Vol.2010-HPC-126 No.10
2010/8/3

SIMD 型計算機向けループ自動並列化手法

中村晃一[†1]　林崎弘成[†2]
稲葉真理[†1]　平木敬[†1]

GPGPU などの SIMD 型並列計算機が，その費用対性能の高さから注目を集めている．この様な計算機の性能を引き出す為にはデータの配置・メモリ階層間の転送を最適化する事，加えて転送のオーバヘッドを演算・転送のパイプライン化により隠蔽する事が重要となる．しかし，ソフトウェア制御のスクラッチパッドメモリを含むメモリ階層の特性を把握し手作業で最適化する事は困難であり，これらを最適化し自動的に並列化を行う並列化コンパイラの必要性が高まっている．データ配置・転送最適化では Copy-candidate Selection 法が最も成功している手法であるが，既存研究では演算・転送のパイプライン並列性を考慮に入れていない為に，ハザードが発生し後のパイプライン化を阻害してしまうという問題がある．

本稿で我々はプログラムの静的解析に基づき，データ配置・転送の最適化と演算・転送のパイプライン化を同時に行う手法:**Modulo Scheduling for Copy-candidate Selection**(MSCS) を提案する．これはデータ転送最適化手法とソフトウェアパイプライニング手法を混合整数計画法を用いて融合させることにより実現され，またその為に我々は **Selective Data Dependence Graph**(SDDG) というデータ構造を開発した．さらに東京大学に於いて開発された SIMD 型超並列計算機 GRAPE-DR を対象として我々が設計・開発を行った自動並列化コンパイラ NGC について, その実装と評価結果について述べる．

Auto Loop Parallelization for SIMD Architecture

Koichi Nakamura,[†1] Hiroshige Hayashizaki,[†2]
Mary Inaba[†1] and Kei Hiraki[†1]

Today, SIMD parallel computers such as GPGPU are getting attentions because of its high cost-effectiveness. For extracting their potential performance, data mapping optimization and data transfer optimization are important. Pipelining of transfer operations and computations to conceal overheads of communications is also important for more advanced speedup. The problem is that it is difficult to optimize them by hand for complicated hierarchical memory structure which contains software-controlled scratch pad memories. Therefore, automatic parallelizing compilers which optimize data mapping and transfer for SIMD parallel computers are anticipated. Copy-candidate Selection methods have been most successful approach to optimize data mapping and transfer. However, since these approaches do not consider influences of pipeline execution, there is a possibility of occurrence of pipeline hazards.

In this paper, we propose Modulo Scheduling for Copy-candidate Selection (MSCS) for obtaining optimal data mapping and transfer procedure under pipelined execution. To achieve the optimization, we combined Software pipelining method into Data transfer optimization method using mixed integer programming. We also propose Selective Data Dependence Graph (SDDG) which is a key structure for the formalization of MSCS. In this research, we developed auto parallelizing compiler NGC for GRAPE-DR, a massively parallel computer developed in The University of Tokyo. We will describe its implementation and evaluation results.

1. はじめに

近年 GPGPU などの SIMD 型並列計算機が，その費用対性能の高さから注目を集めている．また，東京大学に於いてさらにプロセッサの集積度を高めた超並列計算機 GRAPE-DR[1)] が開発されている．しかし，計算機の集積度が増加することによりプロセッサ当たりのメモリ容量とバンド幅が小さくなるとメモリウォール問題が顕著となる[2)]．この問題に対処する為，これらの計算機ではキャッシュメモリとソフトウェア制御のスクラッチパッドメモリ (SPM) から構成される階層型メモリアーキテクチャを採用している．

これらの計算機はそのアーキテクチャの特殊さ故に，プログラム作成の難易度の高さが問題となる．特に計算機の性能を引き出す為にはデータの配置・転送を最適化する事が重要となるが，メモリ階層の特性を把握し手作業で最適化する事は困難である．GPGPU に於いては，NVIDIA 社の CUDA・AMD 社の ATI Stream そして GPGPU 分野の標準技術として期待されている OpenCL が現在使用可能であるが，これらのプログラミングモデルでは各メモリ階層を明確に区別しており，ユーザがデータの配置を決定しなければならない．GRAPE-DR に於いては，データの配置に加え転送の粒度・タイミングも明示的に指定しなければならない．以上の様な背景から，最適なデータ配置・転送を自動的に生成しプログ

