組込みプロセッサのエネルギー消費を最小化するコード配置最適化問題のILPモデル

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：電子情報通信学会技術研究報告, VLD2007-75. 107 (334), pp.31-36, 2007-11. 電子情報通信学会
バージョン：
権利関係：

社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

組込みプロセッサのエネルギー消費を最小化するコード配置最適化問題のILPモデル

石飛　百合子† 　石原　亨†† 　安浦　寛人†††

† 九州大学大学院システム情報科学府　〒819–0395 福岡県福岡市西区元岡 744 番地
†† 九州大学システム LSI 研究センター　〒814–0001 福岡県福岡市早良区百道浜 3–8–33
††† 九州大学大学院システム情報科学研究院　〒819–0395 福岡県福岡市西区元岡 744 番地
E-mail: †ishitobi@c.csce.kyushu-u.ac.jp, ††ishihara@slrc.kyushu-u.ac.jp, †††yasuura@c.csce.kyushu-u.ac.jp

あらまし　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の総消費エネルギーを最適化するコード配置問題の定義および定式化を行う．筆者らはキャッシュを用いたアクセスを行う cacheable 領域，SPM を用いたアクセスを行う scratchpad 領域およびキャッシュを用いずにオフチップへアクセスを行う non-cacheable 領域の 3 領域へのコード配置を，消費エネルギー最小を目的として決定する手法の提案している．本稿では，提案するコード配置手法におけるコード配置決定問題を定式化し，整数計画問題への変換を行った．
キーワード　低消費エネルギー化　組込みプロセッサ　コード配置　スクラッチパッドメモリ

An ILP Model of Code Placement Problem for Minimizing the Energy Consumption in Embedded Processors

Yuriko ISHITOBI†, Tohru ISHIHARA††, and Hiroto YASUURA†††

† Graduate School of Inf. Sci. & E.E.,Kyushu University
Motooka 744, Nishi-ku, Fukuoka-shi, 819–0395 Japan
†† System LSI Research Center,Kyushu University
Momochihama 3–8–33, Sawara-ku, Fukuoka-shi, 814–0001 Japan
††† Faculty of Inf. Sci. & E.E.,Kyushu University
Motooka 744, Nishi-ku, Fukuoka-shi, 819–0395 Japan
E-mail: †ishitobi@c.csce.kyushu-u.ac.jp, ††ishihara@slrc.kyushu-u.ac.jp, †††yasuura@c.csce.kyushu-u.ac.jp

Abstract　This paper formulates a code placement problem to optimize the total energy consumption of a CPU core, on-chip memories and off-chip memories in embedded systems. We proposed a code placement method to minimize the total energy consumption considering a cacheable region, the scratchpad region and the non-cacheable region. The cacheable region uses cache memory at access, the scratchpad region uses SPM and the non-cacheable region accesses off-chip memory directly. The code placement problem to three region is formulated and proposeed the ILP model in this paper.
Key words　energy reduction, embedded processor, code placement, scratchpad memory

1. はじめに

組込み用途向けマイクロプロセッサの消費エネルギーの内訳において，オンチップメモリの消費エネルギー占める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．キャッシュメモリサイズが限定されている場合であっても，キャッシュメモリへのアクセス数およびキャッシュミス数を削減することで，キャッシュメモリで消費するエネルギーを削減することは可能である．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の提案を行った上で，提案手法を用いる際のコード配置最適化問題の定義および定式化を行う．

提案するコード配置手法は 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の考慮を行う．キャッシュを用いずにアクセスを行う領域には，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 (以下 non-cacheable 領域) の 2 種類を想定している．3 つの領域へのコード配置を同時に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．本稿では消費エネルギー最小を目的として 3 領域へのコード配置を決定する問題を定式化し，その ILP モデルの提案を行う．

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問題を定義し，その定式化を行う．4 章では 3 章において定式化したコード配置問題を整数計画問題に変換する．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 $C(=2^m\text{word})$，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．

