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

2008－ＳＥ－160 (8)
2008－EMB－ 9 (8)
2008／6／19

プリエンプティブな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分割法

高瀬英希† 冨山宏之† 高田広章†

本研究では，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．プリエンプティブな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．提案するスクラッチパッドメモリ領域分割方針は，空間分割法，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．

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

Hideki Takase,† Hiroyuki Tomiyama† and Hiroaki Takada†

In this paper, we propose three approaches to allocation of scratch-pad memory for 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.

1. はじめに

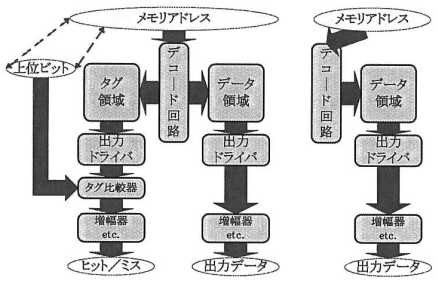
組込みシステムの消費エネルギー最適化は，近年の非常に重要な課題となっている．性能や計算精度を保証しながら消費エネルギーを削減することによって，製造コストや運用コスト，信頼性などといった数多くの意義深い利点が得られる．組込みシステムの高性能・高機能化に従う消費エネルギーの増大もまた，近年の大きな問題となっている．これに対応するため，組込みシステムの消費エネルギーを最適化するための研究が，現在までに盛んに行われている．

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，オンチップメモリとしてキャッシュが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ほど大きいという報告がなされている[1]．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，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 を有効に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．高いリアルタイム応答性の確保が求められる組込みシステムでは，プリエンプティブな優先度ベースのタスク・スケジューリング方式が一般的に採用される．本論文では，このプリエンプティブなマルチタスク環境において適用可能な，SPM 領域分割手法を提案する．提案する方針は，空間分割法，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．整数計画問題の最適解の導出により，タスクごとに使用する SPM 領域のメモリ量，および，SPM に割当てるプログラムコードの同時決定が可能である．

本論文の構成は以下のとおりである．まず，2 章では，関連研究に触れたのち，過去の研究と本論文との位置づけを明確にする．次に，3 章において提案手法の詳細を解説する．4 章では，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．

† 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University



(a) キャッシュ　　(b) SPM

図 1　キャッシュと SPM の回路構成

2. 関連研究と本研究の位置づけ

2.1 スクラッチパッドメモリ

SPM とは，小容量で高速なオンチップ SRAM のことを指す．**図 1(b)** に示すように，SPM はデコード回路，データ領域および出力器のみからなる．**図 1(a)** に示すキャッシュのタグ領域やタグ比較器は，SPM の回路構成に含まれない．

SPM のアクセス 1 回当たりの消費エネルギーは，タグ比較等の処理が不要となるため，キャッシュよりも小さく抑えられる．SPM においては，キャッシュミスのような状況が起きないため，プログラムの実行時間解析が容易となり，高いリアルタイム性を確保できるという利点もある．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回路中に持たない．このため，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．

2.2 関連研究

組込みシステムのメモリシステムとして SPM を適用する研究は，これまでに数多く行われている．

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 のアドレス空間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式化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，文献 3) の手法を適用することにより，SPM で一次メモリを構成するシステムでのプログラム実行時の消費エネルギーを，キャッシュのみのシステムよりも平均して約 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，文献 2)~5) における提案手法は，シングルタスク環境においてのみ適用可能な技術である．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，各タスクに SPM の容量を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（各タスク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となっている．また，文献 7) の手法では，消費エネルギー最小となる SPM 容量の分配は，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は，決定のための探索時間が膨大になる可能性がある．

2.3 本研究の位置づけ

高いリアルタイム応答性が要求される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はあまり用いられず，優先度に従うタスク・スケジューリング規則が一般的に採用される．組込みシステムにおける SPM の有効性を最大限に活かすためには，本規則によって処理されるマルチタスクシステムに対応できうる手法の確立が必須であると考えている．