†1 東京大学
The University of Tokyo
†2 IBM 東京基礎研究所
IBM Research - Tokyo

Vol.2010-HPC-126 No.10
2010/8/3

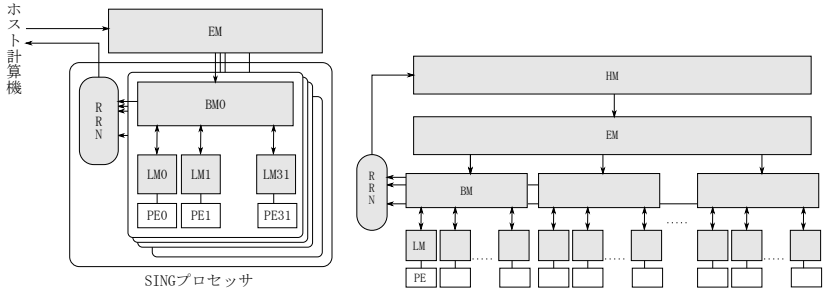
ラムを並列化する自動並列化コンパイラの必要性が高まっている．

プログラムの自動並列化は古くから粗粒度レベル・細粒度レベル共に種々の手法が提案されているが，本研究の対象である SIMD 型並列計算機では細粒度レベルの自動並列化，特にループの自動並列化が重要となる．ループの依存性解析[3),4)] に基づくループの自動並列化は，High Performance Fortran (HPF)[5)] などで既に実用化されており，加えてスカラエクスパンションなど並列度を向上させる最適化手法も提案され実装されている[6)–8)]．

一方，SPM からなる階層型メモリ向けのデータ配置・転送最適化は現在も盛んに研究がなされており，中でも **Copy-candidate Selection** に基づく手法は最も成功している手法である[9)–11)]．これは，データの配置・転送パターンの候補をプログラム解析により割り出し，それらから転送コストが最小となる候補を選択する手法である．しかしながら，以上の方法でメモリ間転送を最適化しても依然として大きい転送のオーバヘッドがプログラム実行のボトルネックとなってしまう．さらなる高速化の為には演算・転送をパイプライン化することにより通信時間を隠蔽する事が重要となるが，既存の転送コストの最小化に基づく Copy-candidate Selection 法ではこのパイプライン実行の影響をモデルに組み込んでいない為，生成されたコードでパイプラインハザードが生じてしまう問題がある．

そこで，本稿で我々はデータ配置・転送最適化とパイプライン化を同時に行う手法 **MSCS**(Modulo Scheduling for Copy-candidate Selection) 法を提案する．我々はデータ転送最適化手法である Copy-candidate Selection 法にソフトウェアパイプライニング手法である Modulo Scheduling 法を混合整数計画法 (MIP) を用いて融合させることによりこれを実現した．一般に MIP は計算量が大きくなるが，MSCS では転送の粒度に合わせて命令列の粒度を調整することにより計算量の削減を行う事が可能である．また，我々は C 言語と GRAPE-DR を対象として MSCS を実装した自動並列化コンパイラ **NGC** を開発した．本稿ではその設計・実装についても述べる．

本論文の構成は以下の様になっている．2 章では本研究で対象としている階層型メモリを持つ SIMD 型並列計算機のアーキテクチャと自動並列化の対象プログラムについて述べる．3 章では MSCS 最適化問題の定式化とアルゴリズムについて述べる．4 章では我々の開発した自動並列化コンパイラ NGC の設計について述べる．5 章では GRAPE-DR を対象とした NGC の実験結果について述べる．6 章では関連研究について述べる．7 章では本研究の総括を行う．



(a) SING プロセッサ　　(b) GRAPE-DR のメモリ階層

図 1　GRAPE-DR のアーキテクチャ
Fig. 1　Architecture of GRAPE-DR

2. 対象アーキテクチャとプログラム

本研究ではソフトウェア制御の SPM を含むメモリ階層を持つ SIMD 型並列計算機を対象としている．**図 1** は GRAPE-DR の SING プロセッサの構成とそのメモリ階層の概念図である．GRAPE-DR のメモリ階層はホスト計算機のメインメモリ (HM)・加速ボード上の外部メモリ (EM)・SING チップ内のブロックメモリ (BM)・各プロセッサエレメント (PE) のローカルメモリ (LM) からなる．また，縮約演算ネットワーク (RRN) が存在する事が特徴である．

本研究で対象とするプログラミング言語は C/C++・Fortran などの HPC 分野で広く使われている手続き型言語であり，並列化の対象は **do-all 型ループ**である．do-all 型ループは C 言語では以下の様に記述され

```
for (int i = 0; i < n; i++)
    statement
```

- `statement` 内で `i` への代入を含まない．
- ループ実行中 `n` が定数である．
- イテレーション間に依存関係がなく並列化が可能である．

という条件を満たすループの事である．一般のループの do-all 型ループへの変換・ループ依存の検査の為には種々の手法が提案されており，4 章で我々が採用した手法について述べる．

3. MSCS:Modulo Scheduling for Copy-candidate Selection

本章では **MSCS**(Modulo Scheduling for Copy-candidate Selection) について述べる．本手法はデータ配置・転送最適化手法である Copy-candidate Selection 法に基づいており

(1)　プログラムを解析しデータ配置・転送パターンの候補を抽出する．

(2)　最適な候補を選択する，

という 2 ステップからなる．MSCS は (2) のステップに於いて転送コストではなく総実行時間を最小化する事を目的とした手法であり，その為に Copy-candidate Selection 法に Modulo Scheduling 法を融合したものである．Modulo Scheduling 法とはソフトウェアパイプライニングを行う為の手法であり，これはデータハザード・構造ハザードを考慮に入れてループのパイプライン並列性を高める事を目的とした手法である．