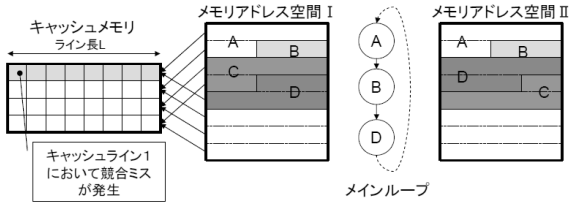

図 1　コード配置例

2.1.2 SPM への静的なコード配置

キャッシュは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オンチップメモリである，一方で 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オンチップメモリである．SPM はハードウェアによるデータ転送のサポートを行わないため，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]．

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

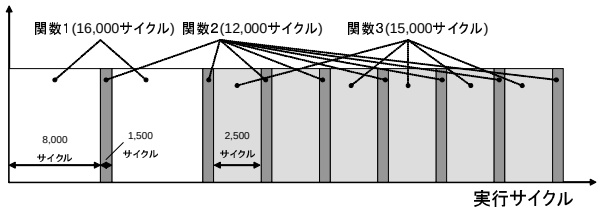

図 2　Motivation example

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

参考文献 [19] において，筆者らは低消費エネルギー化を目的としたコード配置手法を提案している．

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 の利用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．提案手法の利点を図 2 に示すキャッシュへのアクセストレースを用いて示す．仮定として，アクセスとレース中で参照される関数 1，関数 2 および関数 3 はすべて同サイズで 2KB であるとする．またオンチップメモリとして 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．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，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，消費エネルギーが削減される場合は non-cacheable 領域への配置を行うことも可能である．

提案するコード配置手法は，cacheable 領域，scratchpad 領域および 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アクセスあたりの消費エネルギー削減に加え，scratchpad 領域へのアクセスがキャッシュを用いないことを考慮し，scratchpad 領域へのコード配置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ことで，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 non-cacheable 領域を

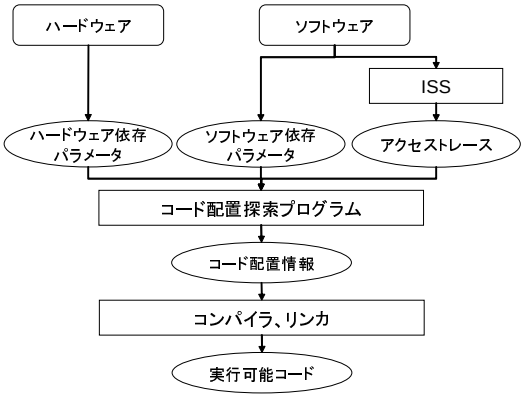

図 3　コード配置手法のフロー

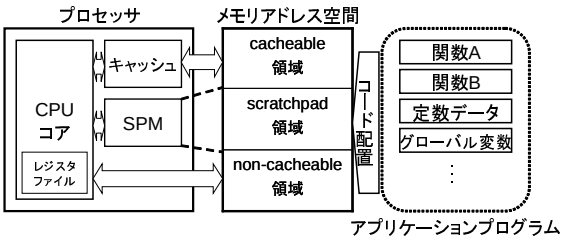

図 4　提案手法におけるコード配置概念図

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 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，実行可能コードに変換することができる．

3. 問 題 定 義

本章では，提案手法におけるコード配置決定問題の定義および定式化を行う．

3.1 提案手法におけるコード配置

図 4 に提案手法におけるコード配置の概念図を示す．本稿におけるコード配置は，アプリケーションプログラム内の関数，グローバル変数および定数データを含むメモリオブジェクト単位で配置の決定を行う．各メモリオブジェクトは cacheable 領域，scratchpad 領域および non-cacheable 領域の 3 領域のいずれかに配置される．cacheable 領域に配置される場合は，配置されるアドレス領域がキャッシュミス数に影響を及ぼすため，cacheable 領域内での配置順序の指定も行う．

3.2 消費エネルギー見積り関数

目的関数として，式 (2) に示す消費エネルギー見積り関数を使用する．

$$TE_{total} = N_{cache} \cdot E_{cache} + N_{miss} \cdot E_{miss} + N_{wb} \cdot E_{wb} + N_{spm} \cdot E_{spm} + (\alpha \cdot N_{miss} + \alpha \cdot N_{wb} + N_{offs}) E_{off} + (P_{off} + P_{logic}) \cdot t_{all} \quad (2)$$

$$t_{all} = CT(N_{inst} \cdot CN_{inst} + N_{miss} \cdot CN_{miss} + N_{offs} \cdot CN_{offs} + N_{wb} \cdot CN_{wb}) \quad (3)$$

