LC-004

マトリックス型超並列プロセッサのための処理分割手法の提案

Task Partitioning Method for Massively Parallel Processor Based on Matrix Architecture

中村　仁†　　田中　浩明†　　坂主　圭史†　　今井　正治†　　武内　良典†

1. はじめに

近年，携帯電話やデジタルカメラなどの組み込みシステムにおいて，大量の演算を必要とするマルチメディアアプリケーションの実行が増加している．これに伴い，組み込みシステムに求められる処理能力への要求は日々高まってきている．一方，多様なマルチメディアアプリケーションの登場とともに，さまざまなマルチメディアデータの規格も次々と登場している．したがって，今後の組み込みシステムでは高い処理能力とともにさまざまな規格に対応可能な柔軟性を併せ持つ必要がある．従来はこういったさまざまなマルチメディア処理に対応した専用ハードウェアを組み込みシステムに搭載することによって処理能力を向上させていた．しかしながら，専用ハードウェアを用いる場合，特定の演算に対して高い処理性能を発揮することはできるが，新しい規格などへの柔軟性に欠ける．また，柔軟性の観点から，組み込みシステムに汎用プロセッサを用いてマルチメディアアプリケーションを実行する方法もあるが，これではあまり高い処理性能が期待できない．

そこで，高い処理性能と柔軟性を併せ持つプログラマブルデバイスとして，株式会社ルネサステクノロジ社よりマトリックス型超並列プロセッサ(MTA)が提案されている．MTA では細粒度の 2 入力プロセッシングエレメント(PE)とその左右に配置された 2 つのメモリを 1 ラインとし，命令メモリに格納された命令に従って 1,024 ライン同時実行演算を行うことが可能である[1][3]．MTA は 200MHz 動作時に Intel Pentium 4 3.4GHz 相当以上の処理能力を発揮することが報告されており[2]，同一処理の繰り返しが要求されるマルチメディアアプリケーションに非常に適している．

この MTA の処理能力を有効に使用するためには，並列性の高い繰り返し処理をアプリケーションプログラムから抽出し，適切に MTA で演算する必要がある．

そこで本研究では，アプリケーションプログラムに含まれている繰り返し処理を対象に，実行サイクル数を最小化する CPU/MTA 分割手法について提案する．

本稿の構成は次のとおりである．まず第 2 章で関連研究について述べ，第 3 章で MTA の構成，実行モデルと MTA 利用時の問題点を説明し，次の第 4 章で提案するシステムの概要，MTA を用いることによって削減されるサイクル数の算出手法，MTA で処理を行うループの選択手法について説明し，第 5 章で評価実験，第 6 章でまとめと今後の課題について述べる．

2. 関連研究

本研究と関連する研究として R.Deepa らの手法[7]がある．[7]では，ヘテロジーニアスなマルチプロセッサシステムにおけるタスクスケジューリング手法を提案しており，タスクの依存関係と各タスクの各プロセッサごとの実行コストを元にタスクスケジューリングを行う．

本研究で対象とする，ホスト CPU と MTA の 2 つのプロセッサを含むシステムはヘテロジーニアスなマルチプロセッサシステムであるといえる．しかし，MTA は SIMD 処理のみに特化したプロセッサであり，データメモリ量，命令メモリ量の一般的なプロセッサにはみられない制約を持つため，既存のマルチプロセッサシステムにおけるタスクスケジューリング手法をそのまま適用して処理の分割を行うことはできない．したがって本研究では，MTA の特徴を考慮し，MTA に適した処理分割手法を提案する．

3. MTA

本章ではまず MTA の構成について説明し，次に MTA の実行モデルについて説明する．そして最後に MTA の利用における問題を説明する．

3.1 MTA の構成

MTA は 2 ビットの演算器を 1,000 を超える並列度で同時に動作させることが可能な，非常に細粒度の SIMD プロセッサであり，高い柔軟性と高い処理性能を両立している．本節では，MTA の構成について説明する．