これらの融合の為に我々は **Selective Data Dependence Graph**(SDDG) というデータ構造を開発し，混合整数計画法 (MIP) により定式化を行った．以下，MSCS の定式化の為のプログラムのモデル化・SDDG・最適化問題の定式化・最適化のアルゴリズムについて述べる．

3.1 プログラムのモデル化

我々のモデルでは入力ループ本体を **Processing Block** と呼ばれる複数のブロックに分割する．例えば，1 つの Processing Block はインストラクション単位スケジューリングでは連続する複数インストラクション，スレッド単位スケジューリングでは個々のスレッドに対応するものと考えれば良い． また，メモリ階層間のデータ転送は **Copy-candidate Block** と呼ばれるブロックにより表現する．MSCS ではこの 2 種類のブロック単位でのスケジューリングを行う．

このブロック化によりスケジューリングの粒度を調整する事ができ MIP の計算量を削減する事が可能となる．また，インストラクション単位のスケジューリングと異なりブロック化スケジューリングでは各ブロックの実行タイミングをクロック単位で取り扱う必要がなくなる．その為，後の節で定義されるスケジューリングに関する変数を実数近似する事ができ，これも MSCS の計算量削減に貢献している．

最後に，実数値パラメータ II (Initiation Interval) を各イテレーションの実行時間として定義する．この II を最小化することが我々の目標である．

3.2 Selective Data Dependence Graph

Copy-candidate selection 問題を定式化するにあたって，我々は Data Dependence Graph と Copy-candidate Graph[9] を融合したデータ構造:**Selective Data Dependence Graph**(SDDG) を提案する．SDDG は Data Dependence Graph の各頂点・辺に選択性を持たせることにより Copy-candidate selection 問題を SDDG 上での部分グラフ選択問題として表現し直したものであり，スケジューリング最適化と転送最適化を同時に行うことを可能とする．**図 2** が SDDG の例である．

SDDG は有向グラフ (V, E) であり，$B_i \in V$ は Processing Block もしくは Copy-candidate Block に対応する．$(i, j) \in E$ は B_i から B_j への両者が実行される場合のデータ依存を表す．すなわち $(i, j) \in E$ が存在し，かつ MIP ソルバが B_i と B_j を実行ブロックとして選択した場合には，B_i は B_j よりも前方にスケジュールされなければならない．各エッジ $(i, j) \in E$ には以下のパラメータが割り当てられている．

$D_{ij} \equiv B_i$ の開始から B_j の開始までに必要とする時間.

$\Delta_{ij} \equiv B_i$ の開始 B_j の開始までに必要とするイテレーション数.

すなわち $\Delta_{ij} = \delta$ の時，B_i の k 回目の実行は B_j の $k+\delta$ 回目の実行よりも前にスケジュールされなければならない．また，部分グラフ選択問題の定式化の為 $0-1$ 変数 s_i を各頂点 $B_i \in V$ に，d_{ij} を各エッジ $(i, j) \in E$ に割り当てる．

$$s_i \equiv \begin{cases} 1, & B_i \text{ を実行する場合} \\ 0, & \text{上記以外} \end{cases} \qquad d_{ij} \equiv \begin{cases} 1, & B_j \text{ が } B_i \text{ に依存する場合} \\ 0, & \text{上記以外} \end{cases} \tag{1}$$

SDDG の定義により $s_i = 1$ かつ $s_j = 1$ の時のみ $d_{ij} = 1$ となる．すなわち $d_{ij} = s_i s_j$ である．Watters の reduction 法[12] を用いると，この制約は以下の線形式で表す事ができる．

$$\begin{aligned} 2d_{ij} &\leq s_i + s_j \\ d_{ij} + 1 &\geq s_i + s_j \end{aligned} \tag{2}$$

但し Processing Block は必ず実行されるので常に $s_i = 1$ である．

3.3 最適化問題の定式化

3.3.1 Valid Copy-candidate Constraints

データが正しく転送される為には，SDDG から適切な Copy-candidate Block の集合を選択しなければならない． この制約を **Valid Copy-candidate Constraints** と呼ぶ．この制約を立式する為，我々は **Producer Set**:PS_x^i を以下の様に定義する．