われわれは，これまでに，文献 8) においてマルチタスク環境に対応し，かつ，消費エネルギー最小となる SPM 領域分割を一意に決定可能な手法を提案している．しかし，この手法は，非プリエンプティブなマルチタスク環境を仮定している．高優先度タスクによるプリエンプションを許可しない環境では，CPU 使用率が高い場合にデッドラインミスが頻発するおそれが高くなる．デッドライン制約が厳しいハードリアルタイムシステムでは，デッドラインミスの発生がシステム全体の致命的な信頼性低下に直結する．

本研究は，文献 8) の手法を，プリエンプティブなマルチタスク環境にも対応できるように拡張したものである．本拡張により，ハードリアルタイム・マルチ

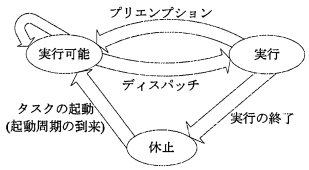


図 2 タスクの状態遷移

タスクシステムにおいても，有効性の高い SPM 領域の活用が可能になる．

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領域分割手法の詳細を述べる．提案する SPM 領域分割手法は，空間分割法，時間分割法，および混合分割法の 3 種類である．消費エネルギーの観点から各方針におけるタスクごとの SPM 領域分割およびコード割当ての決定は，消費エネルギー削減量の最大化を目的関数とした整数計画問題に定式化する．なお，本研究では，プログラムコードのみをメモリ領域への割当て対象とし，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 2** 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．タスクの優先度は，起動周期の短い順に高くなるとする．

タスクのスケジューリング規則は，レートモノトニック・スケジューリング方式[9] に従う．これは，プリエンプティブなマルチタスク環境において固定優先度を使用するもののうちで最適となることが知られているタスク・スケジューリング規則である．プリエンプティブなマルチタスク環境では，優先度の高いタスクが実行可能状態になったときに，より優先度の低いタスクが実行状態にあってもタスク切替えが発生する．あるタスクが実行状態であるときに，より高優先度のタスクが実行可能状態となった場合には，低優先度のタスクの処理は中断され，実行可能状態へと遷移する．低優先度タスクの実行は，高優先度タスクの処理が終了したのちに，中断時点から再開される．

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，システム動作前のタスクごとのプロファイル処理によってすべて取得可能である．

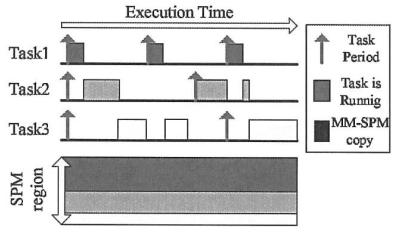


図 3 空間分割法

3.3 空間分割法

空間分割法は，SPM の容量を固定的に分配して各タスクに与える手法である（**図 3**）．各タスクは，与えられた SPM 領域を占有して使用する．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ない．例えば，タスク数 3 の図 3 においては，SPM のアドレス空間を 3 つに分割して各タスクに与えている．

空間分割法では，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．周期が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 3 では，最高優先度の Task1 は SPM 領域をより多く占有している．

各タスクに与える SPM の容量と，タスクごとの SPM 領域への関数割当ての決定は，次に示す整数計画

表 1 変数の定義

変数名	定義
$task_i$	i 番目のタスク．$1 \leq i \leq M$
$period_i$	$task_i$ の起動周期
$func_{i,j}$	$task_i$ の j 番目の関数．$1 \leq j \leq N$
$fetch_{i,j}$	$func_{i,j}$ の命令フェッチ総数
$size_{i,j}$	$func_{i,j}$ のコードサイズ
E_{gain}	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_{C_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_{S_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_{S_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

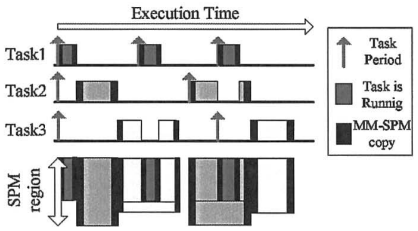


図 4 時間分割法

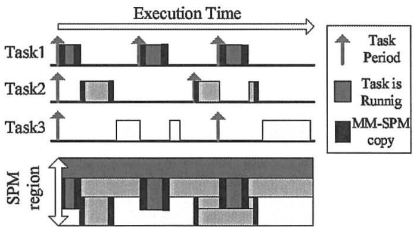


図 5 混合分割法

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 $period_i$ を付与することにより，SPM の活用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．

