自動並列化コンパイラについて まとめたこと

舟本 一久

april 27.1995

1 はじめに

次世代スーパコンピュータとして超並列計算機に期待がよせられるに従い、その重要性が日に日に増しているといっても過言ではない自動並列化コンパイラについて述べます。

先ず簡単に並列化コンパイラの必要性とその役割について述べた後、プログラム並列化のための基礎技術について解説します。そして、最新応用技術にふれた後で、具体的な自動並列化コンパイラを紹介します。

2 並列化コンパイラの役割

自動並列化コンパイラとは、逐次言語によって記述されたプログラムを自動的に並列化するものです。

１つのプログラムの処理を並列処理によって高速する場合、以下に示す並列化のための作業が必要となります。

- プログラムをタスクに分割
- タスク間の並列性の検出・指示
- 各タスクのプロセッサへの配置
- プロセッサ間のデータ通信コード・同期コードの挿入

一般に、これらの作業を行うには大きく分けて２つのアプローチがあります。１つは、これらの項目をプログラム中に陽に記述することのできる、並列記述プログラミング言語を使用し、ユーザがこれらの作業を行うというアプローチ、もう１つは、ユーザは並列性を意識せずにプログラムし、並列化はシステムが行うというアプローチです。前者には、ハードウェアの性能を十分に利用できる可能性がある、というメリットがある反面、性能を追求する余り、ハードウェアに依存したポータビリティリーの無いコードとなってしまう、プログラムの開発・デバッグが複雑になる、などのデメリットがあります。後者のアプローチは、既に存在するプログラムの並列化を自動的に行うことに加え、今日では、プログラムの生産性及びポータビリティーの維持のために、新規のプログラム開発にも必要であるとの考え方が広まっています。また、並列化の作業は複雑かつ単調で、ユーザが並列化の全てを行うことは不可能であるとも考えられています。

しかしながら、プログラムの′意味′が分からないと並列性が認識できないような部分が存在するのも確かで、ユーザによる並列性指示と、コンパイラによる並列性検出を組み合わせた形のアプローチが望まれています。

3 並列化コンパイラの基礎技術

一般に並列化コンパイラはプログラムを以下のような手順で並列化します。

1) データの流れ及び制御（条件分岐など）の流れの解析
2) 並列実行可能な部分の検出
3) プログラムを並列処理できる形へ自動的に変換することによるプログラム中からの並列性の検出
4) 並列実行可能部分のプロセッサへのスケジューリング
5) スケジューリング結果に基づいた並列化コードの生成

従来、並列化の主な対象となってきたのはループであり、ループに対する並列化技術が最も進んでいます。というのも、ループには多くの並列性が期待できる、ループ処理時間がプログラム実行時間に占める割合が大きい、解析が行いやすい、という理由があるからです。その中でも Doall 処理及び Doacross 処理と呼ばれる、Do ループの各イタレーションを異なるプロセッサに割り当てる方法が一般的です。

```
        do I=1,N
                do J=2,N
S1:                     C(I,J)=A(I,J-1)+B(I,J)
S2:                     D(I,J)=3.0*C(I,J)
S3:                     A(I,J)=C(I,J)/2.0
S4:                     B8I,J)=A(I,J)+D(I,J)
                end do
        end do
```

図 1: Doall 及び Doacross ループの例

(a) Doall 処理

図 1のような Do ループは I=1,I=2,···,I=N の各イタレーションは全く独立に計算可能です。このそれぞれのイタレーションを別々のプロセッサに割り当て実行するのが Doall 方式です。

(b) Doacross 処理

一方、J に関しては、J=k の s1 を処理するときに、J=k-1 の s3 の結果を待たねばなりません。このようにあるイタレーションの途中の処理を行うのに他のイタレーションの処理の途中結果を待たねばならないものを Doacross 処理といいます。

スタティックなスケジューリング法は、実行時のスケジューリングオーバーヘッドがない、プロセッサ間データ転送を最小化することが可能であるという長所がありますが、ループ中に条件分岐などがあり、各イタレーションの処理時間が大きく変動する場合には、プロセッサ間の負荷バランスが崩れることがあります。

このようなイタレーション間の処理時間の大きな変動による負荷バランスの崩れを避けるために、セルフスケジューリング、チャンクスケジューリング、ガイデッドセルフスケジューリングなどと呼ばれるダイナミックスケジューリング方式が提案されています。

- セルフスケジューリング

  共有メモリ上に処理を終えたイタレーション数を記憶しておくカウンタを用意し、各プロセッサが１イタレーションの処理を終えるとそのカウンタを参照し、次に自分が処理すべきイタレーションを１つ決め、カウンタをインクリメントするという方式です。

