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

2007－SLDM－130（1）
2007／5／10

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

小橋 晶† 谷口 一徹† 坂主 圭史† 武内 良典† 今井 正治†
中田 清††

† 大阪大学 大学院情報科学研究科 〒565-0871 大阪府吹田市山田丘 1-5
†† 株式会社 ルネサステクノロジ システムソリューション統括本部 〒664-0005 兵庫県伊丹市瑞原
E-mail: †{a-kobasi,i-tanigu,sakanusi,takeuchi,imai}@ist.osaka-u.ac.jp, ††nakata.kiyoshi@renesas.com

あらまし 本研究では, マトリックス型超並列プロセッサ MTA (MaTrix processing Array) の変数のメモリへの割り当て最適化手法について提案する. MTA は, 株式会社ルネサステクノロジにより開発された信号処理用途向け並列プロセッサであり, MTA 内部に複数存在するメモリへの変数の割り当てによって, 演算に必要な実行サイクル数が大きく変化する. そこで本研究では, 変数の割り当てによって発生するオーバーヘッドを最小にする, 変数のメモリへの割り当て最適化手法を提案する. 提案手法により, DCT 演算における変数の割り当てを考慮しない場合におけるオーバーヘッドを約 96%削減できることを確認した.
キーワード MTA, 組合わせ最適化, メモリ割り当て, 最大カット問題

Memory Assignment Method for Matrix Processing Array

Akira KOBASHI†, Ittetsu TANIGUCHI†, Keishi SAKANUSHI†, Yoshinori TAKEUCHI†,
Masaharu IMAI†, and Kiyoshi NAKATA††

† Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka Suita, Osaka 565-0871 Japan
†† RENESAS Technology Corporation
Mizuhara Itami, Hyogo 664-0005 Japan
E-mail: †{a-kobasi,i-tanigu,sakanusi,takeuchi,imai}@ist.osaka-u.ac.jp, ††nakata.kiyoshi@renesas.com

Abstract MTA (MaTrix processing Array), which is developed by Renesas Technology Corp., can achieve high performance for digital signal processing using its high parallelism. However, the execution cycles of MTA greatly change by the data assignment for the internal memories. In this paper, we propose a data assignment optimization method for MTA. Experimental results show that proposed method can reduce overhead about 96% in the case of DCT.
Key words MTA, Combinatorial optimization, Memory assignment, Maximum cut problem

1. はじめに

近年, デジタルカメラや携帯音楽プレーヤなどの組み込みシステムにおいて, 大量の演算を行うマルチメディアアプリケーションが実行される機会が多くなってきている. それに伴い, 組み込みシステムにおいて要求される処理能力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, 新しいマルチメディアデータの規格などが次々と提案されており, 今後の組み込みシステムには, 大量の演算を実行できる高い処理能力と様々な規格へ容易に対応する柔軟性が求められる.

従来は, ディジタル信号処理に向いた専用演算回路を組み込みシステムに搭載することでマルチメディアアプリケーションを実現していた. しかしながら, 専用回路を用いる場合, 特定のアプリケーションに対しては非常に高い処理能力を発揮できるが, 新しい規格などへの対応という点では柔軟性に欠ける. 一方で, 柔軟性の観点から, 組み込みシステムでマルチメディアアプリケーションを実現するために, 汎用プロセッサが用いられ

ることがあるが, あまり高い処理性能は期待できない.

そこで, 高性能, 柔軟性の特徴を持ったプログラマブルデバイスとして, マトリックス型超並列プロセッサ MTA(Matrix Processing Array) が株式会社ルネサステクノロジ社より提案されている. MTA では, 細粒度な 2 入力の演算器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 1024 ライン同時に演算を行う [2] [4]. MTA は 200MHz 動作時において, 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 [3], 今後の展開が大いに期待される. しかしながら, MTA での演算は, 左右に配置されたメモリに格納されている変数を同時に読み出して演算することを前提としているため, 演算に必要な変数が同じ側のメモリに格納されている場合, メモリから 2 つの変数を同時に取り出すことができないため, 演算にオーバーヘッドが生じてしまう. したがって, 演算に用いる変数をどちらのメモリに格納するかは, MTA でのアプリケーションの実行サイクル数に大きく影響する.