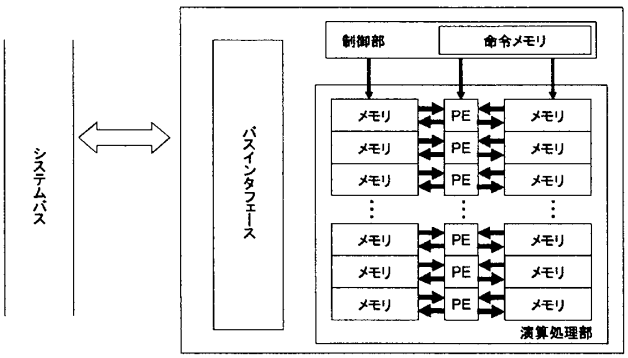

図 1：MTA の構成

MTAは図 1に示すように，演算処理部，制御部，バスインタフェースの部分からなる．

演算処理部はプロセッシングエレメント(PE)と呼ばれる小規模の演算器が縦に並び，その両翼にデータメモリとして用いられるメモリが配置されている．演算処理部では 1,000 を超える並列度で演算が実行される．

制御部は PE の処理内容を決定するための命令メモリを内蔵する．制御部の命令メモリの内容を変更することによって，演算処理部で実行される演算の内容を変化させることができる．

バスインタフェースは外部のバスと通信するためのインタフェースである．MTA はホスト CPU のアクセラレータとしての利用を想定しており，このバスインタフェースを

† 大阪大学大学院情報科学研究科

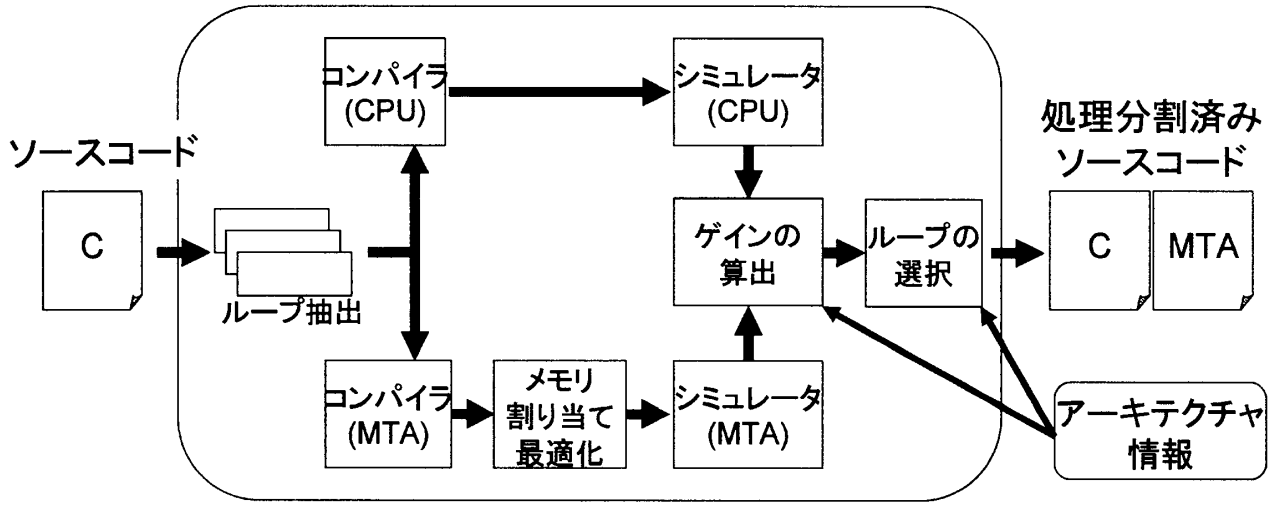

図 2：CPU/MTA 分割手法の概要

通して処理対象データを送受信する．

また，PE 数，データメモリ量，命令メモリ量などが異なるさまざまなバリエーションの MTA が存在する．

3.2 MTA の実行モデル

本節では，MTA の実行モデルについて述べる．

MTA では SIMD 型演算器の特徴を生かして，並列に展開したループを処理する．そして MTA で処理を行う場合，MTA を次の 3 つのフェーズで動作させる．

1. 入力フェーズ：メインメモリから MTA のデータメモリへデータを書き込む
2. 演算フェース：制御部の命令メモリの内容にしたがって処理を実行する
3. 出力フェーズ：MTA のデータメモリからメインメモリへデータを書き出す

例えば，MTA 内の PE 数を 1,024 とし，2,048 回のループを MTA で処理する場合，まずループ 1,024 回分のデータを MTA のデータメモリに入力し，処理を実行し，処理されたデータを出力する．続けて，残りのループ 1,024 回分のデータについても同様に入力，処理，出力を行う．

