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

2008－SLDM－134／2008－EMB－8（19）
2008 / 3 / 27

マルチタスク環境におけるスクラッチパッドメモリ領域活用法

高瀬 英希† 冨山 宏之† 高田 広章†

† 名古屋大学大学院情報科学研究科 〒464-8603 名古屋市千種区不老町
E-mail: †{takase,tomiyama,hiro}@ertl.jp

あらまし 本研究では，マルチタスク環境にスクラッチパッドメモリを活用することにより，組込みシステムの命令メモリにおける消費エネルギーの削減を目指す．固定優先度付きの周期タスクが複数存在するシステムに対応した，スクラッチパッドメモリ領域分割方針を提案する．提案するスクラッチパッドメモリ領域分割方針は，領域分割法，時間分割法，および混合分割法の3種類である．各方針について，タスクごとの領域分割およびコード割当てを同時に決定可能な整数計画問題として定式化する．評価実験を行い，提案手法の有効性を確認した．
キーワード 組込みシステム，スクラッチパッドメモリ，消費エネルギー，マルチタスクシステム

Allocation of Scratch-Pad Memory in Non-Preemptive Multi-Task Systems

Hideki TAKASE†, Hiroyuki TOMIYAMA†, and Hiroaki TAKADA†

† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8603, Japan
E-mail: †{takase,tomiyama,hiro}@ertl.jp

Abstract In this paper, we propose three approaches to allocation of scratch-pad memory for non-preemptive fixed-priority multi-task systems. These approaches can reduce energy consumption of instruction memory. Each approach is formulated as an integer programming problem which simultaneously determines (1) allocation of scratch-pad memory spaces to the tasks, and (2) allocation of functions to the scratch-pad memory space for each task. The experimental results show the effectiveness of the proposed approaches.
Key words embedded systems, scratch-pad memories, energy consumption, multi-task systems

1. はじめに

組込みシステムの性能や計算精度を保証しながら消費エネルギーを削減することは，製造コストや運用コスト，信頼性などの面から，重要な課題となっている．組込みシステムの高性能・高機能化に従う消費エネルギーの増大もまた，近年の大きな問題となっている．これに対応するために，組込みシステムの消費エネルギーを最適化するための研究が現在までに盛んに行われている．

近年では，組込み向けプロセッサの性能向上を目的として，チップ上にキャッシュが組み込まれている．しかし，キャッシュにおける消費エネルギーは，プロセッサ全体の半分近くを占めるという報告がなされており [1]，キャッシュにおける消費エネルギーは無視できないほど大きくなっている．このことから，メモリで消費されるエネルギーを削減することは，組込みシステム全体の消費エネルギーを削減することにつながる．そこで本研究では，小容量で高速なSRAMであるスクラッチパッドメモリ（以下，SPM）の活用に着目する．キャッシュよりもアクセス1回当たりの消費エネルギーが小さいSPMを活用することにより，組込みシステムの消費エネルギー最小化を目指す．

大規模・複雑化が進む現在の組込みシステムでは，複数のタスクを並行して処理することが要求される．リアルタイム応答が重要視される組込みシステムでは，優先度ベースのタスクスケジューリング方式が採用されるのが一般的である．本論文では，このマルチタスク環境において適用可能な，SPM領域の活用手法を提案する．提案する方針は，領域分割法，時間分割法，および混合分割法の3種類である．各方針におけるSPM領域分割の決定は，システムの命令メモリの消費エネルギー削減量を目的関数とした整数計画問題に帰着させる．タスクの起動周期を基にして，タスクごとに使用するSPM領域のメモリ量とSPMに割当てるプログラムコードが同時に決定できる．

本論文の構成は以下のとおりである．まず，2章にて関連研

究について触れる．3章では，本研究の対象とするシステム構成，および，提案する3種類のSPM領域活用方針の詳細を解説する．4章にて，提案手法の有効性を検証するために行った評価実験の結果および考察を示す．最後に，5章でまとめと今後の方針を述べる．

2. 関連研究

SPMは，デコード回路，データ領域および出力器のみからなり，キャッシュにおけるタグ領域やタグ比較器などは構成に含まれない．このため，SPMのアクセス1回当たりの消費エネルギーは，キャッシュよりも小さく抑えられる．いっぽう，SPMは保持する内容を動的に変更するハードウェア機構を持たない．設計者やコンパイラなどが，SPMに割当てる内容を明示的に決定する必要がある．

