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

2006－HPC－107 (50)
2006／8／2

# Cellプロセッサにおける DMA 転送の融合方式の提案

城田祐介 † 橘内和也 †
松崎秀則 † 前田誠司 †

一般に、DMA 転送では、転送サイズが小さい場合に転送レートが低下し転送時間が大きくなってしまうという課題がある。本稿では、DMA 転送命令を自動挿入する Cell プロセッサ向けコンパイラにおいて、転送サイズを考慮することで転送時間を小さくする DMA 転送の最適化手法を提案する。提案手法では、全体的に分散している実アプリケーションのデータ参照領域のうち、領域間の間隔が小さい領域のみを選択的に融合する転送方式が利用できる。この方式により、必要なデータのみをより大きい転送サイズで転送することで、転送時間を小さくすることができる。その一方で、アプリケーションのデータ参照領域の分布によっては、別の転送方式で転送時間が小さくなる場合も考えられる。提案手法では、複数の転送方式を用意し、それらの転送時間を予測することで転送時間が最小となる方式を自動選択することができる。予備評価を Cell プロセッサ上で行なった結果、提案手法を利用することで性能向上が実現できる可能性を確認した。

# A DMA Transfer Fusion Method for the Cell Processor

Yusuke Shirota Kazuya Kitsunai Hidenori Matsuzaki †,†,† and Seiji Maeda †

DMA transfers suffer from transfer rate deterioration in case where the transfer size is small. In this paper, compiler optimizations, whereby the compiler optimizes the insertion of DMA transfer instructions in consideration of the transfer size are introduced as to reduce DMA transfer time for the Cell processor. We propose a new data transfer method which by fusing DMA transfers of only the small neighboring data in the highly dispersed access regions exhibited by real applications, sends less redundant data and reduces transfer time. Despite its optimality, appropriate data transfer method depends on the dispersities of access region of the application. In order to reduce transfer time on differing access region dispersities of applications, a method which automatically selects a data transfer method that minimizes transfer time, is also introduced. The results of preliminary evaluations on the Cell processor show that the proposed optimizations can be useful.

## 1. はじめに

近年、ディジタル情報機器では、コーデック等のマルチメディア処理が重要になっている。マルチメディア処理では、コンテンツの高解像度化・高画質化に伴い、そのアルゴリズムが複雑化しているため、高い処理性能が要求されている。オンチップマルチコアプロセッサである Cell Broadband Engine™(Cell プロセッサ)[1] は高い性能を備えており、このようなマルチメディア処理をソフトウェアで実現することができる[2]。

マルチメディア処理の特徴として、処理するデータ量が膨大である場合が多いことが挙げられる。そのため、メモリアクセス時間が大きくなり、全体の処理時間に大きく影響する傾向にある。Cell プロセッサの場合、マルチメディア処理は、同処理に特化したコアである SPE(Synergistic Processor Element) で行なう。SPE のメモリアクセスでは、SPE のローカルメモリである LS(Local Storage) とメインメモリ間の DMA 転送に時間を要するため、DMA 転送の最適化が重要となる。

アプリケーションプログラムがメインメモリ上のデータにアクセスするためには、コンパイラがデータアクセスに応じてデータ参照領域を転送する DMA 転送命令を自動挿入する必要がある。本稿では、その際の DMA 転送時間を小さくするコンパイラ最適化手法を提案する。

DMA 転送時間は、初期遅延時間+(転送サイズ/LS とメインメモリ間の転送レート) で表わされる。このため、転送サイズが小さい場合には、DMA 転送時間の多くは初期遅延時間で占められてしまうため、DMA 転送の転送レートは、LS とメインメモリ間の転送レートより大きく低下し、転送時間が大きくなってしまう。しかし、コンパイラが DMA 転送命令を挿入する際に、転送サイズは従来考慮されていなかったため、転送するデータ参照領域が小さい場合に、転送時間が大きくなってしまうという課題があった。