MTA を用いて処理を高速化させるためには，データの入出力や PE の特性などを考慮して，どのような処理をホスト CPU から MTA へ切り出すかが重要となる．以下に MTA で処理を切り出す場合に考慮しなければならない点について述べる．

- MTA で実行する処理の内容やデータ量によって削減されるサイクル数は大きく異なるため，データの入出力も考慮しながら，サイクル数削減効果の大きい処理をホスト CPU から MTA へ切り出さなければならない．また，MTA で処理を実行している間は命令メモリを書き換えることができない．したがって，MTA 内の命令メモリ容量内でサイクル数削減効果の大きい処理をホスト CPU から MTA へ切り出さなければならない．
- MTA 内の PE は 2 ビットで処理を実行する．したがって，単体の PE はホスト CPU 内の演算器と比較すると性能が劣るため，並列度の高い処理を MTA で実行する必要がある．また，MTA はその構造上，異なる処理を並列に実行できないため，並列度の高い同一の処理をホスト CPU から MTA へ切り出さなければならない．
- MTA の PE 数，データメモリ量，命令メモリ量，そして MTA と外部メモリ間のデータの転送速度は利用する MTA および MTA を含むシステムによって異なる．そのため，利用するシステムによって，サイクル数削減効果や MTA で処理可能な命令数が変化する．したがって，選択した MTA および MTA を含むシステムに応じて，削減されるサイクル数を最大化するように，処理をホスト CPU から MTA へ切り出さなければならない．

以上のことから本稿では，対象アプリケーションを解析し，実行サイクル数を最小化するようなホスト CPU から MTA へ処理を切り出す手法(CPU/MTA 分割手法)を提案する．

4. CPU/MTA 分割手法

図 2 に提案する CPU/MTA 分割手法の概要を示す．提案手法は，MTA を含む組み込みシステム上で実行する対象のアプリケーションの C 言語ソースコードを入力とし，ホスト CPU 実行部分の C 言語プログラムと MTA 実行部分の命令のソースコードを出力する．

4.1. CPU/MTA 分割手法の概要

以下に CPU/MTA 分割手法の処理の概要を示す．

1. 入力された C 言語のソースコードからループを抽出する．
2. 抽出したループをホスト CPU で処理する場合に必要となるサイクル数を見積もる．
   - I. ループをホスト CPU のコンパイラでコンパイルする．
   - II. 回路シミュレータを用いて，ホスト CPU で処理を行う場合に必要となるサイクル数を見積もる．

3. 抽出したループを MTA で処理する場合に必要となるデータ量，命令メモリ量，演算フェーズで必要となるサイクル数を見積もる．
   I. ループを MTA のコンパイラでコンパイルし，初期 MTA コードを生成する．
   II. MTA メモリ最適化アルゴリズム[4]を用いて MTA コードのメモリ最適化を行い，最適化された MTA コードを得る．
   III. 最適化された MTA コードを MTA のシミュレータを用いて， MTA で処理を行う場合に必要となるデータ量，命令メモリ量，演算フェーズで必要となるサイクル数を見積もる．
4. 2, 3 で得た情報とアーキテクチャ情報により，各ループを MTA で実行する際に削減されるサイクル数(削減量)をそれぞれ計算する．
5. 命令メモリの範囲内で削減量を最大化するように MTA で処理するループを選択する．
6. MTA で処理するループ部分を書き換えたホスト CPU の C 言語のソースコードと MTA に切り出されたループ部分の MTA コードを出力する．

提案手法中の，ホスト CPU のコンパイラと回路シミュレータ，MTA のコンパイラと MTA のシミュレータは既存のものを使用する．

MTA はその演算モデルの特徴により，演算に使用する変数が，PE を挟んで両翼に配置されているメモリのどちらに配置されているによって，演算にかかる実行サイクル数が異なってくる[4]．提案手法では，我々が別途開発した，メモリ割り当て最適化手法[4]を用いて，MTA コンパイラが主力したコードを最適化して，MTA での実行サイクル数を算出している．

次に，提案手法中の削減量の算出手法とループの選択手法を説明する．

4.2 削減量の計算

提案手法では以下の情報を用いて，ループをMTAで実行した場合の実効サイクル数の削減量を求める．ここで，入力されたソースコードに含まれるループの集合を$L=\{l_i|i=1,...,n\}$とする．

