとする．図 5 はタスク B の優先度が高く，他のタスクに割り込んで実行されている例である．表 1 の Code Size のカッコ内にある一つ目のサイズは，IHPC アーキテクチャにおける Multi-Cache 領域に配置するコードサイズであり，HPC アーキテクチャでは Mini-Cache 領域に対応する．また，カッコ内にある二つ目のサイズは，Single-Cache 領域（HPC アーキテクチャでは Main-Cache 領域）に配置するコードサイズである．従来のシステムおよび L0-Cache を持つシステムではアドレス空間の領域は Cache 領域一つであるため，全てのコードが Cache 領域に配置される．Tiny-Cache 利用優先度が高いタスクのコードは低いタスクのコードによって追い出されることはなく，この例では長

表 1: 各タスクのパラメータ

Task	Code Size	Tiny-Cache Priority
A	4KB (512B, 3.5KB)	Low
B	1KB (512B, 512B)	High
C	4KB (512B, 3.5KB)	Low
D	2KB (512B, 1.5KB)	Low

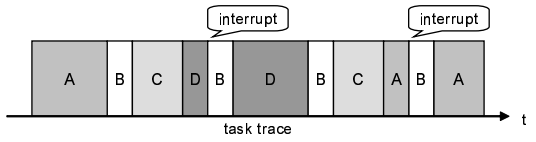

図 5: 適用するタスクトレース

期間アクセスされないことによる追い出しは発生しないとする．表 2 に各メモリのパラメータを示す．ただし，キャッシュメモリのラインサイズは全てにおいて共通で 16 バイトとする．全ての命令は 1 サイクルで終了すると仮定し，さらに，全てのタスクに対して次の二つの仮定をおく．1) Multi-Cache 領域（Mini-Cache 領域）に配置されるコードは，一度のタスク実行で 500 回実行される．2) Single-Cache 領域（Main-Cache 領域）に配置されるコードは，タスクの起動時と終了時に一回ずつ実行される．

以上の条件下において，各キャッシュメモリのアクセス回数とミス回数，各メモリシステムにおけるタスクの実行サイクル数と消費エネルギーを見積もった．表 3 にその結果を示す．ただし，表 3 における Small-Cache とは，L0-Cache を持つアーキテクチャにおいては L0-Cache，HPC アーキテクチャにおいては Mini-Cache，IHPC アーキテクチャにおいては Tiny-Cache のことを意味している．また，Large-Cache とは，従来のアーキテクチャおよび L0-Cache を持つアーキテクチャにおいては L1-Cache，HPC アーキテクチャにおいては Main-Cache，IHPC アーキテクチャにおいては Inclusive-Cache のことを意味している．IHPC アーキテクチャは従来のアーキテクチャと比較し，実行サイクル数が増加することなく消費エネルギーを約 35%削減できている．また，関連研究と比較しても実行サイクル数および消費エネルギーともに改善しており，実アプリケーションにおいても効果があることを期待できる．

3 IHPC アーキテクチャ

3.1 実装

図 6 に Tiny-Cache が 2way，Inclusive-Cache が 4way の場合の IHPC アーキテクチャを示す．IHPC アーキテクチャではアドレスを Region-Tag，Tiny-Tag，Index，および Offset にわける．Region-Tag はアドレスが Single-Cache 領域と Multi-Cache 領域のどちらを指しているかを判定するためのタグであり，Tiny-Tag は Tiny-Cache のためのタグである．Inclusive-Cache のためのタグ（Inclusive-Tag）は，Region-Tag と Tiny-Tag を合わせたものである．

表 2: 各メモリのパラメータ

	Memory Size	Access Latency	Access Energy
L0-Cache	1KB 2way	1cycle	10pJ
Mini-Cache	1KB 2way	1cycle	10pJ
Tiny-Cache	1KB 2way	1cycle	10pJ
L1-Cache	8KB 4way	1cycle	30pJ
Main-Cache	8KB 4way	1cycle	30pJ
Inclusive-Cache	8KB 4way	1cycle	30pJ
Off-Chip Mem.	-	20cycles	20nJ

両キャッシュメモリのワードラインを活性化させる前に Tiny-Cache のヒット／ミスを判定するために，Tiny-Cache のステータスフィールドおよびタグフィールドはフリップフロップアレイにより構成される．SRAM アレイに保持されている値を読み出すには，ビットラインのプリチャージ，ワードラインの活性化，およびセンスアンプによる読み出しという手順が必要であるが，フリップフロップアレイに保持されている値は常にアクセス可能である．したがって，高速なヒット／ミス判定が可能である．

