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

2007－ＳＬＤＭ－132 (35)
2007／11／22

超並列細粒度SIMDプロセッサにおける オペランド順を考慮した変数のメモリ割り当て最適化手法

小橋 晶† 谷口 一徹† 田中 浩明† 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,h-tanaka,sakanusi,takeuchi,imai}@ist.osaka-u.ac.jp,
††nakata.kiyoshi@renesas.com

あらまし 近年, 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普及し, 組み込みシステムに要求される処理能力は日々増加している. これらの要求に応えるため, 株式会社ルネサステクノロジにより, 超並列細粒度 SIMD プロセッサ MX(Massively Parallel Fine-grained SIMD Processor MX) が提案されている. MX のプロセッシングアレイ部分である MTA では, 2 個のメモリバンクへの変数の割り当てによって, 演算の実行サイクル数が大きく変化する. また MTA では変数の左右メモリバンクへの割り当てだけでなく, オペランド順によっても実行サイクル数は変化する. そこで本研究では, 演算とデータの関係を表現する Data Relational Hypergraph(DRH) を提案し, MTA の変数の左右メモリバンクへの割り当てを最適化する手法を提案する.
キーワード MX, MTA, 組合わせ最適化, メモリ割り当て, ハイパーグラフ

Memory Assignment Method Considering Orders of Operands for Massively Parallel Fine-grained SIMD Processor

Akira KOBASHI†, Ittetsu TANIGUCHI†, Hiroaki TANAKA†, 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,h-tanaka,sakanusi,takeuchi,imai}@ist.osaka-u.ac.jp,
††nakata.kiyoshi@renesas.com

Abstract In recent years, spread of data intensive multimedia applications requires high-performance in embedded systems. Massively Parallel Fine-grained SIMD Processor (MX), developed by Renesas Technology Corp., achieves high performance for digital signal processing using its high parallelism. The execution cycles of MX, however, is greatly influenced by the way of the data assigned to the internal memory banks called MTA, and the orders of operands in the code of MTA. In this paper, we propose Data Relational Hypergraph (DRH) and an optimization method for the memory assignment in MTA.
Key words MX, MTA, Combinatorial optimization, Memory assignment, Hypergraph

1. はじめに

近年, デジタルカメラや携帯音楽プレーヤ等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搭載され, それに伴い, 組み込みシステムには高い処理能力が求められている. また, 新しいマルチメディア・アプリ

ケーションが次々と登場しており，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ている．

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，新しい規格へは流用できないことが多く，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．

高性能で，かつ高い柔軟性を持ったプログラマブルデバイスとして，超並列細粒度 SIMD プロセッサ MX(Massively Parallel Fine-grained SIMD Processor MX) が，株式会社ルネサステクノロジより提案されている [1] [2]．MX のプロセッシングアレイ部分 MTA では，2 入力 1 出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．

MTA で JPEG2000 エンコーダと MP3 デコーダの一部を 200MHz で実行すると，Intel Pentium4 3.4GHz と同等以上の性能が得られる [3]．また，Ant Colony Optimization による Traveling Salesman Problem の探索を実行すると，200MHz 動作時において，Intel Pentium M 1.1GHz と同等の性能が得られる [4]．

MTA での演算は，左右に配置されたメモリバンクに格納されている変数を読み出して演算することを前提としており，このとき変数を同時に読み出すことができる．しかしながら，演算に必要な変数が同じ側のメモリバンクに格納されている場合，2 つの変数を同時に読み出すことができず，演算にオーバーヘッドが生じる．したがって，演算に用いる変数を左右のどちらのメモリバンクに格納するかによって，MTA でのアプリケーションの実行サイクル数は大きく変化する．

著者らは，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バンクへの割り当てを最適化する手法を提案している [5]．文献 [5] では変数のメモリバンクへの割り当てを最適化するために，MTA での処理において，どの変数を用いて演算が行われるかを表現する Data Relational Graph(DRG) を提案し，DRG によって命令列のモデル化を行い，変数のメモリ割り当てを最適化する手法を提案している．また，Jeonghun Cho らによって，データのメモリバンクへの割り当て問題について，最大全域木 (Maximum spanning tree) を応用した最大カット問題の解法が提案されている [6] [7]．