そこで本稿では、近接するデータ参照領域のみを選択的に融合して転送する DMA 転送方式を提案する。転送サイズが大きくなるように融合するだけでは、実アプリケーションで多く見られる分散したデータ参照領域では、領域間の不要なデータも転送してしまうため、かえって転送時間が大きくなってしまう場合がある。提案方式では、近接する領域のみに限定することで、不要なデータの転送を少なくし、転送時間が大きくなることを抑制できる。

一方で、アプリケーションプログラムのデータ参照領域の分布によっては、提案方式とは別の方式を用いた方が転

† (株) 東芝 研究開発センター
Toshiba Corporation, Corporate Research & Development Center

送時間を小さくできる場合もあると考えられる。

そこで本稿では、提案方式とその他に2つの一般的な方式を用意し、転送時間がその中で最小となる方式を自動選択する手法を提案する。これにより、アプリケーションプログラムのデータ参照領域の分布に応じて転送時間を小さくすることができる。

以下、2章でDMA転送命令を自動挿入するCellプロセッサ向けコンパイラについて述べ、3章で一般的なDMA転送方式とその課題について説明する。4章では転送サイズを考慮したDMA転送の最適化手法を提案する。以降、5章で提案手法の特徴である転送時間が大きくなることを抑制するDMA転送方式について、6章で複数のDMA転送方式の中で転送時間が最小となる方式を選択する手法について述べる。7章でマルチメディアアプリケーションをモチーフにした予備評価を行ない、8章で関連研究について述べ、9章でまとめる。

## 2. Cellプロセッサ向けコンパイラ

### 2.1 Cellプロセッサ

Cellプロセッサの構成を図1に示す。Cellプロセッサは1つのPPE(Power Processor Element)と8つのSPEが1チップ上に実装されているヘテロジニアスマルチコアプロセッサである。PPEは、Powerアーキテクチャの64bitコア本体(PPU)と512KBのL2キャッシュを備える。SPEは、マルチメディア処理に特化したコアであり、SIMD命令を中心とした命令セットアーキテクチャと256KBのLSで構成されるコア本体(SPU)、DMAC(DMAコントローラ)等で構成されるMFCを持つ。各コアは、EIB(Element Interconnect Bus)を介して相互接続される。メインメモリは、XDR-DRAMを用いて構成し、XIO経由でCellプロセッサに直接接続する。

構成要素であるSPUは、プログラムとデータをLS上に置いて処理を行なう。そのため、メインメモリ上に置かれているデータを処理する際には、DMACを使ってLSへDMA転送してから処理を行なう。処理後の出力データは、LSからメインメモリへDMA転送する。

また、SPUとDMACは独立に動作可能であるため、DMA転送はSPUの処理とオーバラップすることができる。

### 2.2 コンパイラの構成

SPU上で動作するプログラムでは、DMA転送命令を使ってメインメモリとLS間のDMA転送を行なう。よって、コンパイラは、プログラムのデータアクセスに応じて、そのデータ参照領域を転送するDMA転送命令を自動挿入する必要がある。

本稿で想定するコンパイラの構成を図2に示す。想定するコンパイラの入力は、DMA転送命令のないC言語で記述されたソースプログラムとする。まず、フロントエンドにより、入力ソースプログラムを中間言語に変換する。次に、DMA転送命令挿入ルーチンにより、中間言語に対してDMA転送命令を挿入する。最後に、バックエンドにより、中間言語からDMA転送命令が挿入されたSPU向け実行プログラムを生成し、これを出力する。

DMA転送命令挿入ルーチンでは、DMA転送命令を挿入するために、まず、データ参照領域の解析を行なう。次に、そのデータ参照領域を転送するDMA転送命令の挿入を、後述するDMA転送方式に基づいて行なう。