- MTA に固有のパラメタ
  - M : MTA の命令メモリ量 (byte)
- MTA を含むシステムに固有のパラメタ
  - R : メインメモリ，MTA 内のデータメモリ間のデータの転送速度(byte/cycle)
- ループごとのパラメタ
  - $C_{mta,i}$: ループl_i をMTAで処理するために必要なサイクル数 (cycle)
  - $C_{cpu,i}$: ループl_i をホストCPUで処理するために必要なサイクル数 (cycle)
  - E_i :ループl_iの演算フェーズに必要なサイクル数 (cycle)
  - $D_{in,i}$: ループl_i で処理されるためにホストCPUからMTAへ転送されるデータ量 (byte)
  - $D_{out,i}$: ループl_i で処理し， MTAからホストCPUへ転送されるデータ量 (byte)
  - I_i : ループl_i をMTAで処理するために必要な命令メモリ量 (byte)

MTA の命令メモリ量 M は使用する MTA が決定されることで決まる．また，メインメモリと MTA 間のデータ転送速度も，MTA を使用するシステムのシステムバスの性能によって決定される．

ループl_i をホストCPUで処理した場合に必要なサイクル数$C_{cpu,i}$ は，ループl_iをホストCPUのコンパイラでコンパイルし，回路シミュレータを用いることで求まる．

ループl_i で処理されるためにホストCPUからMTAへ転送されるデータ量 $D_{in,i}$と，ループl_i で処理してMTAからホストCPUへ転送されるデータ量$D_{out,i}$ は， ループl_iをMTAコンパイラでコンパイルすることで求められる．

また，ループl_i をMTAで処理するために必要な命令メモリ量I_i もMTAコンパイラでコンパイルすることで求められる．

3.2 節で述べたように，ループl_i をMTAで処理するために必要なサイクル数 $C_{mta,i}$ は，入力フェーズ，演算フェーズ，出力フェーズに必要なサイクル数の和となる．演算フェーズに必要なサイクル数E_iはMTAのシミュレータを用いることで求められる．入力フェーズ，出力フェーズに必要なサイクル数はメインメモリ，MTAのデータメモリ間のデータの入出力量をそれぞれ転送速度で割ることによって求められる．したがってループl_iをMTAで処理する場合に必要な総サイクル数 $C_{mta,i}$ は以下のように計算される．

$$C_{mta,i}=\frac{D_{in,i}}{R}+E_i+\frac{D_{out,i}}{R}$$

4.3 ループの選択

4.2 節で求めた各ループの$C_{cpu,i}$ と $C_{mta,i}$を用いて，どのループをMTAで処理するかを決定する．特にループをMTAで処理した場合に必要となる命令メモリ量によっては，全てのループをMTAで処理出来ないことがある．そこで本節では，命令メモリを制約とし，実行サイクル数を最小にするようなループのMTAへの切り出し問題(CPU/MTA分割問題)を定式化する．

ループをホストCPUで処理するかMTAで処理するかを表す変数x_iを定義する．ループl_iをホストCPUで処理する場合$x_i=0$とし， l_iをMTAで処理する場合$x_i=1$とする．

前述したように，MTA の命令メモリ量以上のループは処理出来ないため，全てのループを MTA で処理できないこともある．したがって，MTA で処理するループの命令コード量の総和が命令メモリ量を超えないことが制約条件となり，以下の式となる．

制約： $$\sum_{i=1}^{n} I_i \cdot x_i \le M$$

アプリケーションの実行サイクル数を最小にするように，上記制約を満たす範囲で，どのループをMTAで処理するかを決定する必要がある．つまり，上記制約を満たす範囲で，MTAで処理することで削減されるサイクル数を最大化する必要がある．ここで，ループをMTAで処理することで削減されるサイクル数は， l_i をホストCPUで処理した場合に必要なサイクル数$C_{cpu,i}$と l_i をMTAで処理した場合に必要なサ