そこで本研究では, MTA での処理の高速化を目指し, MTA で実行する演算について, 変数の左右のメモリへの割り当てを最適化する手法を提案する. 変数のメモリへの割り当てを考えるために, 本稿では, MTA での処理において, どの 2 変数を用いて演算が行われるかを表現する Data Relational Graph(DRG) を導入し, DRG 上で変数の左右のメモリへの割り当てを最適化する.

本稿の構成を以下に示す. 第 2 章で MTA の構成について述べ, 第 3 章で MTA での演算のモデル化について説明する. そして, 第 4 章で, 提案する変数の割当て最適化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それぞれ述べる.

2. Matrix Processing Array

本研究で対象とする MTA(MaTrix processing Array) について説明する.

2.1 MTA の全体構成

図 1 に MTA の構成を示す. MTA は演算部, メモリ部, 制御部からなる. 演算部とメモリ部は, PE (Processing Element) と PE の左右に配置された 2 つの 512bit メモリを 1 ラインとし, それが縦に 1024 ライン並んだ構成となっている. 制御部は命令メモリとコントローラからなり, 命令メモリに格納された MTA 命令列にしたがって PE とメモリを制御する. MTA は基本的に, 各 PE は各々のラインの両側に配置されたメモリに格納されている変数に対して演算を行うため, 1 命令で最大 1024 個の演算が可能な SIMD (Single Instruction Multiple Data) 型演算器である.

2.2 変数の配置による実行サイクル数の変化

MTA では, PE と左右のメモリの接続関係より, 演算に必要な変数が異なるメモリから読み込まれることを前提にしている. そのため MTA では, 演算命令の 2 つの変数が同じメモリから読み込まれる場合は, 演算の実行サイクルにオーバーヘッドが発生する.

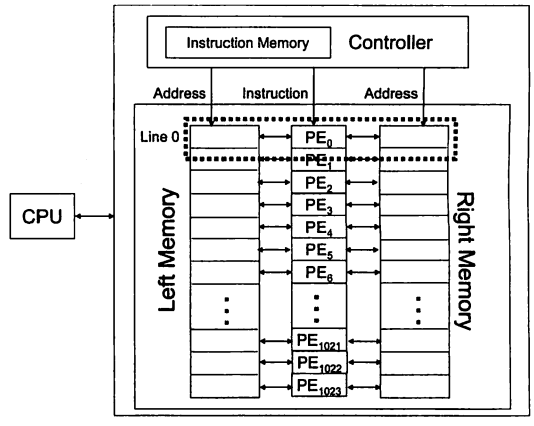

図 1 MTA(MaTrix processing Array) の全体構成

図 2 に MTA で想定している標準的な演算モデルを示す. 図 2 では左右のメモリから変数 x と変数 y を同時に読み込み, PE で演算し結果を変数 x に出力する. これを各ラインに格納されている変数 $x[0]$ から $x[1023]$, 変数 $y[0]$ から $y[1023]$ 対して同時に行う. このように MTA では, 演算に必要な変数が左右の異なるメモリに保存されていることを前提としている.

一方, 図 3 に, 演算に必要な変数が同じ側のメモリに格納されている場合を示す. この場合, 左メモリから変数 x と変数 y を順に読み込み, PE で演算し結果を変数 x に出力する. したがって, 演算に必要な変数が同じメモリに格納されている場合は, 2 つの変数を同時に読み込むことができず, 順に読み込まなければならないため, 変数が異なるメモリに格納されている場合に比べ, 実行サイクルにオーバーヘッドが生じてしまう.