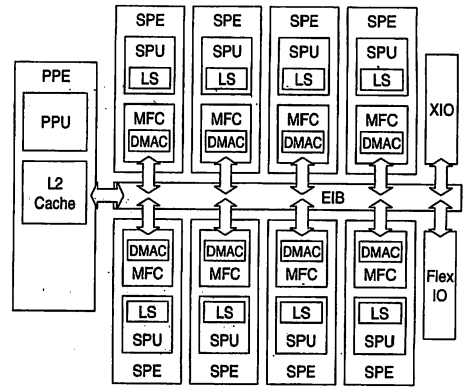

図1 Cellプロセッサの構成

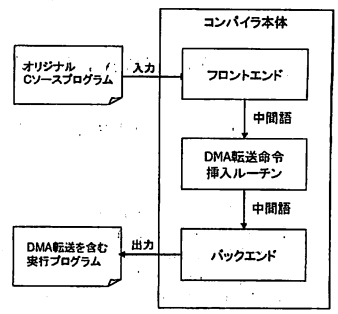

図2 想定コンパイラの構成

## 3. 一般的なDMA転送方式とその課題

本章では、一般的なDMA転送方式とその課題について述べる。

### 3.1 一般的なDMA転送方式

コンパイラによるDMA転送命令の挿入は、一般に、次の2つのDMA転送方式のいずれかに基づいて行なわれると考えられる。

一つは、プログラムのデータ参照領域を個別に転送する方式である。もう一つは、全データ参照領域を融合して転送する方式である。

### 3.2 課　　題

DMA転送では、転送レートが転送サイズによって変化するという性質がある。転送サイズが小さい場合、転送時間中の初期遅延時間の比率が大きくなるため、DMA転送の転送レートは、LSとメインメモリ間の転送レートより大きく低下し、転送時間は大きくなってしまう。

従来、DMA転送命令を挿入する際に、転送サイズは考慮されていなかった。そのため、データ参照領域を個別に

転送するDMA転送方式では、個々のデータ参照領域が小さい場合、転送時間が大きくなってしまう課題があった。

## 4. 転送サイズを考慮したDMA転送の最適化手法の提案

本章では、転送サイズを考慮することで、個々のデータ参照領域が小さい場合に転送時間が大きくなってしまうことを抑制する手法を提案する。

個々のデータ参照領域が小さい場合には、複数の領域を融合してより大きい転送サイズで転送した方が、領域間の不要な部分があっても結果的に転送時間が小さくなる場合がある。全データ参照領域を融合して転送するDMA転送方式はこの効果を期待したものであるが、実アプリケーションのデータ参照領域は分散しているため、効果は限定的である。

そこで本稿では、全データ参照領域を融合して転送する方式のように一括して融合するのではなく、近接するデータ参照領域のみを選択的に融合するDMA転送方式を提案する。

提案方式では、領域間の間隔が小さい部分と大きい部分が混在している実アプリケーションプログラムのデータ参照領域において、不要な部分が少ない領域を融合し、不要な部分が多い領域は融合しないというように、領域の分布に応じて融合処理を切り替えることができるため、転送時間を小さくすることができる。

一方で、提案方式は必ずしも全てのアプリケーションプログラムのデータ参照領域の分布に対して、一般的な2つの方式と比較して転送時間を小さくできるわけではない。例えば、データ参照領域間の間隔が全体的に大きい場合には、個別に転送する方が転送時間を小さくできる場合もある。また、間隔が全体的に小さい場合には、全領域を融合して転送する方が転送時間を小さくできる場合もある。そこで本稿では、提案方式1つのみを利用するのではなく、3つの転送方式の中で最も小さい転送時間で転送できるように、各方式での転送時間を予測し、転送時間が最小となる方式を選択する手法を提案する。

以下に、転送サイズを考慮することを特徴とした、DMA転送の最適化手法の全体の処理フローを示す。本手法は、3ステップで構成される。

最初に、データ参照領域から以下の3つのDMA転送方式による転送内容を解析する。

- データ参照領域を個別に転送する方式
- 近接データ参照領域を融合して転送する方式
- 全データ参照領域を融合して転送する方式

次に、解析した転送内容を基に、各DMA転送方式での転送時間を予測し、転送時間が最小となる方式を選択する。最後に、選択されたDMA転送方式に基づいて、DMA転送命令を挿入する。