式 (2) において TE_{total} は CPU コア，オンチップメモリおよびオフチップメモリの総消費エネルギー値を示す．TE_{total} を計算する式において用いられるパラメータの説明を行う．N_{cache}，N_{miss} および N_{wb} は各々キャッシュへのアクセス回数，キャッシュミス数およびキャッシュからのライトバック数を示す．N_{spm} は SPM へのアクセス回数を示す．N_{offs} はオフチップメモリへのシングルアクセスによるアクセス回数を示す．E_{cache}，E_{miss} および E_{wb} は各々キャッシュへのアクセスあたりに必要なエネルギー，ミス処理あたりにキャッシュで消費されるエネルギーおよびライトバック処理あたりにキャッシュで消費されるエネルギーを示す．E_{spm} および E_{off} は各々SPM でアクセスあたりに消費されるエネルギーおよびオフチップメモリへのアクセスあたりに消費されるエネルギーを示す．P_{logic} および P_{off} は各々ロジック部の平均消費電力およびオフチップメモリで静的に消費される電力を表す．α はキャッシュミスおよびライトバックあたりでのオフチップメモリへのアクセス回数を示す．t_{total} は実行時間である．式 (3) おいて CT はクロックサイクル時間を示し，N_{inst} は実行命令数を示す．CN_{inst}，CN_{miss}，CN_{offs} および CN_{wb} は各々1 命令実行あたりに必要なサイクル数，ミス処理あたりに必要なサイクル数，シングルアクセスによるオフチップメモリへのアクセスに必要なサイクル数およびライトバック処理に必要なサイクル数を示す．

目的関数中のパラメータにおいて，コード配置に影響を受けるパラメータは N_{cache}，N_{spm}，N_{offs}，N_{miss} および N_{wb} のみであり，他のパラメータは回路シミュレータ，回路シミュレータおよび電力解析ツールを用いた測定により得られる．

3.3 問題の定式化

コード配置問題の定式化に用いるパラメータを以下に示す．

N_{obj}	メモリオブジェクト数
N_{mem}	メモリ空間内の全メモリブロック数
N_{way}，N_{set}	キャッシュのウェイ数およびセット数
S_{spm}	scratchpad 領域のメモリブロック数
o_i	メモリオブジェクト i $(i = 0, 1, 2 \ldots N_{obj})$
b_i	o_i のメモリブロック数
acc_i	実行時の o_i へのアクセス回数

メモリブロックとはキャッシュとオフチップメモリの入れ替えの単位であり，キャッシュのラインサイズの大きさのブロックである．

各 o_i の配置を示す変数 x_i を以下のように定義する．

$$\begin{aligned} & x_i \in \{0, 1, 2, \ldots, N_{obj} - 1, A_{spm}, A_{nc}\} \\ & i \neq i' \text{ and } x_i \neq A_{spm} \Rightarrow x_i \neq x_{i'} \\ & i \neq i' \text{ and } x_i \neq A_{nc} \Rightarrow x_i \neq x_{i'} \\ & A_{spm} > N_{mem}, A_{nc} > N_{mem}, A_{spm} \neq A_{nc} \end{aligned} \quad (4)$$

$$X = (x_0, x_1, x_2, \ldots, x_{N_{obj}})$$

X は各 x_i のベクタである．x_i の値は，A_{spm} のとき scratchpad 領域に配置されていることを示し，A_{nc} のとき non-cacheable 領域に配置されていることを示す．N_{obj} より小さい値をとる場合は，cacheable 領域の中で x_i 番目に配置されることを意味する．

目的関数中の N_{cache}，N_{spm} および N_{offs} を求めるために，$c(x_i)$，$s(x_i)$ および $u(x_i)$ を以下のように定義する．

$$c(x_i) = \begin{cases} 1 & \text{if } x_i \leqq N_{obj} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

$$s(x_i) = \begin{cases} 1 & \text{if } x_i = A_{spm} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$u(x_i) = \begin{cases} 1 & \text{if } x_i = A_{nc} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