文献[2]においてAvissarらは，SPM領域へのデータ変数割当てのためのコンパイル手法を提案している．文献[3]では，SPMのアドレス領域に割当てるデータを，そのメモリ量とプログラム実行時の消費エネルギーで定義されるナップサック問題によって決定する手法が提案されている．Banakarらは，このナップサック問題を用いてSPMへのデータ割当てを決定することによって，プログラム実行時の消費エネルギーを，キャッシュのみの場合よりも平均して約40％削減できることを確認している[4]．SPMのアドレス領域への命令またはデータ割当てに，動的計画法を適用する手法として，文献[5]がある．ナップサック問題はNP困難であるため，この最適解を導出する既知の最良アルゴリズムは指数関数時間のオーダとなる．Angioliniらは，動的計画法を用いることによって，アドレス領域割当てに要する計算時間を抑えている．しかし，[2]〜[5]は，シングルタスク環境を対象として想定している．

文献[6]においてJanapsatyaらは，SPMの保持する内容を制御するための独自のハードウェアを用意している．これを用いることによって，SPMの保持する内容を，システム動作中に動的に更新することが可能となっている．

Vermaらは，文献[7]において，タスクが複数存在する環境においてSPMを適用する手法を提案している．マルチタスク処理中の消費エネルギー最適化を目的とし，SPM領域をタスクごとに分配して与える方針を示している．しかし，この研究の対象とする環境は，ラウンドロビン方式（各プロセスの処理を一定時間ずつ順番に行う方式）のスケジューリング規則に従って，複数のタスクを並行して処理する時分割システムが前提である．一般に，リアルタイム処理の要求が高い組込みシステムにおいては，ラウンドロビンやタイムシェアリング方式のスケジューリング規則はあまり採用されない．また，文献[7]の手法では，消費エネルギー最小となるSPM領域の分配は，全数探索を用いるアルゴリズムによって決定される．このため，アルゴリズムへの入力数によって，決定のための探索時間が膨大になる可能性がある．

われわれは，優先度付きスケジューリング方式のマルチタスクシステムに対応した，SPM領域活用手法の確立が必要であると考えている．本論文の提案手法を適用することにより，優先度付きマルチタスクシステムにおいてSPMを有効に活用できるようになる．さらに，SPM領域分割およびデータ割当てを整数計画問題として定式化することにより，各要素を同時に決定することが可能となる．

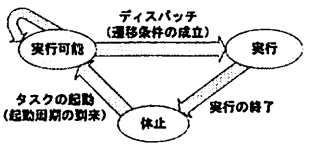


図1　タスクの状態遷移

3. スクラッチパッドメモリ領域活用法

本章では，SPMの有効活用を実現する提案手法の詳細を述べる．提案するSPM領域分割方針は，領域分割法，時間分割法，および混合分割法の3種類である．各方針について，タスクごとのSPM領域分割およびコード割当ての決定は，整数計画問題として定式化する．なお，本研究では，プログラムコードのみを対象とし，メモリ領域への割当て粒度は関数単位とする．

3.1 対象とするシステム構成

本研究の対象とする環境は，処理すべきタスクが複数存在するマルチタスクシステムである．タスクは，図1に示すように，休止，実行可能，および実行の3状態をとる．タスク間の通信・同期処理はなく，各タスクは独立に動作する．全てのタスクは周期的に実行され，その起動周期はシステム動作前に静的に決定される．

タスクのスケジューリング規則は，固定優先度ベース方式に従うとする．CPUが解放された時点で，実行可能状態にある，周期最小のタスクがCPU使用権を得て実行状態に遷移する．また，プリエンプションは発生しないものとする．

3.2 準　　備

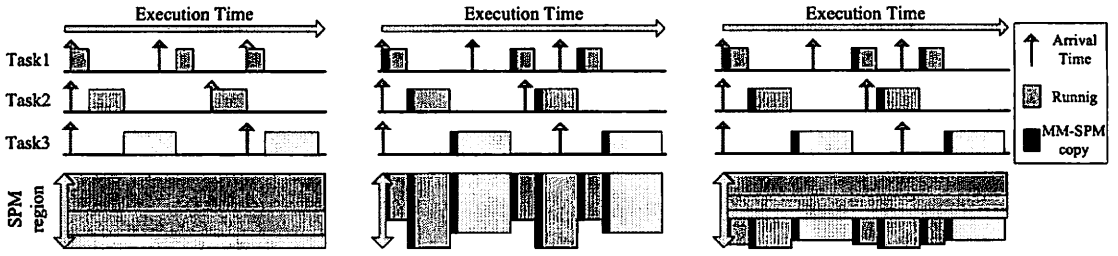
表1にて，本論文における整数計画問題で用いる変数を定義する．表1に示した変数の値は，静的な解析によってすべて取得可能である．