しかし，文献 [5] [6] [7] の方法では，MTA のプログラム内でのコピー命令，加算命令，減算命令，すなわちオペランド数が 2 以下の演算しか扱うことができない．また，オペランドの入れ替えによる実行サイクル数の変化は考慮されていない．そこで本論文では DRG をハイパーグラフに拡張した DRH(Data Relational Hypergraph) を用いることで，オペランド数が 3 の演算に対しても，変数のメモリバンクへの割り当てとオペランド順を最適化する手法を提案する．

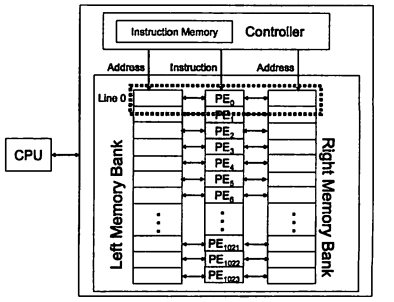

図 1 MTA(Massively Parallel Fine-grained SIMD Processor) の全体構成

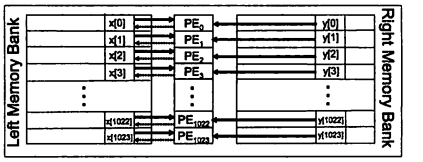

図 2 標準的な演算モデル (2 変数が異なるメモリバンクに格納)

本稿の構成を以下に示す．第 2 章で MTA の構成について概説し，第 3 章で先行研究である DRG と，本研究での提案手法であるハイパーグラフを用いた DRH について述べる．そして，第 4 章で，変数の割り当て最適化手法を示し，第 5 章で評価実験，第 6 章でまとめを述べる．

2. Massively Parallel Fine-grained SIMD Processor

本研究で対象とする，超並列細粒度 SIMD プロセッサ MX(Massively Parallel Fine-grained SIMD Processor) [1] [2] のプロセッシングアレイ部分である MTA について紹介する．

2.1 MTA の全体構成

図 1 に MTA の構成を示す．MTA は演算部とメモリ部から成り，制御部に直結されている．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，MTA の PE とメモリバンクを制御する．MTA では，各 PE は各々のラインの両側に配置されたメモリバンクに格納されている変数に対して同一の演算を同時に行うため，制御部と合わせると，1 命令で最大 1024 個の演算が可能な SIMD (Single Instruction Multiple Data) 型プロセッサとみなせる．

2.2 変数の配置による実行サイクル数の変化

MTA では，PE と左右のメモリバンクの接続関係より，演算に必要な変数が異なるメモリバンクから読み込まれる場合は，データを同時に読み出すことができるため，効率的に演算できる．しかしながら，演算に必要な 2 つの変数が同じメモリバンクから読み込まれる場合は，データを逐次的に読み出すために，演算の実行サイクルにオーバーヘッドが発生する．

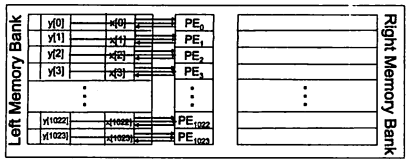

図 3 オーバーヘッドが発生する演算モデル (2 変数が同じメモリバンクに格納)

図 2 に MTA で想定している標準的な演算モデルを示す. 左右のメモリバンクから変数 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] に対して同時に行う.

図 3 は, 演算に必要な変数が同じ側のメモリバンクに格納されている場合を示している. 図 3 では, 左メモリバンクから変数 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バンク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, 逐次的に読み込まなければならないため, 変数が異なるメモリバンク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る.

以上のように MTA では演算に使用される変数が, どちらのメモリに格納されているかによって, 実行サイクル数は大きく変化する.