以降、5章で選択候補の一つとなる近接データ参照領域を融合するDMA転送方式について説明し、6章で選択手法について述べる。

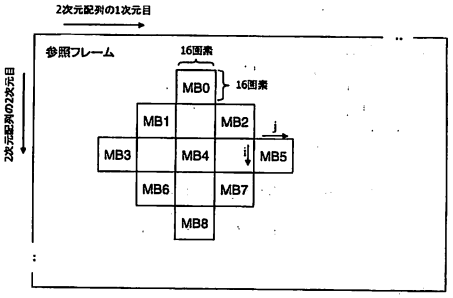


図3 例題プログラムの2次元配列上のデータ参照領域

## 5. 転送時間が大きくなるのを抑制するデータ参照領域の融合方式

本章では、近接データ参照領域を融合するDMA転送方式とその転送内容の解析手法について、マルチメディアアプリケーションをモチーフにした例題プログラムを用いて説明する。

### 5.1 モチーフ

モチーフとして利用するマルチメディアアプリケーションは、ビデオコーデックにおける動き検出処理の検出アルゴリズムの一つであるダイアモンドサーチ[3]とした。ダイアモンドサーチでは、図3に示すような参照フレーム上のダイアモンド形状の検索範囲中のマクロブロック(MB)から、符号化対象マクロブロックとのSAD(Sum of Absolute Difference)が最小となるものを検索する。

ダイアモンドサーチをモチーフにした例題プログラムを図4に示す。

例題プログラムでは、1920×1080画素の参照フレームは2次元配列として確保され、マクロブロックの大きさは16×16画素となっている。

jループではマクロブロックの各行(図3のj方向の16画素分)、iループではマクロブロック1個分のSADを計算する。外側のkループでは、図3のMB番号順に、合計9個分のマクロブロックのSADを計算する。

### 5.2 データ参照領域の解析例

例題プログラムについて、各DMA転送方式での転送内容を解析するために、データ参照領域の解析を行なう。

データ参照領域の解析は、最内側ループから外側に向かって行なう。ここでは、データ参照領域は、[データ参照領域の先頭アドレス:データ参照領域のサイズ]と表記することとする。

ここでは、ポインタ変数ref_pixに注目する。最内側のjループでは、ref_pixは16[B]移動するので、j=0のときのref_pixのアドレスをref_pix_j0とすると、データ参照領域は[ref_pix_j0:16]となる。

一つ外側のiループでは、i=0のときのref_pixのアドレスをref_pix_i0とすると、データ参照領域は、[ref_pix_i0 + i*(1904+16):16](i=0..15)となる。

以上から、16[B]のデータ参照領域が16個存在し、各領域の先頭アドレスの間隔が1920[B]であることが解析でき

```
const int offset[9] =
{
  -1920*16*2,                  /* MB0     */
  -1920*16-16, -1920*16+16,    /* MB1,2   */
  -16*2, 0, 16*2,              /* MB3,4,5 */
  1920*16-16, 1920*16+16,      /* MB6,7   */
  1920*16*2                    /* MB8     */
};

void diamond_search(char* _cur_frame,
                    char* _ref_frame,
                    int _f_i,int _f_j, int *_k)
{

 int k,i,j;
 int sad = 0;

 char *ref_mb,*ref_pix,*cur_mb,*cur_pix;

 cur_mb = _cur_frame + _f_i * 1920  + _f_j;
 ref_mb = _ref_frame + _f_i * 1920  + _f_j;

 for(k=0; k<9; k++)
 {
   cur_pix = cur_mb;
   ref_pix = ref_mb + offset[k];

   for(i=0; i<16; i++)
   {
     for(j=0; j<16; j++)
     {
       sad += abs((int)*(ref_pix++)-(int)*(cur_pix++));
     }
     ref_pix += 1904;
     cur_pix += 1904;
   }
   /* minimum? */
   sad = 0;
 }
}
```