3.3 領域分割法

図2(a)に示すように，領域分割法は，SPMの領域を各タスクに分割して与える手法である．タスクは与えられたSPM領域を，タスクセット実行中に占有して使用する．タスク数3の図2(a)においては，SPMのアドレス空間を3領域に分割して各タスクへ与えている．タスクごとに使用するSPM容量およびSPM領域への関数割当ては，すべて静的に決定する．SPMの保持する内容は，システム動作中に変更されることはない．

領域分割法では，タスクの起動周期を情報として用いることにより，実行頻度の高いコードがSPM領域により多く割当てられるようにする．これにより，SPMのさらなる有効活用を実現する．起動周期の短いタスクの関数は実行頻度が高くなるため，与えられるSPM領域は大きくなる．例えば，図2(a)では，周期最小のTask1は，SPM領域を多く占有している．

それぞれのタスクに与えるSPMの容量と，タスクごとの



(a) 領域分割法　(b) 時間分割法　(c) 混合分割法

図 2　提案手法の概要

SPM 領域への関数割当ての決定は，以下に示す整数計画問題として定式化した．関数の命令フェッチ総数にタスクの起動周期を付与することにより，SPM を用いるによって削減できる消費エネルギーの総量 $Esaving$ の最大化を目指している．

$$\textbf{Maximize}: Esaving = \sum_i \sum_j Esaving_{i,j} \times x_{i,j}$$

$$Esaving_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{SPMgain}$$

$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$

$$\text{s.t.}: \sum_i \sum_j size_{i,j} \times x_{i,j} \leqq SPMsize$$

ここで，式中の決定変数 $x_{i,j}$ は，$func_{i,j}$ が SPM 領域に割当てられるときに 1 となる 0-1 変数である．この解集合を求めることにより，タスクごとのメモリ量およびコード割当てを，同時に決定することが可能である．

表 1　変数の定義

変数名	定義
$task_i$	i 番目のタスク．$1 \leqq i \leqq M$
$period_i$	$task_i$ の起動周期
$func_{i,j}$	$task_i$ の j 番目の関数．$1 \leqq j \leqq N$
$fetch_{i,j}$	$func_{i,j}$ の命令フェッチ総数
$size_{i,j}$	$func_{i,j}$ のコードサイズ
$E_{SPMgain}$	SPM とキャッシュとの読出しアクセスの消費エネルギー差
E_{Cache_read}	キャッシュへの 1 回当たりの読出しアクセスに要する消費エネルギー
E_{SPM_read}	SPM への 1 回当たりの読出しアクセスに要する消費エネルギー
E_{SPM_write}	SPM への 1 回当たりの書込みアクセスに要する消費エネルギー
E_{MM_read}	主記憶への 1 回当たりの読出しアクセスに要する消費エネルギー
$Esaving_{i,j}$	$func_{i,j}$ を SPM 領域に割当てることにより削減できる消費エネルギー量
$Eoverhead_{i,j}$	$func_{i,j}$ を主記憶から SPM にコピーするときに消費されるエネルギー量
$SPMsize$	SPM の総容量
$hyperperiod$	全タスクの起動周期の最小公倍数

3.4　時間分割法

時間分割法では，実行状態にあるタスクが，SPM の全容量を独占して使用する（図 2(b)）．タスクが実行状態に遷移する際に，SPM 領域に割当てるプログラムコードを，主記憶から SPM へ伝送する必要がある．図 2(b) 中の 'MM-SPM copy' が，この主記憶－ SPM 間伝送を行っている部分にあたる．SPM 領域に割当てられる関数は，この伝送にかかる消費エネルギーのオーバヘッドを加味して決定される．このため，オーバヘッドが大きい関数が多く含まれるタスクは，SPM の全容量を使いきるとは限らない．