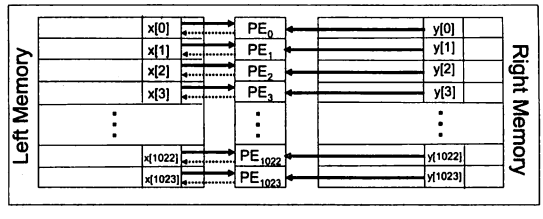

図 2 標準的な演算モデル (2 変数が異なるメモリに格納)

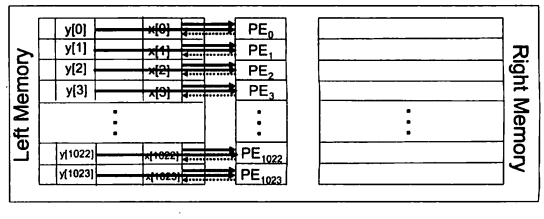

図 3 標準的でない演算モデル (2 変数が同じメモリに格納)

以上のように MTA では, 演算に使用される変数が左右のどちらのメモリに格納されているかが, 実行サイクル数に大きな影響を及ぼす. そこで本研究では, 与えられた MTA の命令列

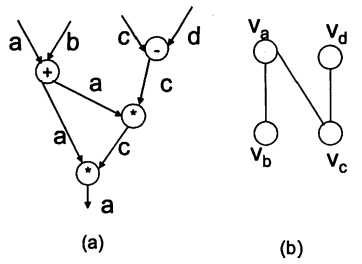

図 4 DFG and DRG

に対して, 使用する変数の左右のメモリへの割り当てを最適化するアルゴリズムを提案する.

3. MTA での演算のモデル化

本研究では MTA における演算を Data Relational Graph (DRG) とよばれるグラフを用いてモデル化し, DRG 上で変数のメモリ割り当てを最適化する. 本章では DRG について説明する.

3.1 Data Relational Graph (DRG)

処理の依存関係を表現する方法として, 演算を点で, 演算に用いる変数を辺で表す Data Flow Graph (DFG) がしばしば用いられる. DFG は演算間の順序や依存関係の把握には適しているが, 同じ変数を表す辺が複数存在するため, 本研究で扱う変数のメモリへの割り当てを DFG 上で扱うことは難しい. そこで本研究では, 変数間の演算による関係を表現するために, 変数を点とし, 変数間の演算による関係を辺で表す Data Relational Graph (DRG) を提案する.

DRG $G = (V, E)$ は, MTA の命令列における変数と演算の関係を, 変数に対応する点 v_i の集合 V と変数間の演算による関係に対応する辺 e_k の集合 E で表現する. DRG 中の辺 $e_k = (v_i, v_j)$ は, 辺 e_k に対応する演算で使用する変数のうち, 点 v_i と点 v_j に対応する変数のメモリ割り当てが演算のオーバーヘッドに影響することを表している. これを演算による関係という. また, 辺 e_k は, 両端の点 v_i, 点 v_j が同じ側のメモリに割り当てられた場合に発生する演算のオーバーヘッドを重み w_k として持つ.

MTA での演算 $a = a + b$; $c = c - d$; $c = a * c$; $a = a * c$ に対する DFG 表現を図 4(a) に示し, DRG 表現を図 4(b) に示す. 図 4(a) に示すように, MTA での処理を DFG で表現すると, 同一の変数に対応する辺が複数存在してしまうが, 図 4(b) の DRG では, 変数に対応する点は 1 つのみとなっている.

ここで, 変数 a と変数 c の間には, 2 つの乗算 ($c = a * c$, $a = a * c$) による関係が存在するが, DRG グラフ上では, 点 a と点 c の間に 2 本の辺を張るのではなく, 1 本の辺に対して 2 つの演算のオーバーヘッドの和を重みとする.

3.2 DRG 上でのメモリ割り当ての表現

提案手法では, MTA の命令列から変数と演算による関係を DRG で表現し, DRG 上で演算のオーバーヘッドを評価しながら, 変数のメモリへの割り当てを最適化する. 本研究では, 変数の左右のメモリへの割り当て ASN を, 左メモリ集合 MEM_L と右メモリ集合 MEM_R の組み合わせ $ASN = (MEM_L, MEM_R)$ で表現する. 左メモリ集合 MEM_L は, 左側のメモリに割り当てられた変数の集合を表し, 右メモリ集合 MEM_R は, 右側のメモリに割り当てられた変数の集合を表す. すべての変数は必ず MEM_L または MEM_R のどちらか一方のみに含まれる.