  この方式は、１イタレーション毎にスケジューリングを行うため、１イタレーションの処理に要する時間が大きく、また各イタレーションの処理時間の変動が大きいときに有効です。

- チャンクスケジューリング

  処理を終えたプロセッサが、事前に設定された数の複数イタレーションを取ってくる方式です。

  この方式は、セルフスケジューリングよりオーバーヘッドが小さくなるという長所がありますが、プロセッサ負荷がアンバランスになるという欠点もあります。

- ガイデッドセルフスケジューリング

  セルフスケジューリングとチャンクスケジューリングの欠点を補おうとする方式です。各プロセッサが次に処理すべきイタレーションを決める際、その時点で残っているイタレーション数 Ri をプロセッサ数 m で割った数、すなわち

  X=[Ri/m]　　(但し [Ri/m] は Ri/m 以上の最小の整数）

  をこのイタレーションを次に処理するイタレーションとして獲得する方式です。

実行時のオーバヘッドを減らす為にはできるだけコンパイル時に行うスタティックスケジューリングが望ましいのですが、やむを得ずダイナミックスケジューリングをする場合でも、コンパイル時にできるだけ必要な情報を収集したり、ダイナミックスケジューリングコード自体をソースプログラム毎に最適化してコンパイラが生成するなどして、オーバヘッドを減らす工夫をすべきであるとの認識が広まっています。

コンパイラは、ループが Doall 可能か Doacross 可能かをチェックしなければなりません。すなわちイタレーション間にわたるデータ依存解析を行い、依存関係を検出します。現在のマルチプロセッサでは、各プロセッサが演算パイプラインを持っている場合が多く、この場合

には多重ループの外側のループに Doall 処理を行い、最内側ループをベクトル処理を行う方式がとられます。このとき両方の性能を引き出すために、ループインタチェンジ（内外ループの入れ替え）などが必要に応じて適用されます。

しかし、ある多重ループがインタチェンジ可能であっても、どのループにインタチェンジを適用し、どのループを Doall、或いは Doacross し、どのループをベクトル化するかは難しい問題であり、並列性、メモリアクセスのオーバーヘッドなど種々のファクタを考慮して決定されねばなりません。Kuck&Associate 社の KAP のようなリストラクチャラ (restructurer) は、ループインタチェンジ可能でかつ並列化及びベクトル化の両方が可能な多重ループに対して、並列化とベクトル化を適用できる全ての組み合わせの処理時間を推定し、最小の処理時間を与える組み合わせで、並列化及びベクトル化を行っています。

4 最新技術について

次に、単一のプログラムを並列処理マシン上で処理する為の並列化コンパイラの研究動向を簡単に解説します。ここでは、プログラム全域にわたる並列処理を行う為に現在行われている研究を、並列単位の大きさにより細粒度（命令或いはステートメントレベル）、中粒度（Do ループイタレーションレベル）、粗粒度（Do ループ間、サブルーチン間、基本ブロック間）の 3 種に大きく分類しながら説明します。また、最後に、分散メモリマルチプロセッサシステムのための並列化技術に関しても説明します。

4.1 細粒度タスクの並列処理

イリノイ大は Parafrase（リストラクチャリングプリプロセッサ）プロジェクトで、変数名をコンパイラが自動的に付け替えることにより、ステートメントレベル間の逆依存・出力依存を取り除き、並列性を抽出するスカラリネーミング、算術式の計算を行う際に交換・結合・分配法則を使い、命令間の並列性を引き出す計算木を作成するツリーハイツリダクションなどの細粒度の並列処理に有効な種々の手法を開発しました。しかし、現在イリノイ大で行われている CEDAR プロジェクトでは、ハードウェアの性能などを考慮して細粒度タスクの並列処理はインプリメントされていません。

現在マルチプロセッサシステム上で、複数命令或いはステートメントからなる細粒度タスクの並列性を実際にインプリメントしているのは、早大の OSCAR プロジェクトのみです。

OSCAR コンパイラでは

(1) データ依存のみを持つ細粒度タスク間の並列性抽出

(2) タスクの CP/DT/MISF アルゴリズム（プロセッサ間データ転送を考慮し、実行時間の最小化を行うヒューリスティックアルゴリズム）を用いたプロセッサへのスケジューリング

(3) スケジューリング結果を使用した各プロセッサ用の機械語プログラムの生成

(4) 同一プロセッサに割り当てられたタスク間でのデータ授受のためのレジスタ最適利用、冗長な同期命令除去による同期オーバヘッドの最小化といった並列機械語プログラムの最適化