3. DRG のハイパーグラフ化

MTA にはオペランド数が 2 である加減算以外にも, オペランド数が 3 である乗算命令と MAC 命令 (積和命令) および MAS 命令 (積差命令) が存在する. ここで本研究ではオペランド数が 2 の演算を 2 項演算とよび, オペランド数が 3 の演算を 3 項演算とよぶことにする. 文献 [5] の手法では, 2 項演算のみをメモリ割り当て最適化の対象としていたが, 本論文では, 演算のオペランド数が 3 の場合に対しても最適化できるように, 文献 [5] の DRG をハイパーグラフに拡張した DRH (Data Relational Hypergraph) を提案する.

3.1 Data Relational Graph

著者らは MTA における 2 項演算を 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方法を提案している [5]. DRG は, 変数を点とし, 変数間の演算による関係を辺で表したグラフである. 文献 [5] の中では, 3 項演算は, 第 3 オペランドを無視して, 2 項演算とみなして DRG で表現している. 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 の集合 V と変数間の演算による関係に対応する辺 e の集合 E で表現する. DRG 中の辺 $e_{ij} = (v_i, v_j)$ は, 点 v_i, v_j に対応する変数を使用する演算を示している.

また, 辺 e_{ij} は, 両端の点 v_i, 点 v_j が同じ側のメモリバンク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 $w(e_{ij})$ として持つ.

左メモリバンク集合 $L \subseteqq V$ は, 左側のメモリバンクに割り当てられた変数の集合を表し, 右メモリバンク集合 $R \subseteqq V$ は, 右側のメモリバンク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 L または R のどちらか一方のみに含まれる.

3.2 3 項演算の性質

3 項演算は 2 項演算と同様に, 第 1 オペランドと第 2 オペランドが格納されているメモリバンクが同じ場合に, 実行サイクル数が増加するが, 第 3 オペランドがどちらのメモリバンクに格納されているかは, 実行サイクル数に影響しない. 一方, 第 2 オペランドと第 3 オペランドを交換しても意味が変わらない演算も存在する. 例えば, $a = b * c$ という乗算命令は, $a = c * b$ と記述しても同じ意味の演算である.

したがって, 第 1 オペランドと第 2 オペランドが同じメモリバンクに格納され, オーバーヘッドが発生する場合でも, 第 2 オペランドと第 3 オペランドが異なるメモリバンクに格納されていれば, それらを交換することで, 第 1 オペランドと第 2 オペランドが異なるメモリバンクに格納されていることになり, オーバーヘッドの発生を防ぐことができる. 例えば, $a+ = b * c$ を表す命令において, a と b が左メモリバンクに, c が右メモリバンクに格納されている場合, オーバーヘッドが発生するが, 第 2 オペランドである b と第 3 オペランドである c を入れ替え, $a+ = c * b$ と書き換えることで, 第 1 オペランドと第 2 オペランドを格納するメモリバンクが異なるようになり, 演算サイクルにオーバーヘッドが発生しなくなる. したがって, 3 項演算は全てのオペランドが同じメモリバンクに格納されている場合にのみ, 実行サイクルにオーバーヘッドが発生し, 3 つのオペランドのうち, 1 つでも異なるメモリバンクに格納されている変数があれば, オペランド順の変更のみでオーバーヘッドを削減できる.

3.3 Data Relational Hypergraph

3 項演算において, 実行サイクルにオーバーヘッドが発生する変数のメモリへの割り当てを正確に評価することは, 文献 [5] で提案されている, 2 点間を結ぶ辺によって構成された DRG では困難である. そこで本研究では, DRG の辺をハイパー辺とすることで, DRG をハイパーグラフ化した DRH を提案する.

DRH $H = (V, E)$ 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 の集合 V と, 演算に対応する辺 e の集合 E で表現する. 辺 e は, 対応する演算が使用するオペランドに対応する点の集合である. また辺 $e = \{v_i | v_i \in V\}$ は, 全ての点が同じメモリバンクに割り当てられた場合に発生する実行サイクルのオーバーヘッドを重み $w(e)$ として持つ.