図 4　例題プログラムの概要

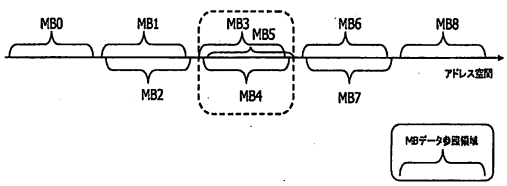


図 5　例題プログラムのデータ参照領域解析結果

る。このマクロブロック 1 個分に対応するデータ参照領域を MB データ参照領域と呼ぶことにする。

さらに外側の k ループでは、データ参照領域は、[ref_mb + offset[k] + i*(1920):16](i=0..15)(k=0..8) となる。

以上から、先頭アドレスが ref_mb + offset[k] となる MB データ参照領域が、9 個存在することが解析できる。図 5 に、9 つの MB データ参照領域のアドレス空間上での配置を示す。

図 6 に、図 5 の点線枠内に相当する MB3、MB4、MB5 の MB データ参照領域が重なっている部分に注目した図を示す。各 MB データ参照領域中の黒塗りの 16[B] の領域が、そのマクロブロックの各行に対応している。3 個の

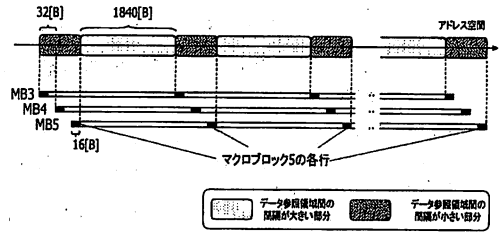


図 6　3 個の重なった MB データ参照領域

MB データ参照領域は、アドレス空間上には 32[B] ずつだけずれた状態で配置されている。このため、MB データ参照領域間の対応する領域が近接しているデータ参照領域間の間隔が小さい部分と、それ以外の間隔が大きい部分とが混在している。このため、近接データ参照領域を融合する DMA 転送方式が効果的であると考えられる。

### 5.3　一般的な DMA 転送方式に基づいた転送

5.2 節で解析されたデータ参照領域の情報を基に、一般的な 2 つの DMA 転送方式での転送内容の解析例を示す。

まず、データ参照領域を個別に転送する DMA 転送方式では、MB データ参照領域中の 16[B] の領域を個別に転送する。この転送を、(MB データ参照領域中の領域 16 個分 x MB データ参照領域 9 個分) 回行なう。以上により、転送内容は、転送サイズが 16[B] の転送が 144 回となる。この場合、個々の転送サイズが小さいため、転送時間が大きくなってしまう可能性がある。

次に、全データ参照領域を融合して転送する DMA 転送方式では、最小の先頭アドレスである MB0 の MB データ参照領域の先頭アドレスである ref_mb + offset[0] から、最大の末尾アドレスである MB8 の MB データ参照領域の末尾アドレスである ref_mb + offset[8] + 28815 までを転送する。以上により、転送内容は、転送サイズが 151696[B] の転送が 1 回となる。この場合、データ参照領域間の不要なデータを多く送ることになるため、転送時間が大きくなってしまう可能性がある。

### 5.4　近接データ参照領域を融合する DMA 転送方式

5.2 節で解析されたデータ参照領域の情報を基に、近接データ参照領域を融合する DMA 転送方式での転送内容の解析例を示す。

近接データ参照領域は、データ参照領域間の間隔が、ある閾値以下の領域と定義する。閾値には、各データ参照領域のサイズに係数を掛けた値を利用する。本稿では、係数を 2 とする。

例えば、データ参照領域のサイズが $l_a$ と $l_b$ のデータ参照領域が $d$ の間隔でアドレスの昇順に並んでいる場合に、$d < l_b * 2$ であれば、2 つのデータ参照領域は近接データ参照領域と判断し融合する。