$$MEM_R, MEM_L = \{v \in V\}$$

$$MEM_R \cup MEM_L = V$$

$$MEM_R \cap MEM_L = \phi$$

ここで, 辺 $e_k = (v_i, v_j)$ が点 v_i と点 v_j を接続しているとき, 点 v_j を点 v_i の隣接点とよぶ. 点 v_i に対し, 点 v_i と同じメモリに含まれている隣接点を同メモリ隣接点とよび, そうでない点を異メモリ隣接点とよぶ. また, 点 v_i と同メモリ隣接点を接続する辺を同メモリ間辺とよび, そうでない辺を異メモリ間辺とよぶ. とくに, 左メモリにある点 v_i と同メモリ隣接点 v_j を接続する辺を左同メモリ間辺とよび, 右メモリにある点 v_i と同メモリ隣接点 v_j を接続する辺を右同メモリ間辺とよぶ.

ここで, DRG $G = (V, E)$ とメモリ割り当て $ASN = (MEM_L, MEM_R)$ に対して異メモリ間辺集合 ED, 同メモリ間辺集合 ES, および左同メモリ間辺集合 ES_L, 右同メモリ間辺集合 ES_R を以下のように定義する.

$$ED = \{e_k = (v_i, v_j) | v_i \in MEM_L, v_j \in MEM_R\}$$

$$ES_L = \{e_k = (v_i, v_j) | v_i, v_j \in MEM_L\}$$

$$ES_R = \{e_k = (v_i, v_j) | v_i, v_j \in MEM_R\}$$

$$ES = ES_L \cup ES_R$$

3.3 DRG 上での演算オーバーヘッド

演算のオーバーヘッドに関係する変数が異なるメモリに格納されている演算は, DRG 上では両端点が異なるメモリ集合に属している辺に対応する. 演算のオーバーヘッドに関係する変数が同じメモリに格納されている演算は, DRG 上では両端点が同じメモリ集合に属している辺に対応する. したがって, MTA での演算サイクルのオーバーヘッドの総量は, 同メモリ間辺集合 ES に含まれる辺の重みの総和となる. したがって, 演算のオーバーヘッドの総量 OH は以下の式で求めることができる.

$$OH = \sum_{e_k \in ES} w_k$$

4. メモリ割り当て最適化手法

4.1 メモリ割り当て問題

本研究で扱う, 与えられた MTA 命令列に対してオーバーヘッド総量 OH を最小にするような変数のメモリ割り当てを求める問題は, 与えられた DRG に対して, 同メモリ間辺集合の辺の重み総和を最小にするような点 $v_i \in V$ のメモリ集合への割り当て $ASM = (MEM_R, MEM_L)$ を求める問題となる.

ここで, 点 $v_i \in V$ のメモリ集合 MEM_R, MEM_L への割り当てによらず各辺の重みは一定であるので, 与えられた DRG

上で同メモリ間辺集合の辺の重み総和の最小化は, 異メモリ間辺集合の辺の重み総和の最大化と同値である. ここで, DRG のメモリへ割り当てをグラフの分割問題ととらえると, 異メモリ間辺集合はカット辺集合となる. したがって, 与えられた MTA 命令列に対して実行サイクル数を最小にするような変数の割り当てを求める問題は, 与えられた DRG に対して, カット辺重み総和を最大にするような DRG のメモリ集合 MEM_R,MEM_L への分割問題とも言うことができる. 一般にグラフの最大カット問題 (Maximum cut, MAX-CUT) は, NP-完全であることが知られている [1]. そこで本研究では, 与えられた DRG $G=(V,E)$ を左右のメモリへ割り当て問題を解く発見的手法を提案する.

本研究では, 変数のメモリへの初期割り当てに対して, 変数を割り当てるメモリを逐次的に変更することで, 同メモリ間辺による実行サイクル数のオーバーヘッドを最小化する手法を提案する. まず, アルゴリズムの詳細を説明する前に, DRG 上での逐次改善における性質, ならびに最適解が持つ性質について述べる.