N_{cache}，N_{spm} および N_{offs} の値は以下の式 (8)，(9) および (10) を用いて計算できる．

$$N_{cache} = \sum_{i=1}^{N_{obj}} c(x_i) \cdot acc_i \quad (8)$$

$$N_{spm} = \sum_{i=1}^{N_{obj}} s(x_i) \cdot acc_i \quad (9)$$

$$N_{offs} = \sum_{i=1}^{N_{obj}} u(x_i) \cdot acc_i \quad (10)$$

scratchpad 領域へのコード配置には，配置可能な領域のサイズに制約があるため，以下の式が示すように制限される．

$$\sum_{i=1}^{N_{obj}} s(x_i) \cdot b_i \leqq S_{spm} \quad (11)$$

N_{miss} および N_{wb} の計算には，cacheable 領域へのアクセストレースを用いる必要がある．アプリケーションプログラムを実行して得たアクセスアドレストレースより，メモリオブジェクトへのアクセストレースを得ることができる．メモリオブジェクト o_i の中の j 番目のメモリブロックへのアクセスを $p_{i,j}$ とすると，アクセストレース TR は以下のように得られる．

$$p_{0,0}, p_{0,1}, p_{0,2}, p_{2,0}, p_{2,1}, p_{5,0}, \ldots, p_{3,0}, p_{3,1}$$

各アクセス $p_{i,j}$ はリードまたはライトアクセスのいずれかである．

各メモリオブジェクトに割当てられたアドレス値は，$x_i (i = 0, 1, 2 \ldots N_{obj})$ を用いて式 (12) により計算できる．

$$bn(i, j) = \sum_{i'=1}^{N_{obj}} y(x_i, x_{i'}) \cdot b_{i'} + j \quad (12)$$

式 (12) 中の $y(x_i, x_{i'})$ は以下のように与えられる．

$$y(x_i, x_{i'}) = \begin{cases} 1 & x_i > x_{i'} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

N_{miss} および N_{wb} をカウントするために，新たに $a_{i,j,l}$ および $w_{i,j,l}$ を定義する．TR 中の各 $p_{i,j}$ に着目し，以下に示す $a_{i,j,l}$ を得る．

$$a_{i,j,l} = \begin{array}{l} \{p_{i',j'} | p_{i',j'} (i \neq i' or j \neq j') \text{appear between the} \\ l\text{th appearance and } (l+1)\text{th appearance of } p_{i,j} \\ \text{in } TR\} \end{array}$$

ライトアクセスである $p_{i,j}$ に着目し，以下のような $w_{i,l,k}$ を得る．

$$w_{i,j,l} = \begin{cases} \phi & : l \text{ th } p_{i.j} \text{ is read} \\ \{a_{i,j,k} | a_{i,j,k} \text{appear between} \\ \text{the write access of } l \text{ th } p_{i,j} \\ \text{and the next write access of} \\ p_{i,j} (k = l, l+1, l+2 \ldots)\} & : l \text{ th } p_{i,j} \text{ is write} \end{cases}$$

N_{miss} および N_{wb} は式 (14) および式 (15) により計算される．

$$N_{miss} = \sum_{\forall a_{i,j,l}} replace(a_{i,j,l}) + Constant \tag{14}$$

$$N_{wb} = \sum_{\forall w_{i,j,l}} writeback(w_{i,j,l}) \tag{15}$$

式 (14) において用いられる $replace(a_{i,j,l})$ は，キャッシュミスをカウントする式である．$a_{i,j,l}$ はトレース中で l 番目の $p_{i,j}$ から $l+1$ 番目の $p_{i,j}$ の間に出現する $p_{i',j'}$ の集合を示す．$replace(a_{i,j,l})$ は $a_{i,j,l}$ が包含する $p_{i',j'}$ へのアクセス中に，$p_{i,j}$ を格納しているラインにおいてキャッシュミスが発生する場合は 1，無ければ 0 の値をとる．$a_{i,j,l}$ の中に，$p_{i,j}$ とアクセスが競合する $p_{i',j'}$ がキャッシュのウェイ数 N_{way} 以上存在する場合，キャッシュミスが発生する．