イクル数$C_{mta,i}$との差である．以上より，目的関数は次の式となる．

目的関数：最大化：$\sum_{i=1}^{n}(C_{cpu,i}-C_{mta,i})\cdot x_i$

以上の議論をまとめ，本研究であつかう問題をCPU/MTA 分割問題として以下の様に定式化する．

CPU/MTA 分割問題

- 入力
  - ループ集合 $L=\{l_i: i=1,\ldots,n\}$
  - ループl_i をMTAで処理するために必要なサイクル数$C_{mta,i}$
  - ループl_i をホストCPUで処理するために必要なサイクル数$C_{cpu,i}$
  - MTA の命令メモリ量 M
- 出力
  - x_i
- 制約
  - $\sum_{i=1}^{n} I_i \cdot x_i \le M$
- 目的
  - 最大化：$\sum_{i=1}^{n}(C_{cpu,i}-C_{mta,i})\cdot x_i$

ここで，CPU/MTA 分割問題の，命令メモリ量 M をナップザックの大きさ，各ループの命令メモリ量を大きさ，各ループの実行サイクルの削減量を価値とみなすと，CPU/MTA 分割問題はナップザック問題として扱うことができる．

ナップザック問題は NP-完全であることが知られており[5]，色々な解法が提案されているが，本稿では，分枝限定法を用いてこの問題を解く．

分枝限定法は全探索の 1 種で，これ以上の探索が不要であると判断された時点で枝刈りを行う手法である．

5. 評価実験

提案手法の有効性を確認するために評価実験を行った．実験には，ホスト CPU としてエイシップ・ソリューションズ社の Brownie_std32 プロセッサを採用したシステムを想定し，ホスト CPU，MTA，バスの動作周波数はともに同じ動作周波数とした．また，対象アプリケーションとして OpenJPEG library[6]の 5-3 ウェーブレット変換のソースコードから，48 種のループを含むソースコードを用い，解の導出には，分枝限定法により CPU/MTA 分割問題を解くプログラムを作成して用いた．

ループ部分に関してサイクル数を比較した結果を表 1 に示す．

	サイクル数	削減率
CPU のみ使用	184,728	-
提案手法	61,484	67%

表 1：サイクル数の比較

提案手法を用いて MTA を利用すると， MTA を用いない場合と比較してサイクル数が約 67%削減され，この例題の場合，約 3 倍高速に実行可能であることが示された．また解は 48 ループ中 18 ループを MTA に割り当てるものとなり，探索時間は Pentium4 3.0GHz, 1GB RAM の環境で 1 秒未満であった．

以上より，提案手法の有効性を確認した．

6. まとめと今後の課題

本稿では，CPU/MTA 分割手法を提案し，その有効性を示した．

今後の課題としては，データの依存関係を考慮することにより，MTA 内のメモリにデータを入出力するサイクル数を削減することが挙げられる．これは入力されたソースコードにおいて，データの依存関係を考慮して処理の順序を適切に入れ替えることによって実現できると考えられる．しかしながら，処理の連続する異なるループ間で，同じ変数を利用している場合においても，ただデータの入出力処理を削除するのではなく，データメモリの配置を考慮して，配置が異なる場合にはデータメモリを再配置するといった処理が必要となる．

文　　献

[1] M. Nakajima et.al., ``A 40GOPS 250mW Massively Parallel Processor Based on Matrix Architecture," Proceedings of ISSCC, pp. 410-411, 2006.

[2] 大野隆行, 山崎博之, 飯田全広, 久我守弘, 末吉敏則, ``Matrix Processing Engine のメディア処理アプリケーションによる性能評価," 信学技報, Vol. 106, No. 49, RECONF2006-6, pp. 31-36, 2006.

[3] 中田 清, 中島雅美, 野田英行, 谷崎哲志, 行天隆幸, ``40GOPS 250mW マトリックス型超並列プロセッサの開発 モバイル向け SoC にも組み込み可能な超高速プロセッサ," 信学技報, vol. 106, no. 71, ICD2006-25, pp. 19-23, 2006.

[4] 小橋 晶, 谷口 一徹, 坂主 圭史, 武内 良典, 今井 正治, 中田 清, ``マトリックス型超並列プロセッサにおける変数のメモリ割り当て最適化手法," 信学技報, 2007.

[5] Michael R. Garey, David S. Johndon, ``Computers and intractability. A guide to the theory of NP-completness", San Francisco, W. H. Freeman, 1979

[6] OpenJPEG library (www.openjpeg.org)

[7] R.Deepa, T.Srinivasan, D.Doreen, H.Miriam, ``An Efficient Task Scheduling Technique in Heterogeneous Systems Using Self-Adaptive Selection-Based Genetic Algorithm", PARELEC, p343-348, 2006.