4.2 DRG の逐次改善と最適解に関する性質

DRG におけるオーバーヘッド総量 OH の逐次改善に関する性質と変数のメモリ割り当ての最適解がもつ性質について示す前に, 用語を定義する. ここで点 v_i の同メモリ間辺の重み総和を同メモリ間辺重みとよび, ws_i と記す. 同様に v_i の異メモリ間辺の重み総和を異メモリ間辺重みとよび, wd_i と記す. MEM_L (MEM_R) に割り当てられている点 v_i を $MEM_R(MEM_L)$ へ移動した場合に変化するオーバーヘッドの総量をゲイン g_i と記す. 点 v_i を移動した場合のゲイン g_i は以下に示す式で計算できる.

$$g_i = ws_i - wd_i$$

点 v_i を MEM_L から MEM_R に移動した場合, 移動前に同メモリ間辺であった演算は移動後に異メモリ間辺となり演算のオーバーヘッドは解消されるが, 移動前に異メモリ間辺であった演算は点 v_i の移動により同メモリ間辺となり演算のオーバーヘッドが発生する.

[補題 1] ゲインが g_i であるような点 v_i を MEM_L (MEM_R) から MEM_R (MEM_L) に移動すると移動後の点 v_i のゲイン g_i' は $-g_i$ となる.

証明: 点 v_i を別のメモリへ移動した場合, 移動前に同メモリ間辺であった辺枝は異メモリ間辺となり, 移動前に異メモリ間辺であった辺は同メモリ間辺となるので, 移動前と移動後の同メモリ間辺重みと異メモリ間辺重みをそれぞれ ws_i, wd_i, ws_i', wd_i' と書くと g_i' は以下のようになる.

$$g_i' = ws_i' - wd_i' = -(ws_i - wd_i) = -g_i$$

□

MEM_L に割り当てられている点 v_i を MEM_R へ移動すると, MEM_L にある左同メモリ間辺が異メモリ間辺となるので, MEM_L に割り当てられている点 v_i 以外の点のゲインは点 v_i の移動前と比べて必ず減少するかまたは同じである.

[補題 2] 点 $v_i, v_i \in MEM_L$ を MEM_L から MEM_R へ移動する場合, 移動後の, MEM_L に割り当てられている点 v_i でない点 v_j のゲインを g_j' と書くとき, 点 v_j の移動後のゲイン g_j' は必ず下記の不等式をみたす.

$$g_j' \leqq g_j$$

証明: 点 v_j が点 v_i に隣接する場合と隣接しない場合に分けて証明する.

まず, 点 v_j が点 v_i に隣接していない場合について証明する. 点 v_i が MEM_R へ移動することで同メモリ間辺から異メモリ間辺となる辺または, 異メモリ間辺から同メモリ間辺となる辺は, 仮定より点 v_j と接続していないので, 点 v_j に接続する同メモリ間辺重み総和 ws_j と異メモリ間辺重み総和 wd_j は変化しない. したがって, $g_j' = g_j$ が成り立つ.

次に, 点 v_j が点 v_i に隣接している場合について証明する. 点 $v_i \in MEM_L$ と点 $v_j \in MEM_L$ を接続する辺を e_k とする. このとき, 点 v_i が MEM_R へ移動することで辺 e_k は同メモリ間辺から異メモリ間辺となる. したがって, 点 v_i が移動することで, 辺 e_k は点 v_j の同メモリ間辺から異メモリ間辺となるため, 点 v_j の同メモリ間辺重み総和 ws_j' は辺 e_k の重み w_k だけ減少する. また, 点 v_i が MEM_R に移動すると, 辺 e_k は異メモリ間辺となるので, 点 v_j の異メモリ間辺総和 wd_j' は w_k だけ増加する. よって, 点 v_i が移動した後の点 v_j のゲイン g_j' は以下の不等式を満たす.

$$\begin{aligned} g_j' &= ws_j' - wd_j' \\ &= (ws_j - w_k) - (wd_j + w_k) \\ &= g_j - 2w_k < g_j \end{aligned}$$

以上より, 補題が成り立つ. □

MEM_L に割り当ている点 v_i を MEM_R へ移動すると, 異メモリ間辺が右同メモリ間辺となるので, MEM_R に割り当てられている点 v_j のゲイン g_j は, 点 v_i の移動前と比べて必ず増加するかまたは同じである.