$$repalce(a_{i,j,l}) = \begin{cases} 1 & \sum_{p_{i',j'} \in a_{i,j,l}} conflict(p_{i,j}, p_{i',j'}) \geqq N_{way} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$writeback(w_{i,j,l})$ は式 (17) で計算される．$w_{i,j,l}$ にはトレース中で，l 番目の $p_{i,j}$ のライトアクセスから $l+1$ 番目の $p_{i,j}$ へのライトアクセスの間に存在する $a_{i,j,k}$ を保持している．ある $p_{i,j}$ へのライトアクセスから次の $p_{i,j}$ へのライトアクセスの間に $p_{i,j}$ が追い出されるキャッシュミスが発生した場合，ライトバックが発生する．

$$writeback(w_{i,j,l}) = \begin{cases} 1 & \sum_{a_{i,j,k} \in w_{i,j,l}} replace(a_{i,j,k}) > 0 \\ 0 & \text{otherwize} \end{cases} \tag{17}$$

式 (16) で用いられる $conflict(p_{i,j}, p_{i',j'})$ の値は式 (18) により求められる．$conflict(p_{i,j}, p_{i',j'})$ は $p_{i,j}$ と $p_{i',j'}$ のアクセスブロックが，同じキャッシュラインにマッピングされるときに 1 の値をとる．

$$conflict(p_{i,j}, p_{i',j'}) = \begin{cases} 1 & \text{if } x_i \neq A_{spm},\ x_i \neq A_{nc} \text{ and} \\ & (bn(i,j) \bmod N_{set}) \\ & = (bn(i',j') \bmod N_{set}) \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

消費エネルギーが最小となるコード配置を求める問題は「TE_{total} を最小とする $X = (x_0, x_1, x_2 \ldots, x_{N_{obj}})$ を求める問題」として定式化される．

3.4 定式化の検証

前節で示したキャッシュミス数およびライトバック数の計算を行う定式の検証を行う．命令セットシミュレータにより得た実アプリケーションのアクセストレースを元に $a_{i,j,l}$ および $w_{i,j,l}$ を生成する．検証に用いたアプリケーションプログラムは jpeg, mpeg2 および compress である．実行開始より 100 万命令実行した際のアクセストレースを用いた．式 (14)-(18) を C 言語を用いて実装し，$a_{i,j,l}$ および $w_{i,j,l}$ を入力として与えキャッシュミス数およびライトバック数のカウントを行った．

式 (14)-(18) を実装したプログラムを用いて得た結果と，キャッシュシミュレータにアクセストレースを与えて得た結果を比較し，式 (14)-(18) により正しくキャッシュミス数およびライトバック数がカウントされていることを確認した．

4. 整数計画問題への定式化

前章で示した定式化では，目的関数および制約条件が線形化されていない．本章では 3 章で示した数式を線形に変換することで，提案手法におけるコード配置問題を整数計画問題として定式化を行う．

$c(x_i)$ の値を保持する変数を c_i とすると，式 (5) は以下のように変換できる．

$$\begin{gathered} (N_{obj} - x_i) + (1 - c_i) \cdot U \geqq 0 \\ (N_{obj} - x_i) - c_i \cdot U < 0 \\ c_i \in \{0, 1\} \end{gathered} \tag{19}$$

$s(x_i)$ の値を保持する変数を s_i とすると，式 (6) は以下のように変換される．

$$\begin{gathered} x_i - A_{spm} - (1 - s_i) \cdot U \leqq 0 \\ x_i - A_{spm} + (1 - s_i) \cdot U \geqq 0 \\ x_i - A_{spm} + s_i \cdot U \neq 0 \\ s_i \in \{0, 1\} \end{gathered} \tag{20}$$

加えて s_i について式 (11) で示される制約は以下のように変換される．

$$\sum_{i=1}^{n_{obje}} s_i \cdot b_i \leqq S_{spm} \tag{21}$$

$u(x_i)$ は値を保持する変数を u_i とすると，s_i および A_{spm} を u_i および A_{nc} に変換することで式 (20) と同様に表すことができる．