$$PS_x^i \equiv \text{データ } x \text{ を } B_i \text{ に転送する全てのブロック} \tag{3}$$

もし B_i が実行されるならば ($s_i = 1$ ならば)，B_i が必要とする全てのデータ x に対して

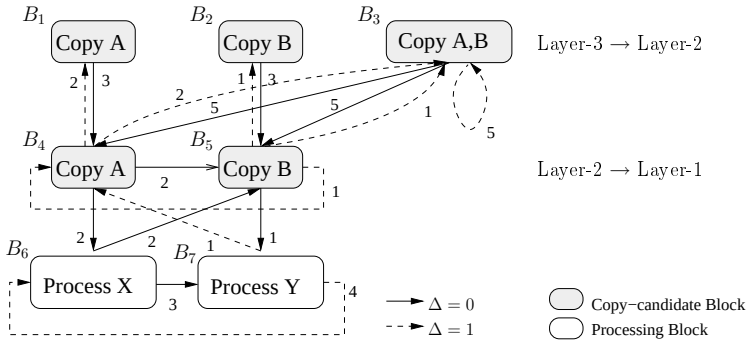


図 2 Selective Data Dependence Graph

PS_x^i 内の一つのブロックが実行されなければならない．従って本制約は次の様に定式化される．

$$s_i \left(\sum_{j \in PS_x^i} s_j - 1 \right) = 0 \tag{4}$$

既に 3.2 節で定義した d_{ij} を用いると Valid Copy-candidate Constraints の線形式が得られる．

$$\sum_{j \in PS_x^i} d_{ji} = s_i \tag{5}$$

3.3.2 Periodic Scheduling Constraints

ブロック B_i のスケジューリング σ は

$$\sigma_k^i \equiv B_i \text{ の } k \text{ 回目の実行開始時間} \tag{6}$$

と表現できる．ここで我々は σ が **Periodic Scheduling Constraints**[13] を満たし

$$\sigma_k^i = t_i + (k-1) \cdot II \tag{7}$$

と表される事を仮定する．但し $t_i = \sigma_1^i$ と定義する．加えて，整数変数 l_i と実数変数 $o_i (< II)$ を以下を満たす値として定義する．

$$t_i = l_i \cdot II + o_i \tag{8}$$

3.3.3 Precedence Constraints

B_j が B_i に依存する場合はその実行順序に関し以下の制約を満たす必要がある．

$$t_j - t_i \geq D_{ij} - II \cdot \Delta_{ij} \tag{9}$$

さらに、我々の定式化に於いては $d_{ij} = 0$ のとき，すなわち両ブロックが実行される場合以外にはこの式は解空間にいかなる制約も課してはならない．そこで我々は Big-M 法を利用して以下の様に制約を修正する．

$$M \cdot (1 - d_{ij}) + t_j - t_i \geq D_{ij} - II \cdot \Delta_{ij} \tag{10}$$

M は $d_{ij} = 0$ の際に左辺が必ず右辺より大きくなるような十分に大きい定数である．もし $d_{ij} = 1$ ならば本式は (9) と一致し，$d_{ij} = 0$ ならば本式はいかなる制約も課さない．ここで M が大きすぎると MIP 問題を解く際に十分な精度が得られない場合があるが，我々の定式化に於いては $M = \sum_{(i,j) \in E} D_{ij}$ であれば十分である．

3.3.4 Resource Constraints

B_i と B_j が同じ資源を使用する場合にはこれらを同時に実行する事が出来きない為

$$t_i + E_i \leq t_j \text{ and } t_j + E_j \leq t_i + II \tag{11}$$

もしくは

$$t_j + E_j \leq t_i \text{ and } t_i + E_i \leq t_j + II \tag{12}$$

のいずれかが成立しなければならない． ここで E_i と E_j は B_i と B_j の実行時間である．この条件は $0-1$ 変数 p_{ij} を使用することで線形制約式としてに表す事ができる．

$$\begin{aligned} t_i + E_i &\leq t_j + p_{ij} \cdot II \\ t_j + E_j &\leq t_i + (1 - p_{ij}) \cdot II \end{aligned} \tag{13}$$

もし $p_{ij} = 0$ ならばこれらの式は (11) と一致し，$p_{ij} = 1$ ならば (12) と一致する．

加えて，Precedence Constraints と同様に Big-M 法を利用して以下の様に修正を行う．

$$\begin{aligned} t_i + E_i &\leq t_j + p_{ij} \cdot II + (2 - s_i - s_j) \cdot M \\ t_j + E_j &\leq t_i + (1 - p_{ij}) \cdot II + (2 - si - s_j) \cdot M \end{aligned} \tag{14}$$

ここでも $M = \sum_{(i,j) \in E} D_{ij}$ とすれば良い．

3.3.5 目 的 関 数

MSCS では与えられた II に対して実行可能解が存在するか否かが分かれば良い．従って目的関数は必ずしも必要ではないが，II の最小性を損なわずに付加的な最適化を行う事が可能である．例えば電力消費を最小化したい場合には以下の目的関数を使用する事が出来る．

$$\text{minimize} \sum_{B_i \in V} C_i \cdot s_i \tag{15}$$

ここで C_i は B_i が実行された場合のコストであり，総コストを線形和として近似したものである．

3.3.6 MSCS 最適化問題

対象プログラムの SDDG を (V, E) とし以上をまとめると，MSCS の最適化問題は以下の様になる．

maximize (or minimize) an objective function

subject to Valid Copy-candidate Constraints

$$\sum_{i \in PS_x^i} d_{ji} = s_i$$

Precedence Constraints

$$M \cdot (1 - d_{ij}) + t_j - t_i \geq D_{ij} - II \cdot \Delta_{ij}$$

Resource Constraints

$$t_i + E_i \leq t_j + p_{ij} \cdot II + (2 - s_i - s_j) \cdot M$$

$$t_j + E_j \leq t_i + (1 - p_{ij}) \cdot II + (2 - si - s_j) \cdot M$$

for each $B_i, B_j \in V$ which require same resource.