3.2 動作

IHPC アーキテクチャでのコード／データ読み出し手順，およびコード／データのキャッシュメモリへの書き込み手順を説明する．

コード／データ読み出し手順

手順 R1. Tiny-Cache ヒット／ミス判定

Index で示される該当セットのタグの値をフリップフロップアレイから選択し，Tiny-Tag と比較する．Region-Tag の比較も同時に行い，Region-Tag および Tiny-Tag がともに一致した場合，Tiny-Cache ヒットとなり手順 R2 へ進む．Tiny-Cache ミスの場合は手順 R3 へ進む．

手順 R2. Tiny-Cache コード／データ読み出し

ヒットしたウェイのみワードラインを活性化し，SRAM アレイからデータを読み出す．この時，Inclusive-Cache のワードラインは活性化させない．

手順 R3. Inclusive-Cache コード／データ読み出し

全てのウェイのワードラインを活性化し，従来のキャッシュメモリの動作と同様にタグフィールドとデータ

表 3: 評価結果

Architectures	Small-Cache*		Large-Cache**		Off-Chip	Runtime	Energy
	Accesses	Misses	Accesses	Misses	Accesses	[cycles]	[μJ]
Conventional	—	—	584,960	1,024	1,024	607,488 (100.0%)	38.06 (100.0%)
L0-Cache	584,960	2,464	2,464	1,024	1,024	614,880 (101.2%)	26.46 (69.5%)
HPC	576,000	256	8,960	736	992	606,784 (99.9%)	25.89 (68.0%)
IHPC	576,000	128	9,472	800	928	606,208 (99.8%)	24.63 (64.7%)

* Small-Cache denotes L0-Cache in L0-Cache architecture, Mini-Cache in HPC architecture, Tiny-Cache in IHPC architecture.
** Large-Cache denotes L1-Cache in conventional and L0-Cache architecture, Main-Cache in HPC architecture, Inclusive-Cache in IHPC architecture