[補題 3] 点 $v_i \in MEM_L$ を MEM_R へ移動するとき, MEM_R に割り当てられている点 v_i 以外の点 v_j の移動前のゲインを g_j と書き, 移動後のゲインを g_j' と記すとすると, 点 v_j の移動後のゲイン g_j' は必ず下記の不等式を満たす.

$$g_j' \geqq g_j$$

証明:

点 v_j が点 v_i に隣接する場合と隣接しない場合に分けて証明する.

まず, 点 v_j が点 v_i に隣接していない場合について証明する. 点 v_i が MEM_R へ移動することで同メモリ間辺から異メモリ間辺となる辺または, 異メモリ間辺から同メモリ間辺となる辺は, 仮定より点 v_j と接続していないので, 点 v_j に接続する同メモリ間辺重み総和 ws_j と異メモリ間辺重み総和 wd_j は変化しない. したがって, $g_j' = g_j$ が成り立つ.

次に, 点 v_j が点 v_i に隣接している場合について証明する.

点 v_i と点 v_j を接続する辺を e_k とする．このとき，点 v_i が MEM_R へ移動することで辺 e_k は異メモリ間辺から同メモリ間辺となる辺とする．したがって，点 v_i が MEM_R に移動することで，辺 e_k は点 v_j の異メモリ間辺から同メモリ間辺となる．よって，点 v_j の同メモリ間辺重み総和 ws'_j は辺 e_k の重み w_k だけ増加する．

$$ws'_j = ws_j + w_k$$

また，点 v_i が MEM_R へ移動すると点 v_j に接続する重み w_k の辺 (v_i, v_j) は異メモリ間辺ではなくなるので，移動後の点 v_j の異メモリ間辺重み総和 wd'_j は移動前の異メモリ間辺重み総和 wd_j から w_k だけ減少する．

$$wd'_j = wd_j - w_k$$

よって，点 v_i が移動した後の点 v_j のゲインは以下の様になり，必ず減少する．

$$\begin{aligned} g'_j &= ws'_j - wd'_j \\ &= (ws_j + w_k) - (wd_j - w_k) \\ &= g_j + 2w_k > g_j \end{aligned}$$

以上より，補題が成り立つ． □

補題 2 より，一般性を失わず，点 $v_i \in MEM_R$ を MEM_L へ移動するときにも，下記の補題が成り立つ．

[補題 4] 点 $v_i, v_i \in MEM_R$ を MEM_R から MEM_L へ移動する場合，移動後の，MEM_R に割り当てられている点 v_i でない点 v_j のゲインを g'_j と書くとき，点 v_j の移動後のゲイン g'_j は必ず下記の不等式をみたす．

$$g'_j \leqq g_j$$

[補題 5] 点 $v_i \in MEM_R$ を MEM_L へ移動するとき，MEM_L に割り当てられている点 v_i 以外の点 v_j の移動前のゲインを g_j と書き，移動後のゲインを g'_j と記すとすると，点 v_j の移動後のゲイン g'_j は必ず下記の不等式を満たす．

$$g'_j \geqq g_j$$

以上までで，あるメモリ割り当て $ASN = (MEM_L, MEM_R)$ から一つの点を移動した場合のゲインの変化についていくつかの補題を証明した．次に本研究で求めようとする最適な変数のメモリ割り当てが持つ DRG 上での性質について説明する．

[補題 6] オーバーヘッドが最小のデータのメモリへの割り当て ASN^{opt} が与えられたとき，対応する DRG グラフ G_{opt} 中にはゲインが正の点は存在しない．

証明: 背理法で証明する．一般性を失わず，ゲインが正の点 v_i が MEM_L にある場合について証明する．DRG グラフ G_{opt} 中にゲインが正の点 v_i が MEM_L に割り当てられていたとする．このとき，ゲインの定義より，点 v_i を MEM_R に移動した場合，オーバーヘッドが減少し，G_{opt} が最適であることに矛盾する．また点 v_i が MEM_R に割り当てられている場合についても同様である． □

[定理 1] オーバーヘッドが最小のデータのメモリへの割り当て ASN^{opt} が与えられたとする．このとき，MEM_R の点のゲインを正にすることなく，かつオーバーヘッド総和 OH を単調増加させながら，MEM_R の点を順に取り出し，MEM_L へ全て移動することができる．