$$s_i, d_{ij} \in \{0, 1\}, \qquad 0 \leq t_i \leq M, \qquad 0 \leq o_i < II,$$

$$l_i \in \left\{0, 1, \cdots, \left\lceil \frac{M}{II} \right\rceil\right\}, \qquad M = \sum_{(i,j) \in E} D_{ij},$$

$$2d_{ij} \leq s_i + s_j, \qquad d_{ij} + 1 \geq s_i + s_j, \qquad t_i = l_i \cdot II + o_i$$

3.4 最適化のアルゴリズム

MSCS 最適化は以下の 3 ステップに分けられる．

(1) ループ本体を Processing Block の集合に分割する．

(2) Copy-candidate を抽出し，選択的データ依存グラフを構築する．

(3) MIP 最適化問題を生成し，それを解く．

3.1 節で述べた様に，ループ本体の Processing Block への分割方法には自由度があり．最終的に得られるスケジューリングの効率と MSCS の計算量はこの分割方法に依存する．

Copy-candidate の抽出には既存の解析手法を利用することができる．Diguet ら[10] は単一プロセッサアーキテクチャ用の手法を, Issenin ら[11),14)] はマルチプロセッサアーキテクチャ用の手法を，そして林崎ら[9] 超並列計算機用の手法 (MCAMP) を提案している．これらの手法を利用し，Copy-candidate Block の抽出とそれらの依存関係を解析することができる．また，3.3.1 節で述べた Producer Set も同時に決定することができる．最後に，Processing Block と Copy-candidate Block 間の依存エッジを追加することで SDDG の構築が完了する．

以上により MSCS 最適化問題を生成した後 II を最小化する解を求める．MSCS 最適化問題を MIP ソルバで解くことにより実行可能解が存在するか否かが分かるので，II の値について二分探索を行うことにより最小の II を求める事ができる．

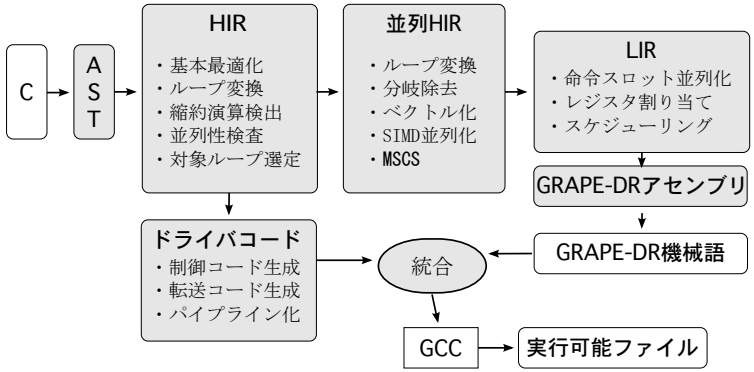


図 3 NGC のコンパイルフロー
Fig. 3 Compilation flow of NGC

4. 自動並列化コンパイラ NGC

提案手法は一般的な SIMD 型計算機と手続き型言語に対して適用が可能であるが，本研究では GRAPE-DR と C 言語を対象として自動並列化コンパイラ NGC を開発した．NGC は元のソースコードに並列性指示ディレクティブの追加など一切の変更を施すことなく，GRAPE-DR 用プログラムにコンパイルできる様に設計されている．

実装は C 言語を用いフルスクラッチで行った．**図 3** にその構成を示す．網掛け部が本研究に於いて新規に開発を行った部分である．NGC は高レベル (HIR)・高レベル並列 (並列 HIR)・低レベル (LIR) の 3 種類の中間言語を使用する．生成された GRAPE-DR 機械語とホスト計算機で実行されるドライバコードは一つの C 言語プログラムへと統合され，GCC(the GNU Compiler Collection) により実行可能ファイルへとコンパイルされる．本章では各中間言語において行われるコード変換・最適化について述べる．

4.1 HIR

このフェーズの目的はプログラムから並列化可能な do-all 型ループを検出し，GRAPE-DR 上で実行する並列 HIR とホスト計算機上で実行するドライバコードに分離する事である．まず入力プログラムに対し各種基本最適化:定数伝播・コピー伝播・無用命令除去・関数呼び出し展開を行った後，ループの並列性を向上させる為にループ交換・ループスキューイング・スカラエクスパンション・縮約演算除去・帰納変数展開[6)–8)] を行う．縮約演算の検