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

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

$$E_{gain} = E_{C\_read} - E_{S\_read}$$

$$\textbf{s.t.}: \sum_i SPMsize\_rgn_i \leq SPMsize$$

$$SPMsize\_rgn_i = \sum_j size_{i,j} \times x_{i,j}$$

ここで，式中の $SPMsize\_rgn_i$ は，$task_i$ に分割して与えられる SPM 領域の容量をあらわす決定変数である．0-1 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 をとる．これらの解集合を求めることにより，タスクごとの SPM 容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．

3.4 時間分割法

図 4 に示すように，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．ただし，タスクがある起動周期内で初めて実行状態に遷移するときと，実行を終了し休止状態に遷移するときの計 2 回，主記憶–SPM 間のプログラムコード転送を行う必要がある．実行開始時は，タスクに含まれるコードのうちから実行頻度の高いものを主記憶から SPM へと転送する．タスクの実行終了時には，主記憶–SPM 間のコード転送を行うことによって SPM の保持する内容を開始前の状態に戻す．図 4 では，'MM-SPM copy' の部分において主記憶–SPM 間転送を処理している．SPM 領域に割当てられる関数の決定には，コード転送にかかる消費エネルギーのオーバヘッドを考慮する必要がある．このため，オーバヘッドが大きい関数を多く含むタスクは，SPM の全容量を使いきるとは限らない．

時間分割法における SPM 領域への関数割当てを定式化した整数計画問題を，以下に示す．計 2 回の主記憶–SPM 間転送のオーバヘッド $Eoverhead_{i,j}$ を加味した上で，$Esaving$ の最大化を目指している．式中の 0-1 決定変数 $y_{i,j}$ の解集合を求めることによって，各タスクの関数割当てが決定できる．

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

$$Esaving_{i,j} = fetch_{i,j} \times E_{gain} - Eoverhead_{i,j} \times 2$$

$$E_{gain} = E_{C\_read} - E_{S\_read}$$

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

$$\textbf{s.t.}: \forall i.\ \ SPMsize\_prd_i \leq SPMsize$$

$$SPMsize\_prd_i = \sum_j size_{i,j} \times y_{i,j}$$

3.5 混合分割法

混合分割法は，上記 2 つの SPM 領域分割法を併用する手法である．**図 5** に示すように，空間分割と時間分割の両手法を組み合わせ，SPM 領域の柔軟な活用を目指す．

混合分割法における高優先度タスクは，空間分割によって与えられる領域に加え，より優先度の低いタスクの使用する領域を時間分割領域として使用することができる．ただし，"横取り" して使用した領域は，その低優先度タスクの実行再開に備え，高優先度タスクの実行終了後に内容を元に戻す必要がある．つまり，高優先度タスクが SPM 領域を横取りする場合には，時間分割法と同様に，主記憶–SPM 間の 2 回分にあたる消費エネルギーのオーバヘッドが掛かる．最低優先度のタスクは，空間分割法による領域しか与えられない．例えば，図 5 では，Task1 は Task2 および Task3 の領域を，Task2 は Task3 の領域を横取りして使用している．最低優先度の Task3 は，みずからの空間分割領域しか使用しない．

タスクごとの空間分割領域，および，高優先度タスクが横取りして時間分割領域として使用する SPM 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，それぞれ決定される．あるタスクが使用できる SPM の容量は，空間分割法によって分配される領域と，時間分割法として用い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，以下に示す．

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

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

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

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

$$Esaving\_prd_{i,j} = (fetch_{i,j} \times E_{gain} - Eoverhead_{i,j} \times 2) \times \frac{hyperperiod}{period_i}$$

$$E_{gain} = E_{C\_read} - E_{S\_read}$$

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

$$\textbf{s.t.}: \sum_i SPMsize\_rgn_i \leq SPMsize \quad (1)$$

$$SPMsize\_rgn_i = \sum_j size_{i,j} \times x_{i,j}$$

$$\textbf{s.t.}: \forall i. \sum_j size_{i,j} \times y_{i,j} \leq SPMsize\_prd_i \quad (2)$$