時間分割法において関数割当ての決定を定式化した整数計画問題は，以下の通りとなる．主記憶－ SPM 間伝送にかかる消費エネルギーのオーバヘッド $Eoverhead_{i,j}$ を考慮した上で，$Esaving$ の最大化を目指している．

$$\textbf{Maximize}: Esaving = \sum_j Esaving_{i,j} \times y_{i,j}$$

$$Esaving_{i,j} = fetch_{i,j} \times E_{SPMgain} - Eoverhead_{i,j}$$

$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$

$$Eoverhead_{i,j} = size_{i,j} \times (E_{SPM\_write} + E_{MM\_read})$$

$$\text{s.t.}: \forall i. \sum_j size_{i,j} \times y_{i,j} \leqq SPMsize$$

決定変数である 0-1 変数 $y_{i,j}$ の解を求めることにより，各タスクでの関数割当てを決定できる．

3.5　混合分割法

混合分割法は，上記 2 つの SPM 領域分割法を併用した手法である．SPM 領域のより柔軟な活用を実現し，消費エネルギーのさらなる削減を目指す．図 2(c) に示すように，領域および時間分割法によって使用する SPM 領域のメモリ量は，消費エネルギー削減量 $Esaving$ が最大となるように，それぞれ分割して与えられる．あるタスクが使用できる SPM の容量は，領域分割法による領域からさらに各タスクに分配される SPM 領域と，時間分割法の方針として用いられる領域の合計になる．混合分割法において定式化した整数計画問題を，次に示す．

$$\textbf{Maximize}: Esaving = Esaving\_rgn + Esaving\_prd$$

$$Esaving\_rgn = \sum_i \sum_j Esaving\_rgn_{i,j} \times x_{i,j}$$

$$Esaving\_prd = \sum_i \sum_j Esaving\_prd_{i,j} \times y_{i,j}$$

$$Esaving\_rgn_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{SPMgain}$$

$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$

$$Esaving\_prd_{i,j} = (fetch_{i,j} \times E_{SPMgain} - Eoverhead_{i,j}) \times \frac{hyperperiod}{period_i}$$

$$Eoverhead_{i,j} = size_{i,j} \times (E_{SPM\_write} + E_{MM\_read})$$

$$\text{s.t.}: SPMsize\_rgn + SPMsize\_prd \leqq SPMsize$$

$$\text{s.t.}: \sum_i \sum_j size_{i,j} \times x_{i,j} \leqq SPMsize\_rgn$$

$$\text{s.t.}: \forall i. \sum_j size_{i,j} \times y_{i,j} \leqq SPMsize\_prd$$

$$\text{s.t.}: \forall i, \forall j. \; x_{i,j} + y_{i,j} \leqq 1$$

本式の決定変数は，$SPMsize\_rgn$，$SPMsize\_prd$，$x_{i,j}$，および $y_{i,j}$ である．$SPMsize\_rgn$ および $SPMsize\_prd$ は，それぞれ，領域分割法および時間分割法として用いる SPM の容量をあらわしている．第 1 の制約式によって，各手法における SPM の容量が決定される．0-1 変数 $x_{i,j}$ は，$func_{i,j}$ が領域分割法による SPM 領域に割当てられるときに 1 となる．$y_{i,j}$ は時間分割法のそれに対応する 0-1 変数である．これらの解を求めることにより，領域および時間分割法によって用いられる SPM の各容量，領域分割法による SPM 領域中で各タスクに割り付けられる SPM の容量，および，タスクごとに SPM に割当てる関数を，全て同時に決定することが可能である．

4. 評価実験

4.1 手　　順

提案する 3 種類の SPM 領域分割方針の有効性を確認するため，評価実験を行った．

実験には，ベンチマークスイート MiBench [8] からプログラムを選定し，表 2 に示す 5 種類のタスクセットを構成して用いた．実験には，C 言語記述である各タスクのコードを，マイクロプロセッサシミュレータ SimpleScalar/ARM [9] 上で実行可能なバイナリコードにコンパイルしたものを用いた．SimpleScalar/ARM は，幅広い用途で採用されている組込みプロセッサのひとつである ARM7TDMI の命令セットに準拠した ISS（命令セットシミュレータ）である．

図 3 に，実験のワークフローを示す．各タスクのバイナリコードに対して，命令セットに準拠したアセンブリ言語への逆アセンブル，および，SimpleScalar/ARM 上でのトレース実行の各処理を行った．これにより，タスクセットに含まれるタスクの各関数のメモリ量と実行命令数の結果を取得した．タスクの起動周期は，タスクセット実行時の CPU 使用率が 60 %程度となるように，命令フェッチの総数に比例して設定した．