証明: 帰納法を用いて証明する．補題 6 より，与えられた最適なメモリ割り当て ASN^{opt} においては，ゲインが正の点は存在しない．

与えられた最適なメモリ割り当て ASN^{opt} から，k 個の点を MEM_R から MEM_L へ移動したメモリ割り当てを $ASN^k = (MEM_L^k, MEM_R^k)$ とする．今，メモリ割り当て $ASN^k = (MEM_L^k, MEM_R^k)$ において，MEM_R^k 内にゲインが正の点がないと仮定する．ここで，$k+1$ 個目の点 v_i を MEM_R^k から MEM_L^k に移動した $ASN^{k+1} = (MEM_L^{k+1}, MEM_R^{k+1})$，においても，命題を満たすことを示す．仮定より，点 v_i のゲインは負であるので，ゲインの定義より点 v_i を MEM_L^k へ移動するとオーバーヘッド総量 OH は増加する．また，補題 2 より，点 v_i が MEM_R^k から MEM_L^k へ移動すると，MEM_R^k にある点 v_i 以外の点のゲインは減少する．よって，MEM_R^k 内の点のゲインが負で，かつオーバーヘッド総和 OH が増加したメモリ割り当て ASN^{k+1} を得ることができる．

以上より，定理は成り立つ．

□

以上までで，逐次的な解の改善によるゲインの変化，および，最適解が持つ性質と，最適解から点の逐次操作で到達可能なメモリ割り当てについて示した．本研究ではこれらの性質を基に，以下のアルゴリズムを提案する．

4.3 メモリ割り当て最適化手法

点の移動操作は可逆であるので，定理 1 より，全ての点が MEM_L に割り当てられているメモリ割り当てから，点の逐次的な移動のみで最適解へ到達可能であることがわかるため，全ての点が MEM_L へ割り当てられているメモリ割り当てを初期解とする．

また定理 1 は，オーバーヘッド総量 OH を増加させることなく，初期解から最適化へ点を逐次的に移動できることを示しているので，本研究では，ゲインが正の点を MEM_L から MEM_R へ移動することのみを考える．

ここで，どのような順番で移動させるかが重要であるが，第 4.1 節で述べたように，最適なメモリ割り当て問題は NP 完全である．ここで本研究では，補題 2 の性質に着目する．点が移動する度に，MEM_L に残された点のゲインは単調に減少していくので，本研究では，MEM_L の中でゲインが最大の点から順に MEM_R に移動させる．

以上をまとめ，与えられた DRG $G = (V, E)$ に対して，OH を最小にするような点集合 V のメモリへの割り当て (MEM_R, MEM_L) を求めるアルゴリズムを以下に示す．

（1） Step 1 $MEM_L = V, MEM_R = \phi$ を初期解とする．

（2） Step 2 全ての点のゲインを計算する．

（3） Step 3 MEM_L にゲインが正の点があれば，ゲインが最大の点 $v_i \in MEM_L$ を MEM_L から取り出し MEM_R へ移

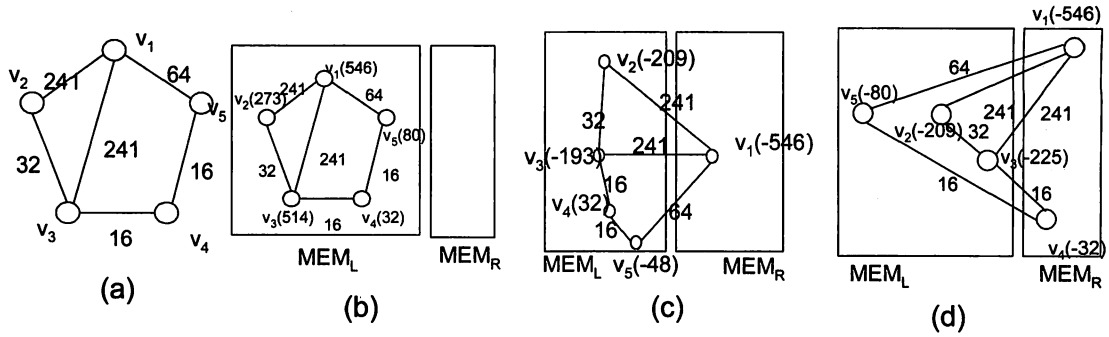

図 5 提案手法によるメモリ割り当て

動し, Step 2 へ. MEM_L にゲインが正の点がなければアルゴリズムを終了する.