$$\exists k,\ period_k > period_i\ .$$

$$SPMsize\_prd_i = SPMsize - \sum_k SPMsize\_rgn_k$$

$$\textbf{s.t.}: \forall i,\ \forall j.\ x_{i,j} + y_{i,j} \leq 1 \quad (3)$$

本式の決定変数は，$SPMsize\_rgn_i$，$x_{i,j}$ および $y_{i,j}$ である．$SPMsize\_rgn_i$ は，各タスクに与えられる空間分割領域の SPM 容量をあらわしている．この変数値は，制約式 (1) によって決定される．制約式 (2) は，高優先度タスクが時間分割領域として使用する低優先度タスクの SPM 容量を決定している．0-1 変数 $x_{i,j}$ は，$func_{i,j}$ が $task_i$ の空間分割領域に割当てられるときに 1 となる．$y_{i,j}$ は時間分割領域のそれに対応する 0-1 変数である（制約式 (3)）．これらの決定変数の解集合を求めることにより，空間分割法による SPM 領域中で各タスクに割り付けられる SPM の容量，高優先度タスクが時間分割領域として用いる 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．

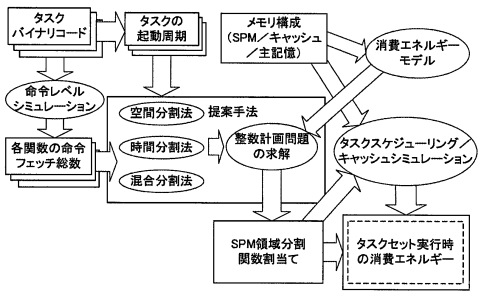


図 6　実験のワークフロー

実験には，ベンチマークスィート MiBench[10] から 8 個のプログラムを選定し，**表 2** に示す 4 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[11] 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命令セットシミュレータである．

図 6 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中の関数ごとのメモリ量および実行命令数を取得した．起動周期は，各タスクの CPU 使用率が 10 %程度となるように設定した．例えば，タスク数 6 の tasksetC では，タスクセット全体の CPU 使用率が約 60 %となる．このため，タスク数が多いタスクセットは，システム動作中の CPU 使用率が高くなる．

これらの処理を経て得られる入力情報に対して，提案する 3 種類の SPM 領域分割手法を適用した．SPM の領域分割やコード割当て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[12] を使用した．一次メモリは，8KBytes 4-way のキャッシュに，1K / 2K / 4K / 8KBytes 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

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	cjpeg, stringsearch	2
tasksetB	tasksetA + rawcaudio, sha	4
tasksetC	tasksetB + ispell, susan	6
tasksetD	tasksetC + djpeg, qsort	8

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．これらの出力値を基にし，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，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，一次メモリであるキャッシュおよび SPM には CACTI 4.2[13] を，外部主記憶には Micron System Power Calculator[14] をそれぞれ用いた．

これまでに，レートモノトニック・スケジューリング方式に従う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 7 に，実験結果を示す☆．グラフの縦軸は，該当タスクセットの実行中にメモリシステム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導出した．横軸の 'xk' は，一次メモリにおける SPM の容量をあらわしている．'Std' は評価基準（各タスクに SPM 容量を均等に分配してからコード割当てを決定する方法）を適用したものであり，'Rgn'，'Prd'，および 'Hyb' は，本論文の提案手法である空間分割法，時間分割法，および混合分割法にそれぞれ対応している．図 7 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量の合計をあらわす．'Cache miss' には，キャッシュ・ラインの内容を更新するために主記憶アクセスを行うときに消費されるエネルギーも含まれている．'Overhead' は，時間および混合分割法でのタスク切替え時に発生する，主記憶–SPM 間コード転送時のメモリアクセスに掛かる消費エネルギー量をあらわしている．

4.3 考　　察

まず，ほとんど全ての状況において，提案手法によってタスクセット実行中の消費エネルギーが小さくできていることがわかる．評価基準と比較して，最大で，空間分割法では 69 %，時間分割法および混合分割法では 71 %の消費エネルギーをそれぞれ削減している．このことから，提案手法の適用によって，プリエンプティブなマルチタスクシステムの消費エネルギーを削減できることが確認できた．