MTA の演算では, 変数が同じメモリバンクに格納されている場合に, 実行サイクルにオーバーヘッドが発生し, 異なるメモリバンクに格納されている場合には, 実行サイクルにオーバーヘッドが発生しない. そこで, 辺 $e \in E$ が, L, R の両側のメモリにまたがっているか否かによって, 辺 e を区別する. 要素である点の集合が, 全て同じメモリバンクに格納されている辺 $e \in \{e | e \subseteqq L \vee e \subseteqq R\}$ を, 同メモリバンク間辺と定義する. 一方で, 要素である点の集合の全てが, 同じメモリバンクに格納されていない辺 $e \in \{e | e \not\subset L \wedge e \not\subset R\}$ を異メモリバンク間辺と定

義する．これらの集合として，同メモリバンク間辺の集合である同メモリバンク間辺集合 E_S と，異メモリバンク間辺の集合である異メモリバンク間辺集合 E_D を，以下のように定義する．

$$E_S = \{e | e \subseteqq L \vee e \subseteqq R\}$$

$$E_D = \{e | e \not\subset L \wedge e \not\subset R\}$$

点 v_i, v_j を含む辺 e が存在するとき，点 v_j を点 v_i の隣接点という．点 v_i に接続する辺の中で，全ての点が v_i と同じメモリバンクに含まれているような辺集合を，点 v_i の同メモリバンク隣接辺集合 $E_S(v_i)$ とする．点 v_i に接続する辺の中で，いずれかの点が v_i と同じメモリバンクに含まれていないような辺集合を，異メモリバンク隣接辺集合 $E_D(v_i)$ とする．

3.4 DRH 上での演算オーバーヘッド

オーバーヘッドが発生する演算は，DRH 上では全ての点が同じメモリバンク集合に属している辺 $e \in E_S$ に対応するため，MTA での同メモリバンク間演算による実行サイクル数の増分は，同メモリバンク間辺集合 E_S に含まれる辺 w_e の重みの総和である．したがって，同メモリバンク演算のオーバーヘッドの総量 W_S は下式で求めることができる．

$$W_S = \sum_{e \in E_S} w(e)$$

4. メモリ割り当て最適化手法

本章では，MTA での演算を表現した DRH 上で，変数のメモリ割り当て最適化手法を提案する．

まず，メモリ割り当て問題を定義し，3 項演算を表現する DRH が持つ性質について示し，文献 [5] で提案した最適化手法を用いて，メモリ割り当てを最適化する手法を提案する．

4.1 メモリ割り当て問題

本研究で扱う，与えられた MTA 命令列に対して実行サイクルのオーバーヘッド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 DRH に対して，同メモリバンク間辺集合 E_S のオーバーヘッド総量 W_S を最小にするような，点 $v_i \in V$ のメモリバンク集合への割り当てを求める問題となる．

本研究では，変数のメモリバンクへの初期割り当てに対して，変数を格納しているメモリバンクを逐次的に変更することで，同メモリバンク間辺集合 E_S 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，DRH 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 DRH の逐次改善と最適解に関する性質

文献 [5] は，オペランド数が 2 の演算を対象とした DRG を提案し，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，いくつかの補題と定理を証明している．本論文で提案している DRH においても，文献 [5] で示した補題と定理が成り立つことを以下に示す．

MTA のプログラムでは，変数を格納しているメモリバンクを変更すると，オーバーヘッドが解消される演算や，オーバーヘッドが新たに発生する演算が存在する．そこで，点 v_i の属するメモリバンクを変更した場合に，解消される実行サイクルのオーバーヘッド s_i と，新たに発生する実行サイクルのオーバーヘッド d_i を定義する．

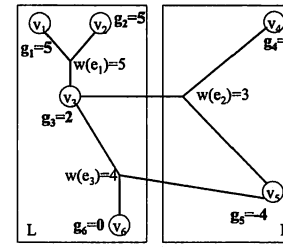

図 4 ゲインの算出例

点 v_i の同メモリバンク隣接辺集合 $E_S(v_i)$ は，点 v_i が移動することで，必ず異メモリバンク隣接辺集合 $E_S(v_i)$ となり，実行サイクル数のオーバーヘッドは解消される．したがって，点 v_i の同メモリバンク隣接辺集合 $E_S(v_i)$ の重み総和が s_i となり，下式となる．