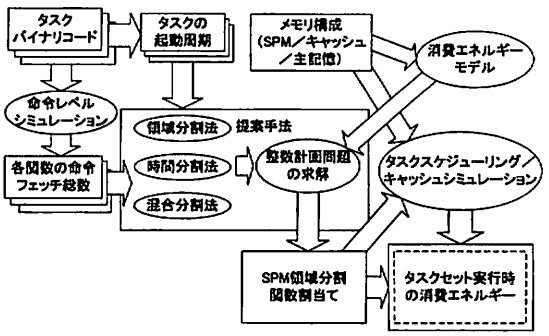


図 3　実験のワークフロー

これらの情報に対して，提案手法を適用した．SPM の領域分割などを決定する整数計画問題の最適解の導出には，シンプレックス法を実装した ILP ソルバである glpsol 4.23 [10] を使用した．一次メモリは，16KBytes 4-way のキャッシュに，4K / 8K / 12K / 16KBytes の SPM をそれぞれ組み合わせて構成した．外部主記憶は，Mobile DDR SDRAM とした．主記憶にはバースト・リードアクセスを行い，アクセス 1 回当たりに 4 ワード分のコードが読み出される．キャッシュ領域に割当てられるプログラムコードについては，タスクスケジューリングおよびキャッシュシミュレーションを行い，キャッシュヒットおよびミス回数を計測した．これらの情報を基にし，タスクセット実行時の命令メモリにおける消費エネルギーの総量を計算した．なお，本研究では，システム動作中にメモリで静的に消費されるリークエネルギーは考慮しない．メモリの消費エネルギーモデルは，一次メモリには CACTI 4.2 [11] を，外部主記憶には Micron System Power Calculator [12] をそれぞれ用いた．

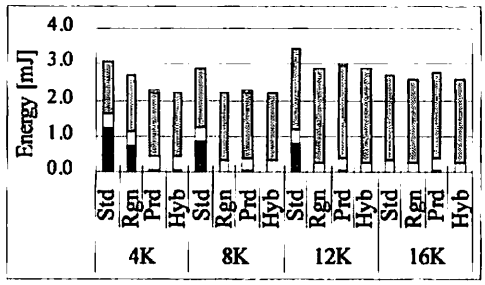
これまでに，優先度付きマルチタスク環境下で SPM を適用する研究は，ほとんど行われていない．このため，提案手法の有効性をみるための評価基準とする次のような手法を導入した．まず，それぞれのタスクに SPM の容量を均等に割り付け，そののちに，各タスクごとに，割り付けられた SPM 容量を制約としたナップサック問題によって SPM 領域へのコード割当てを決定する方法 [3] である．

4.2 実験結果

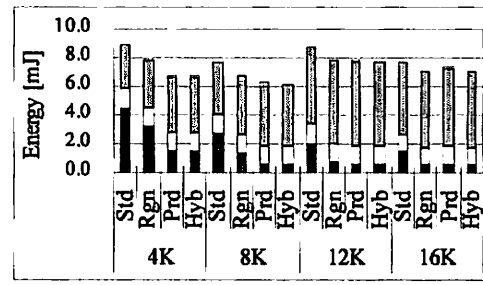
図 4 に，実験結果を示す．グラフの縦軸は，タスクセット実行中にシステムの命令メモリで消費されるエネルギー値（単位は mJ）を示している．各タスクの起動周期は異なるため，周期の最小公倍数時間における消費エネルギーを計算した．

表 2　本実験で用いたタスクセット

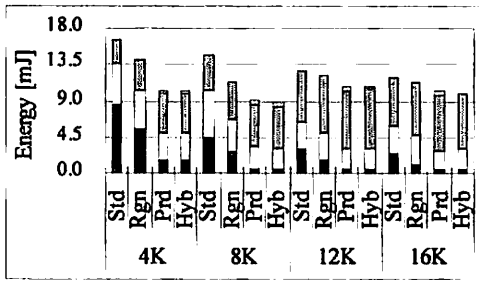
セット名	含まれるタスク	タスク数
tasksetA	bf, tiff2rgba	2
tasksetB	cjpeg, crc, qsort, tiff2rgba	4
tasksetC	bitcnts, cjpeg, ispell, rawcaudio, sha	6
tasksetD	bitcnts, bf, crc, dijkstra, ispell, qsort, rawcaudio, sha	8
tasksetE	bitcnts, bf, cjpeg, crc, dijkstra, ispell, qsort, rawcaudio, sha, tiff2rgba	10