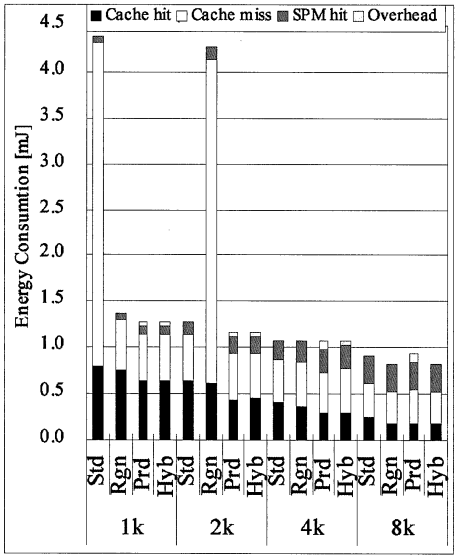
タスク数の多いセットをみる．タスク数 8 の**図 7(d)** では，SPM 領域の内容を動的に変更する時間および混合分割法が有効となる．平均して，時間分割法では 48 %，混合分割法では 42 %の消費エネルギー削減を達成している．タスク数 6 の**図 7(c)** でも，同様の傾向が得られている．このことから，タスク数が大きい場合にとくに，提案手法が有効となることがわかる．

スケジューリング可能性について触れる．本実験と同等のメモリ構成およびタスク起動周期において，タスク数 8 の tasksetD を非プリエンプティブな固定優先度ベースのタスク・スケジューリング方式によってスケジューリングしたところ，タスクセット実行中にデッドラインミスが頻発した．いっぽう，プリエンプションを考慮した本研究での実験環境では，tasksetD のシステム動作中の CPU 使用率は平均で 79.7 %と高いものの，デッドラインミスは発生しなかった．プリエンプティブなマルチタスク・リアルタイムシステムで SPM を有効に活用できる方針を示したこともまた，本論文の大きな貢献である．

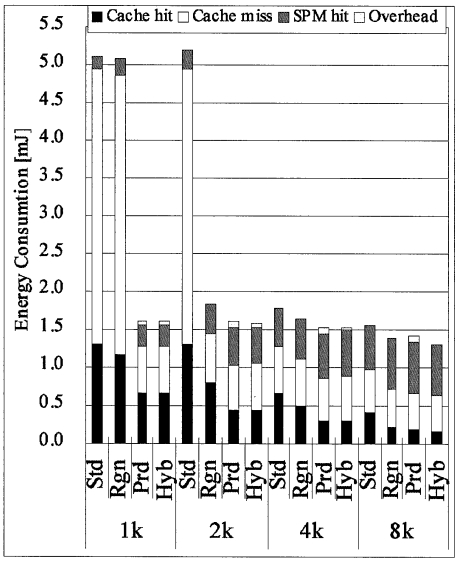
図 7(a) における 2k SPM や，**図 7(c)** における 8k SPM などでは，評価基準 'Std' よりも空間分割法 'Rgn' の消費エネルギーが大きくなっている．これは，空間分割法におけるキャッシュ参照回数は評価基準のときよりも少ないにも関わらず，キャッシュの競合性ミスが大幅に増加したためである．本研究において定式化した整数計画問題は，プログラムコードの実行頻度を指標として用いており，命令メモリのアクセスパターンは考慮されていない．このことから，提案する SPM 領域分割法の有効性をより高めるためには，メモリアクセスパターンを考慮した定式化の確立が必要であると考えられる．

最後に，時間分割法と混合分割法を比較する．時間分割法では，SPM 容量が大きいとき，主記憶–SPM 間コード転送にかかる消費エネルギーのオーバヘッドが大きくなる．とくに，図 7(a) における 8k SPM の時間分割法では，オーバヘッドの消費エネルギー量が大きいために，システム全体の消費エネルギーが "Std" や "Rgn" よりも大きくなる．いっぽう，混合分割法では，主記憶–SPM 間コード転送のオーバヘッドが，