以下に、例題プログラムで具体例を示す。図 5 において、MB データ参照領域が重なっている各部分について、ひとつずつ説明していく。

図 6 で注目した MB3、MB4、MB5 が重なる部分から説明する。まず、MB3 と MB4 の MB データ参照領域の領域 [ref_mb + offset[3]:16] と [ref_mb + offset[4]:16] に

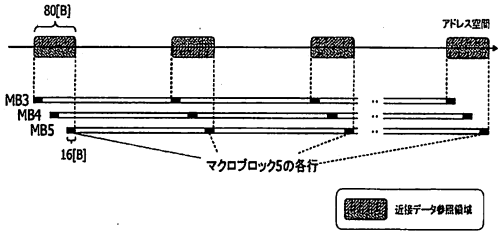


図 7　近接データ参照領域の融合

注目すると、$d = 16$、$l_b * 2 = 32$ となり、近接データ参照領域と判定される。MB4 と MB5 の領域 [ref_mb + offset[4]:16] と [ref_mb + offset[5]:16] ついても、同様に近接データ参照領域と判定される。次に、MB5 と MB3 の領域 [ref_mb + offset[5]:16] と [ref_mb + offset[3] + 1920:16] に注目すると、$d = 1840$、$l_b * 2 = 32$ となり、近接データ参照領域と判定されない。よって、MB3 の領域 [ref_mb + offset[3]:16] から MB5 の領域 [ref_mb + offset[5]:16] までを含む 80[B] の領域が近接データ参照領域となる。以上のように融合すると、図 7 に示すように、3 個の MB データ参照領域間の対応する各領域を融合した近接データ参照領域が 16 個判定される。よって、この部分についての転送内容は、転送サイズが 80[B] となる転送が 16 回となる。

図 5 の MB1 と MB2 が重なる部分では、2 個の MB データ参照領域がアドレス空間上に 32[B] ずつだけずれた状態で配置されている。このため、この部分でも同様に、データ参照領域間の間隔が小さい部分と、間隔が大きい部分とが混在している。まず、MB1 と MB2 の MB データ参照領域の領域 [ref_mb + offset[1]:16] と [ref_mb + offset[2]:16] に注目すると、$d = 16$、$l_b * 2 = 32$ となり、近接データ参照領域と判定される。次に、MB2 と MB1 の領域 [ref_mb + offset[2]:16] と [ref_mb + offset[1] + 1920:16] に注目すると、$d = 1872$、$l_b * 2 = 32$ となり、近接データ参照領域と判定されない。よって、MB1 の領域 [ref_mb + offset[1]:16] から [ref_mb + offset[2]:16] までを含む 48[B] の領域が近接データ参照領域となる。以上のように融合すると、2 個の MB データ参照領域間の対応する各領域を融合した近接データ参照領域が 16 個判定される。よって、転送内容は、転送サイズが 48[B] となる転送が 16 回となる。また、MB6 と MB7 が重なる部分についてもこれと同様である。

図 5 の MB0 の部分では、MB データ参照領域の重なりがない。そのため、各データ参照領域間で $d = 16$、$l_b * 2 = 1904$ となり、近接データ参照領域は存在しない。このため、16[B] の各領域を個別に転送することになる。よって、転送内容は、転送サイズが 16[B] の転送が 16 回となる。また、MB8 についてもこれと同様である。

## 6. 転送時間を小さくする DMA 転送方式の選択手法

5 章までにおいて、3 つの DMA 転送方式での転送内容が解析できた。次は、この中で転送時間が最小になる DMA 転送方式を選択する。

表 1　コンパイラが保持する転送時間の例

| 転送サイズ [B] | 4 | 16 | 64 | 256 | 1024 | 4096 | 16384 |
|---|---|---|---|---|---|---|---|
| 転送時間 [ns] | 50 | 51 | 53 | 62 | 97 | 238 | 804 |

転送時間が最小になる方式を選択するために、まずは各方式を利用した際の転送時間を予測する。次に、その転送時間を比較し、転送時間が最小となる方式を選択する。

転送時間を予測する際には、事前に測定した各転送サイズのデータの転送時間を利用する。しかし、すべての転送サイズでの測定値を保持することは現実的ではない。特定の転送サイズの測定値がない場合には、以下のように線形補間を行ない、その補間値を用いる。