初期状態として全ての点を MEM_L に割り当てる. 点 v_i に関して g_i を算出し, g_i が最大となる点を MEM_R に移動する. g_i が最大の点を MEM_R に移動するため, 補題 1 より, MEM_R に移動した点のゲインは負となる. 移動するにつれ, 補題 2 より, MEM_L に格納されている点のゲイン g は単調減少し, 補題 3 より, MEM_R に格納されている点のゲインは単調増加する. MEM_R にゲインが正の点が出現したら, 移動を停止する.

図 5(a) に DRG グラフの例を示す. 図 5(b) に全ての点を MEM_L に割り当てた様子と各点のゲインを示す. MEM_L の中で点 v_1 のゲイン 546 が最大であるので, 点 v_1 を MEM_R へ移動する. 移動した点 v_1 のゲインは, -546 となる (補題 1). 点 v_1 が移動することで, MEM_L の他の点のゲインは減少するかもしくは変化しない (補題 2). 移動後のゲインを計算し直すと, 点 v_4 のゲイン 32 が最大となる (図 5(c)) ので, 点 v_4 を MEM_R へ移動する. 移動後の MEM_R の点 v_1 のゲインは増加するかもしくは変化しない (補題 3). 図 5(d) よりゲインが正の点が存在しなくなったので, 現在の割り当て (MEM_L, MEM_R) を出力する (補題 6).

5. 評価実験

本節では, 提案手法の有効性を示すために, Poly, DCT, subDCT の 3 種類の DRG に対し提案手法を適用し, 演算のオーバーヘッド総和 OH の削減率と CPU 時間で評価した. Poly は多項式演算, DCT は離散コサイン変換 (Discrete Cosine Transformation), subDCT は部分 DCT の DRG をそれぞれ表している. 実験環境は Pentium4 3GHz, 1GB メモリである.

表 1 に実験結果を示す. OH_{init} は提案手法を適応する前のオーバーヘッド総量で, 変数の割当てを考慮せずにコンパイルして得られた解のオーバーヘッド総量である. OH_{prop} は提案手法により得られた解のオーバーヘッド総量, OH_{opt} は 4.1 節で定式化したメモリ割当て問題の最適解のオーバーヘッド総量である. RR_{prop} と RR_{opt} はそれぞれ提案手法と最適解のオーバーヘッド総量の削減率を表す.

表 1 より, subDCT に関して, 提案手法を用いることで, 全探索により求められた最適解のオーバーヘッドの削減率と同じ削減率を得ることができた. 一方, subDCT の最適解を得るために約 1100 秒を要したが, 提案手法では 1 秒未満で解を得ることができた. これより, 提案手法を用いる事で, 非常に良い解を高速に探索可能であることが分かる. また, DCT に関しては, 実用的な時間で最適解を得ることはできなかったが, 提案手法により, 初期解のオーバーヘッドを約 96%削減することができた.

表 1 実験結果

	Ratio of OH Reduction				
	OH_{init} [cycle]	OH_{prop} [cycles]	RR_{prop}	OH_{opt} [cycle]	RR_{opt}
Poly	0	0	0.0%	0	0.0%
DCT	196	8	95.9%	NA	-
subDCT	104	8	92.3%	8	92.3%

6. まとめと今後の課題

本稿では, マトリックス型超並列プロセッサ MTA のための変数の割振り最適化手法を提案した. 提案手法により, MTA の左右のメモリへの変数の割振りを非常に高速に求めることができ, DCT 演算のオーバーヘッドを約 96%削減できた. 今後の課題は, 3 個以上のオペランドを用いた演算への対応である.

文 献

[1] Michael R. Garey, David S. Johnson, "Computers and Intractability," pp. 210, W .H. Freeman and Company, 2002.

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

[3] 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.

[4] 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.