$$s_i = \sum_{e \in E_S(v_i)} w(e)$$

点 v_i の異メモリバンク隣接辺集合 $E_D(v_i)$ の中で，点 v_i 以外の全ての点が異なるメモリバンクに属しているような辺は，点 v_i が移動すると同メモリバンク隣接辺集合 $E_S(v_i)$ となるため，新たに実行サイクル数のオーバーヘッドが発生する．ここで，一般性を失わず点 v_i が左メモリバンクに属していると仮定すると，d_i は下式となる．

$$d_i = \sum_{v_i \in L, e - \{v_i\} \subseteqq R, v_i \in e} w(e)$$

点 v_i が右メモリバンクに属しているときも同様に定義される．

$L(R)$ に割り当てられている点 v_i を $R(L)$ へ移動した場合に変化するオーバーヘッド総量 W_S の差分をゲイン g_i とする．点 v_i を移動した場合のゲイン g_i は次式で表せる．

$$g_i = s_i - d_i$$

例えば，図 4 に示す DRH 中の点 v_3 のゲイン g_3 は，以下のようにして求められる．点 v_3 を R に移動する場合，点 v_3 の同メモリバンク隣接辺集合に含まれる辺 $e_1 \in E_S(v_3)$ は，点 v_3 の異メモリバンク隣接辺集合 $E_D(v_3)$ に含まれるため，オーバーヘッドは解消される．したがって，$s_3 = w(e_1) = 5$ となる．しかし点 v_3 を移動する場合，点 v_3 に接続する異メモリバンク隣接辺集合に含まれる辺 $e_2 \in E_D(v_3)$ は，点 v_3 の同メモリバンク隣接辺集合に含まれる辺 $e_2 \in E_S(v_3)$ となるため，$d_3 = w(e_2) = 3$ となる．辺 e_3 に関しては，点 v_3 を移動させても，点 v_5 が右メモリバンクに格納されているため，ゲインには影響を与えない．よって点 v_3 のゲインは $g_3 = s_3 - d_3 = 5 - 3 = 2$ となる．

点を移動した場合のゲインの変化について，次の補題が成り立つ．

[補題 1] ゲインが g_i であるような点 v_i を L (R) から R (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 の移動後のオーバーヘッド総量を W'_S とする. 点 v_i をさらに R から L に移動した後のオーバーヘッド総量を W''_S とする. このとき点 v_i を L から R に移動し, R から L に移動した場合, メモリ割り当ては元に戻り, オーバーヘッド総量は同じになるため, $W_S = W''_S$ となる. g_i は点 v_i を L から R に移動する場合のオーバーヘッド総量 W_S の減少量を表し, g'_i は点 v_i を R から L に移動した場合に減少するオーバーヘッド総量 W'_S の減少量を表しているため, 下の 2 式が成り立つ.

$$W'_S = W_S - g_i, W''_S = W'_S - g'_i$$

したがって,

$$W_S = W''_S = W'_S - g'_i = W_S - g_i - g'_i$$

$$g'_i = -g_i$$

□

また, L に割り当てられている点 v_i を R へ移動すると, L にある点 v_i の同メモリバンク隣接辺集合に含まれる辺 $e \in E_S(v_i)$ が, 異メモリバンク隣接辺集合に含まれる辺 $e \in E_D(v_i)$ となるので, 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になる性質がある.

[補題 2] 点 $v_i \in L$ を L から R へ移動する場合, 移動後の, L に割り当てられている点 $v_j (v_i \neq v_j)$ のゲインを g'_j と書くとき, 点 v_i の移動後の点 v_j 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 R へ移動することで点 v_i の同メモリバンク隣接辺集合 $E_S(v_i)$ から, 点 v_i の異メモリバンク隣接辺集合 $E_D(v_i)$ に移動する辺, または, 異メモリバンク隣接辺集合 $E_D(v_i)$ から同メモリバンク隣接辺集合 $E_S(v_i)$ に含まれる辺は, 仮定より存在しないため, 点 v_j の同メモリバンク隣接辺集合 $E_S(v_j)$ と, 点 v_j 異メモリバンク隣接辺集合 $E_D(v_j)$ 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 $v_i \in L$ と点 $v_j \in L$ を含む辺を e とする. 点 v_i, v_j 以外の e に含まれる全ての点が L に属している場合, 点 v_i が L から R に移動することによって, 辺 e は点 v_j の異メモリバンク隣接辺集合 $E_D(v_j)$ に含まれるため, オーバーヘッドが解消される. 点 v_i, v_j 以外の e に含まれるいずれかの点が L に属していない場合, 辺 e は点 v_j の異メモリバンク隣接辺集合 $E_D(v_j)$ に含まれており, また, 点 v_j が L に含まれているため, 点 v_i が L から R に移動しても, 辺 e は点 v_j の同メモリバンク隣接辺集合 $E_S(v_j)$ には含まれないので, 新たにオーバーヘッドが発生することはない.