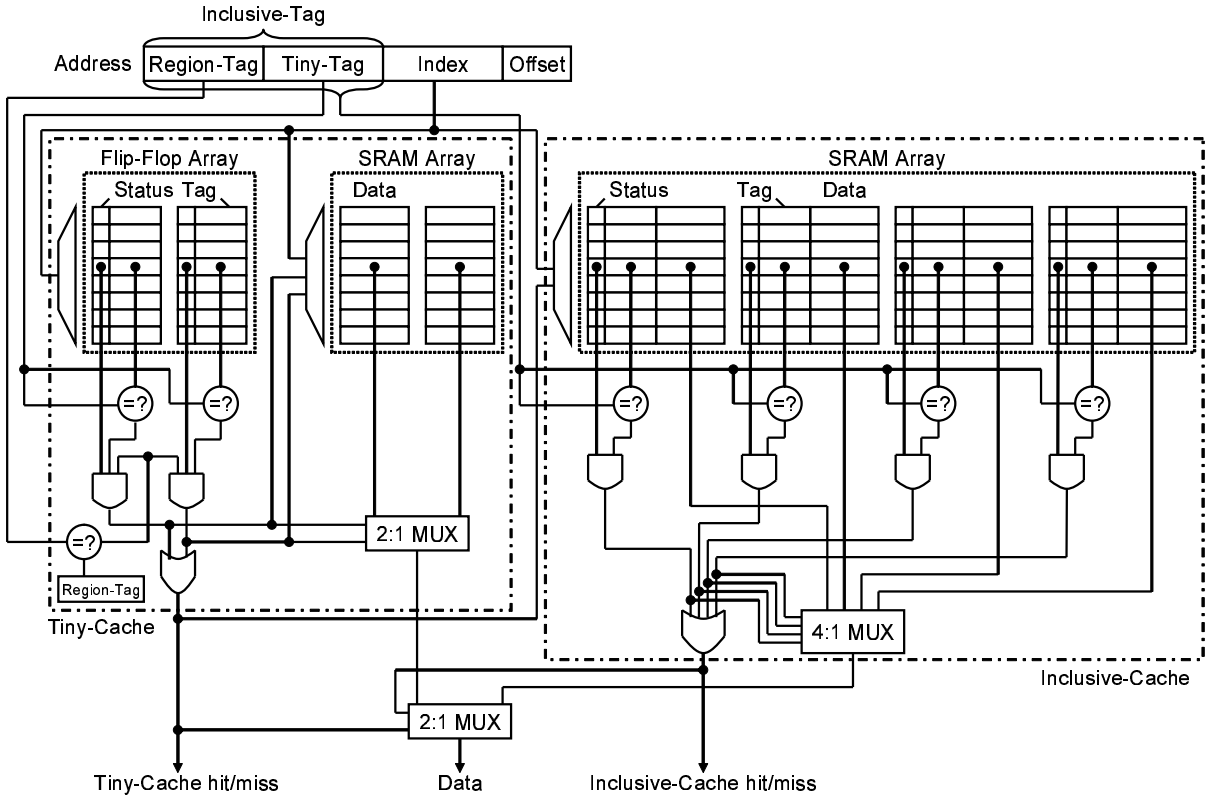

図 6: IHPC アーキテクチャ（Tiny-Cache:2way, Inclusive-Cache:4way の場合）

フィールドから保持している値を同時に読み出す．この時，Tiny-Cache のワードラインは活性化させない．Inclusive-Tag とタグ比較を行い，Inclusive-Cache ヒットの場合は，ヒットしたウェイから読み出したコード／データを選択し出力する．一方，Inclusive-Cache ミスの場合は下位層のメモリからコード／データを取得し，コード／データ書き込み手順にしたがって値を書き込む．

コード／データ書き込み手順

手順 W1. 領域判定

書き込むコード／データのアドレスが Single-Cache 領域か Multi-Cache 領域かを Region-Tag 部により判定する．Single-Cache 領域の場合は手順 W2 へ，Multi-Cache 領域の場合は手順 W3 へ進む．

手順 W2. Inclusive-Cache への書き込み

従来のキャッシュメモリのライン入れ替え方式と同様の方式（例えば LRU 方式）でコード／データを書き込む

手順 W3. Tiny-Cache 利用優先度判定

書き込むコード／データのアドレスの Index により示されるセットのステータスフィールドに保持されている各ラインの Tiny-Cache 利用優先度と書き込むコード／データの Tiny-Cache 利用優先度を比較する．書き込むコード／データの Tiny-Cache 利用優先度が低ければ手順 W2 に従い Inclusive-Cache へ書き込み，同じもしくは高ければ手順 W4 へ進む．

手順 W4. Tiny-Cache への書き込み

Index が示すセットの中で Tiny-Cache 利用優先度が一番低いラインのコード／データを Inclusive-Cache へ追い出して書き込む．Inclusive-Cache へ追い出されるコード／データは手順 W2 に従って書き込む．Tiny-Cache 利用優先度が一番低いラインが複数存在する場合は，LRU 方式で追い出す．また，Tiny-Cache 利用優先度が高いコード／データは長期間アクセスされずに Tiny-Cache を利用し続けることをさけるために，ある一定期間アクセスされないコード／データは優先的に追い出す．

Multi-Cache 領域のコード／データが Inclusive-Cache でヒットした場合，コード／データ書き込み手順 W3 に従って Tiny-Cache とのライン入れ替えを行う．

4 おわりに

本稿では，メモリシステムで消費されるエネルギーを削減するために，消費エネルギーが不均一な二種類のキャッシュメモリを持つ Inclusive Horizontally Partitioned Cache（IHPC）アーキテクチャを提案した．IHPC アーキテクチャはアドレス空間を二種類のキャッシュメモリを利用できる Multi-Cache 領域と，消費エネルギーが大きい従来のキャッシュメモリのみ利用できる Single-Cache 領域の二つの領域に分割する．アクセス頻度が高いコード／データを Multi-Cache 領域に割り当てることでエネルギー削減を達成する．例を用いた評価では従来の手法と比較して，実行サイクル数の増加なく消費エネルギーを約 35%削減することができた．関連研究と比較しても実行サイクル数および消費エネルギーともによい結果となった．

IHPC アーキテクチャは，アクセス頻度が高いコード／データが頻繁に切り替わるマルチタスク環境でより効果を発揮できると考える．今後は実アプリケーションを用いたマルチタスク環境下で，IHPC アーキテクチャおよび関連研究の評価を行う予定である．

謝辞 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．

参考文献

[1] O. Avissar, R. Barua and D. Stewart, “An optimal memory allocation scheme for scratch-pad-based embedded systems,” *IEEE Trans. on Embedded Computing Systems*, Vol. 1, No. 1, pp. 6–26, Nov. 2002.

[2] 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.

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

[4] 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.

[5] N. B. I. Hajj, G. Stamoulis, N. Bellas et al., “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.

[6] 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.

[7] J. Kin, M. Gupta and W. H. M.-Smith, “The filter cache: an energy efficient memory structure,” In *Proc. of International Symposium on Microarchitecture*, pp. 184–193, Aug. 1997.

[8] J. Montanaro, R. T. Witek, K. Anne 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.

[9] 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 Electronics and Design*, pp. 57–62, Aug. 1995.

[10] S. Segars, “Low-power design techniques for microprocessors (Tutorial),” *International Solid-State Circuits Conference*, Feb. 2001.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[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, “Cache-aware scratchpad allocation algorithm,” In *Proc. of Design, Automation and Test in Europe*, Vol. 2, pp. 1264–1269, Feb. 2004.

[17] 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.