を行うことにより細粒度タスクの並列処理を可能にしています。

このほかに細粒度タスクの並列性を引き出そうとしているプロジェクトしては、IBM ワトソン研究所の PTRAN プロジェクトがあります。PTRAN では、条件分岐（制御依存）を考慮して各ステートメント間の並列性を抽出しようとしています。PTRAN コンパイラは、ステートメントをノードとしたコントロールフローグラフを作成し、ポストドミネート[1]という概念を用いてステートメント間のコントロール依存[2]関係を見つけ、ディペンデンスグラフを作成します。このディペンデンスグラフは、もしステートメント間にデータ依存がないならば、ステートメント間の最大並列性を表します。現在 PTRAN では、データ依存を無視してディペンデンスグラフを作成した後、データ依存を満たすように IBM Parallel FORTRAN を用いてステートメント間の並列性を記述するというアプローチをとっています。但しこの方式では、データ依存をディペンデンスグラフ作成後に考慮するため、データ依存を考慮した段階で、多くの並列性が失われてしまうという問題があります。

---

[1] コントロールフローグラフ上のノード X と Y があったときに次のように定義されます。X から出口ノードまで全ての有効パスが Y を含み、かつ X ≠ Y から Y ≠ 出口ノードの時、X は Y にポストドミネートされるといいます。

[2] コントロールフローグラフ上のノード X と Y があったときに次のように定義されます。X から Y へのパス上の全てのノード（但し X と Y を含まない）が Y にポストドミネートされる。X は Y にポストドミネートされないとき、Y は X にコントロール依存するといいます。

4.2　中粒度タスクの並列処理

Do ループイタレーションレベルの並列処理に関する研究では、イリノイ大 CEDAR プロジェクトが進んでおり、複雑な配列添字に対するデータ解析、マルチプロセッサシステム用 Do ループリストラクチャリング、多重ループの階層型マルチプロセッサへのスケジューリングなどの研究が広範に行われています。特に CEDAR では、各々が演算パイプラインを持つ 8 台のプロセッサからなるプロセッサクラスタを 4 クラスタ接続しているので、プロセッサ内のパイプライン、クラスタ内の Doall 及び Doacross、クラスタ間の Doall の 3 階層の並列性を使用できます。更にメモリも、全てのプロセッサにより共有されるグローバルメモリとプロセッサクラスタ内のプロセッサのみに共有されるクラスタメモリ、クラスタ内の共有キャッシュ、プロセッサ内部のベクトルレジスタと複数の階層があるため、これらの階層を十分に使いこなすためのプログラムリストラクチャリング及びメモリ制御に関する研究にも力が入れられています。特に CEDAR コンパイラでよく用いられるリストラクチャリング手法は、ストップマイニングであり、１重ループに何回もストップマイニングを適用することにより多重ループを生成し、３階層の並列処理を行おうとしています。但しこのような階層的な並列処理により希望通りの性能が得られるかどうかは、スケジューリング法及びデータの階層メモリへの分割・配置法に深く関係しており、今後これらの分野でのいっそうの研究努力が望まれています。

4.3　粗粒度タスクの並列処理

粗粒度タスク間並列処理に於いては、粗粒度タスク間に条件分岐が存在するため、全てのタスクが実行されるわけではなく、どの粗粒度タスクを実行すべきかが条件分岐によって決定されるという性質があります。この為並列性検出手法や並列実行方式は、全てのタスク集合を対象としたものと異なったものとなります。以下に粗粒度タスクの並列性検出のための技術を解説します。

4.3.1　先行制約による並列性の表現

タスク間に条件分岐が無く、全てのタスクが実行されるようなタスク集合に対するタスク並列性の検出は、タスク間のデータ依存関係に起因するタスク間の先行制約のみを求めることによって達成されます。この先行制約はタスク間の実行順序の制約という本来の意味の他に、各タスクが実行可能となる時点をも表現しています。すなわち、タスク Ts に直接先行タスク群 Tp が存在する場合、全ての Tp が終了した時点で Ts は実行可能となる、ということを示しています。

一方、どのタスクが実行されるかが条件分岐で実行時に決定されるタスク集合に対する並列性の検出は、データ依存に加えて制御依存に起因するタスク間先行制約を求めることによって達成されます。制御依存に起因する先行制約とは、例えば、Task A を実行することが Task B の分岐条件で決定される場合、Task A は Task B に制御依存しているといい、Task B より先に Task A を実行することはできないとするものです。しかし、この 2 つの依存関係に基づく先行制約は、各タスクの実行開始可能となる時点は表現できません。