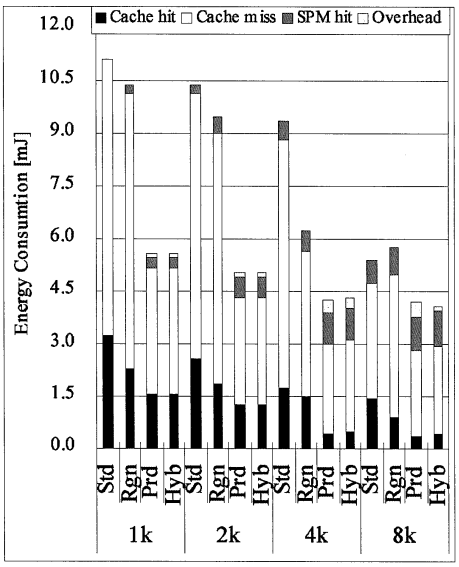
☆ tasksetA の 1k SPM における混合分割法では，今回用いた ILP ソルバで整数計画問題の最適解が導出できなかった．このため，制約式を緩和した上での最適解による値を記載している．



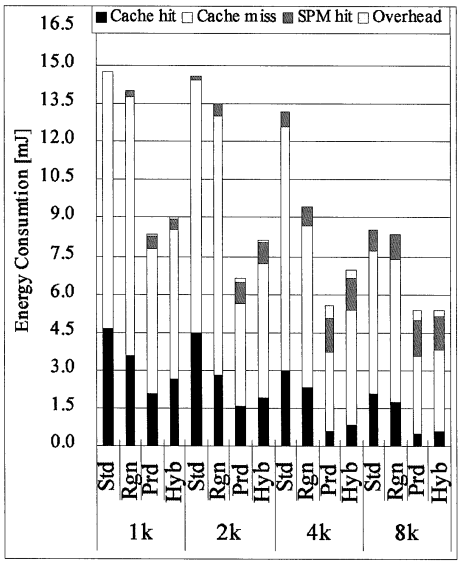
(a) tasksetA



(b) tasksetB



(c) tasksetC



(d) tasksetD

図 7 実験結果

時間分割法よりも小さく抑えられている．tasksetD の 8k SPM における混合分割法は，実験を行ったメモリ構成および SPM 領域分割手法のうちで，消費エネルギーが最も小さくなる．これは，SPM 容量が大きいときには，主記憶–SPM 間コード転送が不要である空間分割領域の SPM 容量が，各タスクに十分に与えられるためである．以上のことから，混合分割法は，SPM 容量が大きいときに，より有効性が高くなる．

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，3 種類の SPM 領域分割方針を提案した．空間分割法は，それぞれのタスクに使用できる SPM 領域を静的に分割して与える手法である．タスクの起動周期を基にすることにより，高優先度タスクのプログラムコードが SPM 領域により多く割当てることができる．時間分割法は，実行状態のタスクが SPM の全領域を独占して用いる．実行タスクの切替え時に，主記憶–SPM 間データ転送を計 2 回行い，SPM の保持する内容を変更する．混合分割法は，空間および時間分割法の双方を同時に活用する手法である．このことより，SPM 領域の柔軟な活用が可能である．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 71 %削減することができた．とくに，タスク数が多いセットでは，主記憶–SPM 間転送を行う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，SPM 容量が大きいときに，主記憶–SPM 間転送に掛かる消費エネルギーが抑えられ，全体の消費エネルギーを小さくすることができた．

今後の方針としては，データメモリに対する評価，および，メモリアクセスパターンを考慮した SPM 領域分割の探索が挙げられる．

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および横山哲郎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．

参考文献

1) S Segars, "Low Power Design Techniques for Microprocessors," *IEEE International Solid-State Circuits Conference (Tutorial)*, 2001.
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, et al., "Scratchpad Sharing Strategies for Multiprocess Embedded Systems: A First Approach," *IEEE 3rd Workshop on Embedded System for Real-Time Multimedia (ESTIMedia)*, 2005.
8) 高瀬 英希, 冨山 宏之, 高田 広章. "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". 組込技術とネットワークに関するワークショップ (ETNET) 2008.
9) C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of ACM*, Vol. 20, No. 1, 1973.
10) M. R. Guthaus, et al., "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", *IEEE International Workshop on Workload Characterization*, 2001.
11) "SimpleScalar LLC," `http://www.simplescalar.com/`.
12) "GLPK (GNU Linear Programming Kit)," `http://www.gnu.org/software/glpk/`.
13) 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.
14) "The Micron System Power Calculator," `http://www.micron.com/support/designsupport/tools/powercalc/powercalc`.