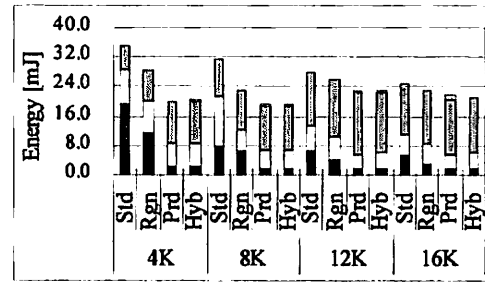
(a) tasksetA



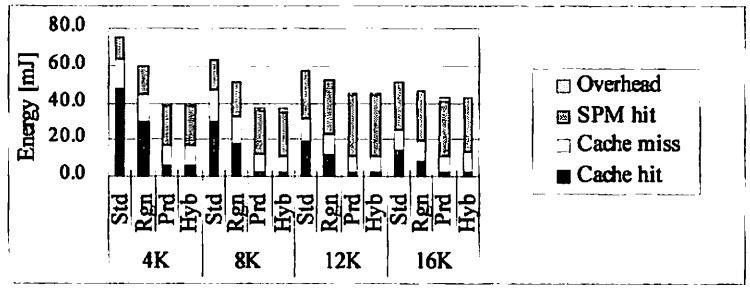
(b) tasksetB



(c) tasksetC



(d) tasksetD



(e) tasksetE

図 4 実験結果

'Std' は評価基準（各タスクに SPM 容量を均等に割り付けてからコード割当てを決定する方法）を適用したときの消費エネルギー量を示している．'Rgn'，'Prd'，および 'Hyb' は，本論文の提案手法である領域分割法，時間分割法，および混合分割法にそれぞれ対応している．横軸の 'xK' は，一次メモリ中の SPM の容量をあらわしている．棒グラフ中の "Cache hit"，"Cache miss"，および "SPM hit" は，キャッシュヒット，キャッシュミス，および SPM アクセスで消費されるエネルギーの総量を示している．"Cache miss" には，キャッシュラインの内容更新に掛かる主記憶アクセスのための消費エネルギーも含まれる．"Overhead" は，主記憶－ SPM 間コード伝送における消費エネルギーの総計である．

4.3 考　　察

実験結果に対する考察を述べる．

まず，ほとんど全ての状況において，提案手法によって消費エネルギーが小さくできていることがわかる．最大で，領域分割法では 27.25 %，時間および混合分割法では 47.18 %の消費エネルギー削減が達成できた．タスクセット，SPM 容量ごとの平均をとると，領域，時間，および混合分割法でそれぞれ，12.37 %，21.61 %，および 23.03 %の消費エネルギー削減となった．このことから，マルチタスクシステムの命令メモリにおける提案手法の有効性が確認できた．

次に，タスクの個数と提案手法との関係をみる．タスク数 10 の図 4(e) では，SPM 容量が小さく限られるとき，これらのタスクセットでは，実行状態のタスクが SPM の全領域を占有する時間および混合分割法の有効性は高くなる．さらに，SPM 容量が大きいときに，領域および時間分割法を同時に利用可能な混合分割法の消費エネルギーが小さくなる．タスク数 6 および 8 の図 4(c) および図 4(d) でも，ほぼ同様の傾向が得られている．このことから，タスク数が大きいとき，混合分割法は消費エネルギーの削減に最も適していることがわかる．

タスク数が少ない図 4(a) および図 4(b) では，SPM の容量が小さいときは領域分割法および混合分割法が有効に作用する．'tasksetA' のうちでは，8K SPM の領域分割法が，実験したメモリ構成および適用した提案手法のうちで，最も消費エネルギーが小さくなる．いっぽう，SPM の容量が大きくなると，これらのタスクセット実行時の消費エネルギー量は大きく増加してしまう．これは，タスク数 2 および 4 である 'tasksetA' および 'tasksetB' の総コードサイズが小さかったことによる．SPM が過容量となり，SPM からの命令フェッチや主記憶 - SPM 間データ伝送にかかる消費エネルギーが余分に掛かっている．