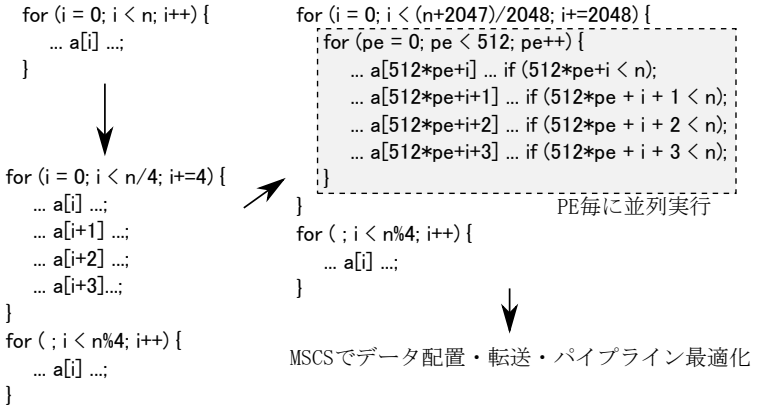


図 4 ループの並列化の流れ
Fig. 4 Loop parallelization sequence

出・帰納変数展開は HIR を静的単一代入形式 (SSA 形式) へと変換した上で行い，縮約演算は RRN への書き込み命令へと変換する．その後実際に並列化が可能であるループの検出を GCD テスト・Banerjee テスト[3),4)] により行う．

4.2 並列 HIR

このフェーズでは並列化可能なループをプロセッサの数やメモリ容量などのハードウェアの制約を満たす様に以下の手順により変換する．また，本稿で提案した MSCS によりデータ配置・転送の最適化を行う．

(1) SING プロセッサの各 PE は長さ 4 のベクトルプロセッサであるので対象ループをストリップマイニングしベクトル化する．
(2) メモリ容量の範囲内でループアンローリングを行う．
(3) ストリップマイニングし PE レベルで並列化を行う．

図 4 はこの一連の変換の流れを表している．以上の後，3.4 節で述べた手順で MSCS を実行してデータの配置の決定・転送コードの生成・ソフトウェアパイプライニングを行う．現在の実装では同一のメモリロケーションにアクセスする連続するインストラクション列をグループ化したものを Processing Block として扱っている．

4.3 LIR

SING の各インストラクションには (1) 整数演算命令 (2) 浮動小数点数命令 (3)BM 制御命令 (4)EM・RRN 制御命令の 4 スロットがありこれらを並行して実行する事ができる．NGC はリストスケジューリングを用いて貪欲法により各スロットに命令を埋めていく方法を採用している．その後，グラフ彩色法に基づきレジスタ割り当てを行う．この際，各 PE に存在する高速なレジスタ間転送を可能とするフィードバックレジスタに高い優先度を設定している．最後に LIR は GRAPE-DR アセンブリ言語へと変換され，アセンブラによって GRAPE-DR 機械語へとコンパイルされる．

4.4 ドライバコード

ホスト計算機では，SING チップの初期化などの制御・HM から EM への転送・縮約演算の結果の RRN からの回収を行う．多くのプログラムでは，この部分は (1)HM から EM への転送，(2)SING での演算の実行，(3)RRN からの結果回収の 3 つの処理からなるループとなる．ここでホスト計算機・GRAPE-DR 間の転送は大きなコストがかかる為，可能ならば NGC はソフトウェアパイプライニングを行う．これにより転送と演算をオーバラップさせる事が出来，通信のオーバヘッドが隠蔽される．

5. 評　　価

本章では，提案手法の現実のプログラムへの適用可能性と最適化性能の評価結果について述べる．まず理研の姫野ベンチマーク[15)] と SPEC CPU2006 ベンチマーク CFP[16)] に含まれるアプリケーションについて NGC により自動並列化が可能かどうか実験した．CFP に含まれるベンチマークのうち，C++もしくは Fortran で書かれているものは，これらのフロントエンドを未実装の NGC ではコンパイルを行う事ができない為，NGC と同じアルゴリズムで自動並列化が可能な構造をしているか否かに関して検証を行った．

結果，姫野ベンチマークと CFP に含まれる 17 のベンチマークの内 C 言語のみで実装されている milc と lbm は NGC により自動的に並列化が可能であった．また CFP 内の Fortran/C++で記述された 10 のベンチマークも並列化が可能であることが確認された．soplex・sphinx3 はツリー型データ構造を主に扱う為ループ並列化に基づく並列化はできず，cactusADM・namd は配列の動的スライシング・関数ポインタといった動的な要素により本手法では並列化ができなかった．また，gromacs はポインタ演算を多用しており並列性の検査が困難であるが，並列性指示文の追加により並列化が可能である．

続いて我々MSCS の最適化性能の実験を行った．既に述べた姫野ベンチマーク・SPEC CPU2006 CFP については，主にバックエンド部の実装に不備があり，実機で動作させるところまで至っていない．そこで，以下の 3 つのシンプルなベンチマークについてそのルー