以上より, 点 v_i が R に移動した場合, 移動後の s'_j は s_j と同じか, または少なくなる. 移動後の d'_j は d_j と同じか, または増加する. よって下式が成り立つ.

$$g'_j = s'_j - d'_j \leqq s_j - d_j = g_j$$

□

点 $v_i \in L$ を R へ移動すると, 点 v_i の異メモリバンク隣接辺集合 $E_D(v_i)$ が同メモリバンク隣接辺集合 $E_S(v_i)$ となるので, 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性質をもつ.

[補題 3] 点 $v_i \in L$ を R へ移動するとき, R に割り当てられている点 $v_j (v_i \neq v_j)$ の移動前のゲインを g_j とし, 移動後のゲインを g'_j 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: L から R に移動した点 v_i を再び L に戻した場合の点 v_j のゲインを, g''_j とする. この場合, 元のメモリ割り当てとなるため, $g_j = g''_j$ となる. 補題 1 より,v_i が R から L に移動する場合, R にある点 v_j のゲイン g'_j は下式となる.

$$g'_j \geqq g''_j = g_j$$

□

以上までで, あるメモリ割り当てから一つの点を移動した場合のゲインの変化について, 3 つの補題を証明した. 次に最適な変数のメモリ割り当てが持つ DRH 上での性質について明らかにする.

[補題 4] オーバーヘッド総量 W_S が最小のデータのメモリ割り当てが与えられたとき, DRH グラフ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 L にある場合について証明する. ゲインの定義より, $g_i > 0$ である点 v_i を R に移動した場合, オーバーヘッド総量 W_S が減少し, そのメモリ割り当てのオーバーヘッド総量 W_S が最小であることに矛盾する. また点 v_i が 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バンクへの割り当てが与えられたとする. このとき, R の点のゲインを正にすることなく, オーバーヘッド総量 W_S を単調増加させながら, R の点を順に取り出し, L へ全て移動することができる.

補題 1,2,3,4 より, DRH においても文献 [5] で示した補題が成り立つ. よって, 定理 1 は文献 [5] と同様に証明される.

以上までの議論で, 本研究で提案する DRH と文献 [5] で提案されている DRG が, 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, 同じ性質を持つことを示した. 本研究では文献 [5] で提案している手法を用いて, DRH を最適化する. そこで文献 [5] で示されている最適化アルゴリズムを紹介する.

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, 全ての点が 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解へ到

表 1 実験結果

		オーバーヘッド総量 (cycle)			実行サイクル数 (cycle)			計算時間	
Benchmark	変数	登場順	DRG	DRH	登場順	DRG	DRH	全探索	提案手法
Sample1	3 変数	0	16	0	1763	1778	1763	0.031sec	0.015sec
Sample2	5 変数	0	0	0	844	844	844	0.046sec	0.031sec
Mandel	5 変数	739	16	16	2389	1908	1908	0.046sec	0.031sec
DCT	24 変数	500	0	0	9299	7264	7052	10872.593sec	0.078sec
Sample3	36 変数	500	144	112	3217	2836	2804	NA	0.140sec

達可能であるため, 全ての点が L に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 W_S を増加させることなく, 初期解から最適解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 L から R へ移動することのみを考える. 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, L に残された点のゲインは単調に減少していくので, L の中でゲインが最大の点から順に R に移動させる. 以上をまとめ, 与えられた DRH $H=(V,E)$ に対して, オーバーヘッド総量 W_S を極小にするような点集合 V の, メモリバンクへの割り当てを求めるアルゴリズムを図 5 に示す.