転送サイズ $l$ での転送時間 $t(l)$ が必要な場合に、コンパイラが保持している $l$ より小さい最も近い値を $l_i$、$l$ より大きい最も近い値を $l_{i+1}$ とする。この際には、線形補間を行ない、転送時間 $t(l)$ を $(1 - dl) * t(l_i) + dl * t(l_{i+1})$ (但し $dl = (l - l_i)/(l_{i+1} - l_i)$ ) のように求める。

各 DMA 転送方式での転送時間を予測する際には、5 章で求めた各 DMA 転送方式のデータ転送内容を用いる。転送時間は、データ転送内容の個々の DMA 転送の転送時間の総和として求める。

以下に例題プログラムでの具体例を示す。ここでは、事前に測定した各転送サイズでの転送時間を表 1 のように仮定する。

データ参照領域を個別に転送する方式では、転送時間は、$\sum_{i=1}^{144}(t(16))$ となる。$t(16)$ には表 1 の測定値が使えるので、転送時間は $\sum_{i=1}^{144}(51) = 7.3[us]$ となる。

全データ参照領域を融合して転送する方式では、転送時間は、$t(151696)$ となる。DMA の転送サイズの上限が 16[KB] であるため、実際には、$\sum_{i=1}^{9}(t(16384)) + t(4240)$ となる。$t(4240)$ については、表 1 の $t(4096)$ と $t(16384)$ から線形補間によって、245[ns] と求めることができるので、転送時間は、$\sum_{i=1}^{9}(804) + 245 = 7.5[us]$ となる。

近接データ参照領域を融合して転送する方式では、転送時間は、$\sum_{i=1}^{32}(t(16)) + \sum_{i=1}^{32}(t(48)) + \sum_{i=1}^{16}(t(80))$ となる。$t(48)$、$t(80)$ は、同様に線形補間することで、それぞれ、52[ns]、54[ns] と求めることができるので、転送時間は、$\sum_{i=1}^{32}(51) + \sum_{i=1}^{32}(52) + \sum_{i=1}^{16}(54) = 4.2[us]$ となる。

以上の結果より、3 つの方式による転送時間を比較すると、近接データ参照領域を融合して転送する方式が 3 つの中で最小となるため、同方式を選択する。

## 7. 予備評価

### 7.1 予備評価の目的

転送サイズを考慮した DMA 転送の最適化手法では、DMA 転送方式によってアプリケーションプログラム性能に有意な差が生じることが前提となる。そこで、各 DMA 転送方式に基づき DMA 転送命令を挿入した評価プログラムを作成し、それらの実行時間に有意な差があることを評価し、本手法による最適化の可能性があることを確認する。

### 7.2 評価環境

表 2 の Cell プロセッサを搭載した東芝評価ボード上で

表 2 評価環境

| 項目 | スペック |
|---|---|
| Cell プロセッサ | 2.8GHz |
| PPE 数 | 1 個 |
| SPE 数 | 8 個 |
| XDR メモリ | 512MB |
| Linux | 2.6.12 |

表 3 各方式での実行時間

| DMA 転送方式 | 実行時間 [us] |
|---|---|
| 個別に転送 | 7.75 |
| 近接領域融合 | 4.89 |
| 全領域融合 | 7.93 |

表 4 各方式での転送時間

| DMA 転送方式 | 転送時間 [us] |
|---|---|
| 個別に転送 | 4.32 |
| 近接領域融合 | 2.59 |
| 全領域融合 | 5.65 |

評価を行なった。

評価には、例題プログラムを基に作成した評価プログラムを利用する。評価プログラムの実装は、コンパイラで行なう最適化として、SAD 計算の SIMD 化とダブルバッファの利用を想定して行なった。