SMUL	# of variables	minimum II	optimality	Solving time
提案手法	146	264	x 1.03	2.42 sec
既存手法	18	296	x 1.16	0.41 sec
手動最適化		256	1	
DMUL	# of variables	minimum II	optimality	Solving time
提案手法	146	536	x 1.05	1.48 sec
既存手法	18	568	x 1.11	0.32 sec
手動最適化		512	1	
GRAVITY	# of variables	minimum II	optimality	Solving time
提案手法	131	896	x 1	1.25 sec
既存手法	24	896	x 1	0.95 sec
手動最適化		896	1	

表 1 実験結果
Table 1 Experimental results

プ開始間隔の測定を行った．GRAPE-DR の機能はソフトウェアにより制御されている為，ここでの測定結果は実機で実行した場合と大きく変わらないと考えてよい．

SMUL 単精度密行列乗算 $C = A \cdot B$. 16×32 型と 32×16 型の部分行列積を各 PE で計算し，RRN によって縮約する．

DMUL 倍精度密行列乗算 $C = A \cdot B$. 8×32 型と 32×8 型の部分行列積を各 PE で計算し，RRN によって縮約する．

GRAVITY 以下の方程式を利用した重力多体計算．

$$\vec{F_i} = -\sum_{j \neq i} m_j \frac{\vec{r_i} - \vec{r_j}}{(|\vec{r_i} - \vec{r_j}|^2 + e_j^2)^{3/2}} \tag{16}$$

MSCS では混合整数計画法を用いているが，ソルバとして `lp_solve` C-API version 5.5.0.6[17] を使用し，最適化の実行時間はソルバの総実行時間とした．また既存手法の評価は以下の手順で行い，最適化の実行時間はこれらの実行時間の和とした．

- 転送コストのみに注目し最適な Copy-candidate を選択.
- 生成されたコードに対して Modulo Scheduling を実行.

最適化性能は手動で注意深く記述した GRAPE-DR アセンブリプログラムのイテレーション実行時間を 1 としてそれに対する比を測定した．**表 1** が実験結果である．

これらの結果は MSCS が既存手法よりもより良い最適化性能を達成する事を示している．また，手動最適化に比べて 5 %以内の性能を達成する事が出来ている．転送コストのみに注目して最適化した既存手法の場合，コストを最小にする為に大きなデータサイズでの EM-BM 転送が行われる．結果としてデータハザードが発生し，EM-BM 転送と BM-LM 転送のパイプライン化が阻害されてしまっている．MSCS が生成した結果では EM-BM 転送を複数回に分けることにより，EM-BM 転送と BM-LM 転送をパイプライン化して実行する事が可能となっている．結果として，総実行時間は MSCS が生成したものの方が小さくなっている．GRAVITY の結果では転送時間に対する演算時間の比が非常に大きい為，転送のオーバヘッドが総実行時間に影響を与えない．従って MSCS と既存手法は同一の結果を生成した．

6. 関 連 研 究

MSCS は Modulo Scheduling 法の MIP モデルに基づいている．Altman ら[13] は構造ハザードを考慮したモデルを提案している．MIP に基づく手法は高い最適化性能を保証する反面，計算量が大きくなってしまう．一方，ヒューリスティクスに基づく計算量の少ない手法も提案されている．Rau ら[18),19)] はヒューリスティクス法に関する先駆的な研究を行った．そして Llosa ら[20] はより洗練された Swing Modulo Scheduling 法を提案している．MSCS はブロック単位でのスケジューリングを行うので粒度の調節が可能であり，計算量の削減が可能な手法である．

Diguet ら[10] は階層型メモリを持つ単一プロセッサアーキテクチャ用に Copy-candidate Selection に基づいてデータ転送を最適化する手法を提案している．Issenin ら[11),14)] はマルチプロセッサアーキテクチャ用に DRDM という手法を提案し, 林崎ら[9] は DRDM を超並列計算機向けに拡張した MCAMP という手法を提案している．MSCS も同様に Copy-candidate Selection に基づいているが，Modulo Scheduling 法を融合した為にパイプライン実行性能を損なわない点が異なる．

Kennedy ら[5] は High Performance Fortran(HPF) を標準化しその自動並列化コンパイラを開発した。HPF は古くから自動並列化を取り入れているプログラミング言語であり，NGC の開発においても大いに参考にしている．笠原ら[21] はマルチグレイン並列化コンパイラ OSCAR を開発しており，その最適化性能の高さから注目を集めている．OSCAR コンパイラは主に粗粒度並列性に注目して並列化を行うコンパイラであり NGC とはその手法・対象が異なる．