SPM の総容量と消費エネルギー量の関係に注目する．図 4 をみると，SPM の総容量を整数計画問題への入力である定数値として設定した本論文の提案手法では，どのタスクセットでも，SPM の容量が 8KBytes のときに消費エネルギー最小になっていることがわかる．これは，本論文で提案した SPM 領域分割方針を適用すると，SPM の容量を増やすことが必ずしも消費エネルギー削減に繋がらないことを示している．今回確認されたこの傾向は，消費エネルギー最小にできる SPM 容量を，変数として決定できる可能性を示唆している．

最後に，混合分割法に触れる．混合分割法は，領域および時間分割法の双方を活かした手法である．このため，提案する 3 種類の SPM 領域分割方針のうちで，混合分割法の消費エネルギー削減量が最も大きくなるはずである．しかし，図 4(d) における 4K SPM など，必ずしも全ての状況において混合分割法が消費エネルギー最小となっているわけではない．これは，整数計画問題の定式化の際に，キャッシュの動的な振る舞いを考慮しなかったためである．整数計画問題の定式化において用いた変数は，システム動作前の静的な解析によってすべて取得可能な値である．いっぽう，キャッシュ・ミス回数の計測には，システム実行中の動的な解析が必要である．しかし，注目すべきは，全タスクセットの全ての SPM 容量で，混合分割法によって消費エネルギーの安定的な削減を達成できていることである．

5. おわりに

本研究では，マルチタスク環境下での命令メモリにおける消費エネルギー削減を目標とした，SPM 領域分割方針を提案した．各方針について，タスクごとに使用する SPM 容量およびプログラムコード割当てを同時に決定可能な，整数計画問題に定式化した．実験により，提案手法の有効性を確認した．提案手法を適用することにより，タスクセット実行時の消費エネルギーを最大で 47.18 %削減することができた．タスク数が多いセットでは，時間および混合分割法の有効性が高くなった．さらに，混合分割法は，実験した全ての状況において安定的に消費エネルギーを削減できた．

今後の方針としては，まず，データメモリに対する評価実験が挙げられる．今回の実験では，プログラムコードのみを対象として SPM の有効性をみた．同様の実験を，データ変数に対しても行う必要がある．また，本研究の対象とするシステム構成および提案手法では，高優先度タスクによるプリエンプションを考慮していない．今後，プリエンプティブなマルチタスク環境でも適用可能な，SPM 領域分割手法および整数計画問題の定式化を提案する予定である．

謝辞 本研究を進めるにあたり，多くの助言をいただいた九州大学の石原亨准教授，名古屋大学の曾剛博士に深く感謝致します．本研究の一部は，科学技術振興事業団 (JST) 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統合化技術」の支援による．

文　献

[1] 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, 1996.

[2] O. Avissar and R. Barua, "An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems," *ACM Transactions on Embedded Computing Systems*, Vol. 1, No. 1, 2002.

[3] S. Steinke, et al., "Assigning Program and Data Objects to Scratchpad for Energy Reduction," *Proceedings of the conference on Design, Automation and Test in Europe*, 2002.

[4] R. Banakar, et al., "Scratchpad Memory: A Design Alternative for Cache Onchip memory in Embedded Systems," *International Symposium on Hardware/Software Codesign (CODES)*, 2002.

[5] F. Angiolini, L. Benini, and A. Caprara, "An Efficient Profile-Based Algorithm for Scratchpad Memory Partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 11, 2005.

[6] A. Janapsatya, A. Ignjatovic, and S. Parameswaran, "Exploiting Statistical Information for Implementation of Instruction Scratchpad Memory in Embedded System," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 14, No. 8, 2006.

[7] M. Verma, K. Petzold, L. Wehmeyer, H. Falk, and P. Marwedel, "Scratchpad Sharing Strategies for Multiprocess Embedded Systems: A First Approach," *IEEE 3rd Workshop on Embedded System for Real-Time Multimedia (ESTIMedia)*, 2005.

[8] M. R. Guthaus, et al., "MiBench: A free, commercially representative embedded benchmark suite", *IEEE International Workshop on Workload Characterization*, 2001.

[9] "SimpleScalar LLC," `http://www.simplescalar.com/`.

[10] "GLPK (GNU Linear Programming Kit)," `http://www.gnu.org/software/glpk/`.

[11] S. J. E. Wilton and N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid State Circuit*, Vol. 31, No. 5, 1996.

[12] "The Micron System Power Calculator," `http://www.micron.com/support/designsupport/tools/powercalc/powercalc` .