式 (4) で示した x_i の制約条件を線形の制約式に変換する．x_i

の制約は以下のように記述できる．

$$(1 - c_i) + (x_i - x_{i'}) \neq 0 \tag{22}$$

式 (13) は $y(x_i, x_{i'})$ の値を保持する変数を $y_{i,i'}$ とすると，以下の式で書き換えることができる．

$$\begin{gathered}(x_i - x_{i'}) + (1 - y_{i,i'}) \cdot U > 0 \\ (x_i - x_{i'}) - y_{i,i'} \cdot U \leqq 0 \\ y_i \in \{0, 1\}\end{gathered} \tag{23}$$

次に式 (18) の変換を行う．$conflict(p_{i,j}, p_{i'.j'})$ の値を保持する変数を $v_{(i,j),(i',j')}$ とし，$bn(i,j)$ の値を $d_{i,j}$ とする．$z_{(i,j),(i',j')}$ が $(d_{i,j} - d_{i',j'})/N_{set}$ の値を保持するとすると，式 (18) は以下のように変換することができる．

$$\begin{gathered}0 \leqq (d_{i,j} - d_{i',j'}) - N_{set} \cdot z_{(i,j),(i',j')} < N_{set} \\ (d_{i,j} - d_{i',j'}) - N_{set} \cdot z_{(i,j),(i',j')} + v_{(i,j),(i',j')} \cdot U \neq 0 \\ (d_{i,j} - d_{i',j'}) - N_{set} \cdot z_{(i,j),(i',j')} - (1 - v_{(i,j),(i',j')}) \leqq 0 \\ x_i - N_{mem} - (1 - v_{(i,j),(i',j')}) \cdot U < 0 \\ x_{i'} - N_{mem} - (1 - v_{(i,j),(i',j')}) \cdot U < 0 \\ v_{(i,j),(i',j')} \in \{0, 1\} \\ z_{(i,j),(i',j')} \in \mathbf{Z}\end{gathered} \tag{24}$$

式 (16) は $replace(a_{i,j,l})$ の値を保持する変数を $e_{i,j,l}$ とすると以下のように線形化できる．

$$\begin{gathered}\sum_{p_{i',j'} \in a_{i,j,l}} v_{(i,j),(i',j')} + (1 - e_{i,j,l}) \cdot U \geqq N_{way} \\ \sum_{p_{i',j'} \in a_{i,j,l}} v_{(i,j),(i',j')} - e_{i,j,l} \cdot U < N_{way} \\ e_{i,j,l} \in \{0, 1\}\end{gathered} \tag{25}$$

式 (17) は $writeback(w_{i,j,l})$ の値を保持する変数を $f_{i,j,l}$ と置くと以下のように変換することができる．

$$\begin{gathered}\sum_{a_{i,j,k} \in w_{i,j,l}} e_{i,j,k} - f_{i,j,l} \cdot U \leqq 0 \\ \sum_{a_{i,j,k} \in w_{i,j,l}} e_{i,j,k} + (1 - f_{i,j,l}) \cdot U > 0 \\ f_{i,j,l} \in \{0, 1\}\end{gathered} \tag{26}$$

式 (19)-(26) で新たに定義した変数および式 (8)-(10) および式 (14)，(15) を線形式に変換することができる．目的関数である式 (2) は式 (8)-(10) および式 (14)，(15) を用いて線形化できる．目的関数を線形で表し，式 (19)-(26) で示したように，制約条件を線形で表すことができたため，コード配置問題は整数計画問題として定式化された．

5. 終わりに

本稿では低消費エネルギー化を目的として，cacheable 領域，scratchpad 領域および non-cacheable 領域へのコード配置を行う問題を定義し，その ILP モデルの提案を行った．

今後の課題として，ILP モデルの検証およびプリフェッチ動作を考慮したモデルの考案を行っていく予定である．

謝辞 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．

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] Yuriko Ishitobi, Tohru Ishihara, Hiroto Yasuura, “ Code Placement for Reducing the Energy Consumption of Embedded Processors with Scratchpad snd Cache memories ”, In Proc. of ESTIMedia, pp.13-18, October 2007.