7. ま と め

我々は，スケジューリングとデータ転送最適化を融合した最適化手法 MSCS 法を提案

した．加えて，MSCS を中核とした手続き型プログラムのループ自動並列化手法を提案した．我々は MSCS 最適化問題を定式化する為に頂点・辺に選択性を持たせた SDDG を定義し，SDDG 上で選択されなかった処理に起因する制約条件を除去する為の変更を Modulo Scheduling 法の MIP モデルに対して行った．また，ブロック単位のスケジューリングを行う事により MIP の計算量を削減し，以上を現実的な時間で実行できるアルゴリズムを開発した．本手法は現実に使用されているプログラムに対し変更を行うことなく自動並列化・最適化を行う事が可能であり，我々の開発した自動並列化コンパイラ NGC に実装されている．

NGC は実際のアプリケーション開発に使用可能なコンパイラとして設計・開発を続けてきた．今回は実装の不備により SPEC CPU2006 などのベンチマークを実機上で動作させる事が出来なかったが，今後も開発を継続し実機でのその最適化性能の検証を行っていく予定である．また，科学技術計算プログラムの多くが Fortran や C++でも記述される事から，それら言語の為のフロントエンドの開発，GPGPU などの他の計算機用のバックエンドの開発も検討しているところである．加えて，今回提案した SDDG に基づく転送とスケジューリングの同時最適化は他の粒度の並列化に対しても応用可能であり，本分野における本手法の有用性を継続して検証していくつもりである．

参 考 文 献

1) Makino, J., Hiraki, K. and Inaba, M.: GRAPE-DR: 2-Pflops massively-parallel computer with 512-core, 512-Gflops processor chips for scientific computing, *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ACM (2007).
2) Wulf, W. and McKee, S.: Hitting the Memory Wall: Implications of the Obvious, *ACM SIGARCH Computer Architecture News archive*, Vol.23, No.1, pp.20–24 (1995).
3) Banerjee, U.: Data dependence in ordinary programs, *scanning*, Vol.1, No.05, p.1.
4) Banerjee, U.: Speedup of ordinary programs (1979).
5) Specification, H.: High Performance Fortran Forum, *Center for Reaserch on Parallel Computation, Rice University, Rice* (1993).
6) Lamport, L.: The parallel execution of DO loops, *Communications of the ACM*, Vol.17, No.2, p.93 (1974).
7) Wolfe, M.: Loops skewing: The wavefront method revisited, *International Journal of Parallel Programming*, Vol.15, No.4, pp.279–293 (1986).
8) Pieper, K.: Parallelizing compilers: implementation and effectiveness (1993).
9) Hayashizaki, H., Sugawara, Y., Inaba, M. and Hiraki, K.: MCAMP: communication optimization on massively parallel machines with hierarchical scratch-pad memory, *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, New York, NY, USA, ACM, pp.102–111 (2008).
10) Diguet, J.P., Wuytack, S., Catthoor, F. and Man, H.D.: Formalized methodology for data reuse exploration in hierarchical memory mappings, *ISLPED '97: Proceedings of the 1997 international symposium on Low power electronics and design*, New York, NY, USA, ACM, pp.30–35 (1997).
11) Issenin, I., Brockmeyer, E., Durinck, B. and Dutt, N.: Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies, *DAC '06: Proceedings of the 43rd annual conference on Design automation*, New York, NY, USA, ACM, pp.49–52 (2006).
12) Watters, L.: Reduction of integer polynominal programming problems to zero-one linear programming problems, *Operations research*, Vol.15 (1967).
13) Altman, E.R., Govindarajan, R. and Gao, G.R.: Scheduling and mapping: software pipelining in the presence of structural hazards, *Proceedings of the ACMSIGPLAN 1995 conference on Programming Language Design and Implementation*, Vol. 30, No.6, pp.139–150 (1995).
14) Issenin, I. and Dutt, N.: Data reuse driven energy-aware MPSoC co-synthesis of memory and communication architecture for streaming applications, *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, New York, NY, USA, ACM, pp.294–299 (2006).
15) Himeno, R.: Himeno Benchmark, `http://accc.riken.jp/HPC/HimenoBMT.html`.
16) Henning, J.: SPEC CPU2006 benchmark descriptions, *ACM SIGARCH Computer Architecture News*, Vol.34, No.4, p.17 (2006).
17) Berkelaar, M.: lp_solve, `http://sourceforge.net/projects/lpsolve/`.
18) Rau, B. and Glaeser, C.: Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing, *ACM SIGMICRO Newsletter*, Vol.12, No.4, pp.183–198 (1981).
19) Rau, B.: Iterative modulo scheduling: an algorithm for software pipelining loops, *Proceedings of the 27th annual international symposium on Microarchitecture*, ACM New York, NY, USA, pp.63–74 (1994).
20) Llosa, J., Gonzalez, A., Ayguade, E. and Valero, M.: Swing Modulo Scheduling: A Lifetime-Sensitive Approach, *PACT*, Vol.96, pp.20–23.
21) Kasahara, H., Obata, M. and Ishizaka, K.: Automatic coarse grain task parallel processing on smp using openmp, *Languages and Compilers for Parallel Computing*, pp.189–207 (2001).