[入力] DRH
[出力] 変数の左右メモリ割り当て

[Step 1] L = V, R=Φを初期解とする.
[Step 2] 全ての点のゲインを計算する.
[Step 3] Lにゲインが正の点があればゲインが最大の点$v_i \in L$をLから取り出し
Rへ移動し, Step 2へ.
[Step 4] 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.

図 5 最適化アルゴリズム

本アルゴリズムでは, 初期状態として全ての点を 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 R に移動する. g_i が最大の点を R に移動するため, 補題 1 より, 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, L に格納されている点のゲインは単調減少し, 補題 3 より, R に格納されている点のゲインは単調増加する. L にゲインが正の点がなくなれば, 移動を停止する.

5. 評価実験

本節では, 提案手法の有効性を示すために, Sample1, Sample2, Mandelbrot, DCT, Sample3 の 5 種類のプログラムに対し提案手法を適用し, 演算のオーバーヘッド総量 W_S の削減量, 実行サイクル数, および最適化に要する CPU 時間を評価した. Sample1, Sample2, Sample3 は 2 項演算と 3 項演算をそれぞれ 2 回と 1 回, 0 回と 5 回,59 回と 22 回使用するサンプルプログラム, Mandel はマンデルブロ集合を計算するプログラム, DCT は 8 点の離散コサイン変換のプログラムである.

実験環境は Pentium4 3GHz, 1GB メモリである. オーバーヘッド総量と実行サイクル数については, 変数を登場順に割り当てた場合を登場順, 3 項演算を 2 項演算とみなし, かつオペランド順を変更せずに割り当てた場合を DRG, 提案手法を DRH, 全探索による最適化を全探索として, 5 種類のプログラムについて実験した結果を表 1 に示す.

Sample1, Sample3 においては, DRH を用いてメモリバンクへの割り当てを最適化することで, オーバーヘッド総量を削減できた. また, DCT においては, オーバーヘッド総量は同じであったが, DRH を用いた方が, 実行サイクル数を削減できた. オーバーヘッド総量が同じであるにもかかわらず, 実行サイクル数に差が生じる理由は, DRG では 3 項演算であっても 2 項演算としてモデル化しているため, オーバーヘッド総量が 0 であっても, 完全にオーバーヘッドを削減できていない場合があるためである.

DRH のメモリバンクへの割り当てを全探索した結果, 提案手法は Sample3 を除いて, すべて最適解と一致した. Sample3 は, 全探索が実用的な時間内に終了しなかった. 最適化に要した時間は, 提案手法がいずれの場合も 1 秒以下であった. したがって, 本提案手法は短時間で良いメモリ割り当てを実現できた.

6. まとめ

本研究では, 超並列細粒度 SIMD プロセッサ MX のプロセッシングアレイ部分である MTA に対し, 変数と演算の関係とオペランド順を DRH によりモデル化することで, 変数のメモリバンクへの割り当てを最適化した. 実験により, 2 項演算をモデル化した DRG と比較して高速化されていることを確認した.

文献

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

[2] 中田 渚, 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.

[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] 中野光臣, 飯田全広, 末吉敏則, "アントコロニー最適化手法の Matrix Processing Engine への実装," 信学技報, Vol. 106, No.247, RECONF2006-27, pp. 1-6, 2006.

[5] 小橋 晶, 谷口一徹, 坂主圭史, 武内良典, 今井正治, "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, Vol. 107, No. 31, VLD2007-1, pp. 1-6, 2007.

[6] Jeonghun Cho, Yunheung Paek, and David Whalley, "Register and memory assignment for non-orthogonal architectures via graph coloring and MST algorithms," ACM SIGPLAN Notices, Vol. 37, Issue 7, pp. 130-138, 2002.

[7] Jeonghun Cho, Yunheung Paek, and David Whalley, "Fast memory bank assignment for fixed-point digital signal processors," ACM Transactions on Design Automation of Electronic Systems, Vol. 9, No. 1, pp 52-74, 2004.