### 7.3 各 DMA 転送方式に基づいた評価プログラムの実行時間

各評価プログラムの実行時間を測定した結果を表 3 に示す。

表 3 より、DMA 転送方式のみが異なる各評価プログラムにおいて、その実行時間に性能差が出ていることが分かる。これにより、本手法によって複数の DMA 転送方式から転送時間を最小化する方式を選択することで、最適化を行なえる可能性があることを確認できた。

次に、近接データ参照領域を融合して転送する DMA 転送方式を用いた場合の実行時間が最小になっていることが分かる。この結果から、この方式の効果が確認できた。

### 7.4 各 DMA 転送方式に基づいた転送時間

5 章で得られた転送内容を基に、各転送サイズでの転送時間の測定値を利用して計算した各 DMA 転送方式での転送時間を表 4 に示す。

表 4 より、近接データ参照領域を融合して転送する方式の転送時間が最小になっていることが分かる。これによって転送時間が最小の方式と実行時間が最小の方式が一致したことになる。この結果は提案する選択手法が有用である可能性を示すものと考えられる。

また、全領域を融合する方式では、個別に転送する方式と比較して、転送時間が大きくなってしまっている。これは、不要な部分を多く転送した結果であると考えられる。提案方式では、不要なデータを取り除きつつ融合することで、個別に転送する方式より転送時間が小さくなっていると考えられる。

しかし、提案方式がすべてのアプリケーションで必ずしも転送時間を最小にできるとは限らない。全データ参照領域を融合する方式では、例えば参照フレームの 2 次元配列の横幅がより小さい場合では、不要な部分の転送が少なくなる。このようにデータ参照領域間の間隔が全体的に小さい場合、提案方式との性能差は小さくなると考えられる。

また、例えば 2 次元配列上のマクロブロックの 1 次元方向の間隔がより大きい場合、提案方式で融合できる領域が減少する。このように、データ参照領域が全体的に分散している場合、提案方式との性能差は少なくなると考えられる。

以上により、転送時間が最小になる方式が変化することも考えられるため、3 つの方式から転送方式を選択することが必要と言える。

## 8. 関連研究

ソフトウェア制御オンチップメモリ向け自動最適化コンパイラの研究には、SCIMA[4] や IBM の single source compiler[5] が知られている。

双方のコンパイラにおいて、オンチップメモリ上のデータの再利用に関するデータ転送の最適化を行なっているが、本手法で行なっているような転送サイズを考慮するものではない。

また、ループブロッキング等のメモリアクセスの局所性を高める手法も数多く存在する[6]。これらの手法が処理順序を変更して、キャッシュの効率利用を計るのに対して、近接データ参照領域を融合する DMA 転送方式では、データ転送のサイズを変更することで DMA 転送の効率化を計っている。

## 9. おわりに

本稿では、コンパイラが DMA 転送命令を挿入する際に、転送サイズを考慮する DMA 転送の最適化手法を提案した。

最適化手法では、近接データ参照領域を融合する DMA 転送方式を含めた 3 つの DMA 転送方式を用意し、その転送時間が最小となる方式を選択する。

予備評価を実施した結果、本手法を用いることで、アプリケーションプログラムの性能を向上させることができる可能性があることを確認した。

今後は、本最適化手法の実装を行ない、様々なアプリケーションプログラムで評価することで、有用性を確認していく。

## 参考文献


1) D.Pham et al. "The Design and Implementation of a First Generation CELL Processor," ISSCC, 2005.
2) S.Maeda et al. "A Real-Time Software Platform for the Cell Processor," IEEE Micro, Vol.25, pp.20-29, 2005.
3) S.Zhu et al. "A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation," IEEE Trans. Image Processing, Vol.9, pp.287-290, 2000.
4) 藤田元信, 近藤正章, 中村宏. ソフトウェア制御オンチップメモリ向け自動最適化コンパイラの提案. 情処論 ACS, Vol. 44, No. SIG1(ACS4), pp. 77-87, 2004.
5) Octopiler: http://www.research.ibm.com/cellcompiler/.
6) Intel: IA-32 Intel Architecture Optimization.