4.3.2　実行開始条件による並列性の表現

これに対し、粗粒度タスク間の並列性を実行開始条件として表現する並列性検出手法が提案されています。

粗粒度タスクの実行条件とは、プログラム実行中にどのような条件がそろったらその粗粒度タスクの実行を開始してよいか、ということを論理式で表現したものです。論理式は、粗粒度タスク MTi の実行が終了したときに真となる変数 (MTi)、条件分岐の評価によって MTi から MTj への分岐が決定したときに真となる (i,j)、MTi 中の条件分岐の評価によって MTj 分岐が決定し、かつ MTi の実行が終了したときに真となる i(i,j)、の 3 種の条件と、論理積・論理和で構成されます。

粗粒度タスクの実行開始条件が生成されれば、粗粒度タスクの並列実行は、各条件の初期値を偽とし、スケジューラが粗粒度タスクの実行情報を元に各粗粒度タスクの実行条件を評価し、実行開始条件が成立したもの（論理式が真になったもの）をスケジューリング戦略に従って順次プロセッサへ割り当てていく、という方法で行うことができます。

4.4　分散メモリマルチプロセッサの為の並列化技術

分散メモリ型のマルチプロセッサシステムのための自動並列化コンパイラの研究がなされている理由としては、次の理由が挙げられます。

- 一般的な分散メモリマシンでは、分散メモリへのデータの分割配置、及びプロセッサ間でのデータ通信を明示的に指示しなければなりません。従来このような作業を高いレベルでサポートするツールの整備が

遅れていたため、分散メモリシステム上でのプログラミングは、低レベルでのプログラミングとなってしまい、複雑でエラーを犯しやすいものとなっていました。その上、そのように開発されたコードは、対象としたマシン専用のものとなってしまっていました。

- 分散メモリ型マシンに於いては、データの分割配置によって処理の割り当てやプロセッサ間通信が決定されます。特に、共有メモリ型マシンに比べプロセッサ間通信のためのコストが高く、データの分割配置に伴う通信オーバヘッドが大きいため、データの分割配置方法の良否が分散メモリ型マシン上での並列処理の効果を決定します。しかしながら、ユーザがその最適化をプログラム全域にわたって行うのは困難である。

このような理由で、逐次プログラムまたは共有メモリマシン用のプログラムを、分散メモリ型マシン用にコンパイルする手法についての研究が活発化してきましたが、それらのどれもが、分散メモリ上での owner computes rule[3] と SPMD(single program,multiple-data) モデル[4]に基づいて doall ループを並列処理しようというものです。

上記研究プロジェクトでのコンパイラでは、グローバルメモリ空間を使用したプログラムに対し、ユーザがグローバルデータの分割を指示し、コンパイラがその指示をもとに、分割されたデータの分散メモリへの配置の決定、グローバルデータへのアクセスをローカル及びノンローカルアクセスへの変換、ノンローカルデータアクセスに対してのメッセージパッシングコードの挿入、を行い、SPMD コードを出力するというのが一般的です。

4.4.1 通信コードの最適化

通信処理のためのコードによるオーバヘッドを減らす為には、できるだけデータの send/receive をまとめて行うことが有効です。

通信ベクトル化 (message vectorization) と呼ばれる手法は、ループの各イタレーション毎に必要となる send/receive を、ループの前にまとめて行うものです。またデータ参照パターンに応じて適当な通信の方式（一対一、一対多、多対多など）を選択することによりオーバヘッドを軽減することができます。これらの方法を実現するコードを生成するためにはデータアクセスパターンがコンパイル時に決定されていなければなりませんが、更にこれを実行時にデータアクセスパターンが決定されるもの (irregiular problems) に対しても適用できるようにするものとして Inspector/Executor 方式が提案されています。この方式では、一つのループに対しデータのアクセスパターンを求める Inspector と、実際に通信・計算を行う Executor を生成します。Inspector では send/receive すべきデータを決定し、通信が必要なループイタレーションと不必要なループイタレーションを決定します。Executor では Inspector で生成された情報をもとに、

(1) 他プロセッサへのデータの send

(2) 通信が不必要なイタレーションの実行

(3) データの receive

(4) receive したデータを用いイタレーションの実行

を行います。Inspector では通信すべきデータ決定のための情報のやりとりが余計に必要となってしまいますが、それでも処理速度が向上することが知られています。また、Inspector の処理を高速化する手法も提案されています。

4.4.2 データ分割配置の最適化

データの分割配置（分配）はプロセッサ間の通信回数と負荷分散に大きな影響を与えます。データの最適分割配置を見つける問題は NP 完全であることが知られており、実際の手法はヒューリスティックなものとなります。この最適分割配置の問題は次の 2 つの問題に要約できます。

- プログラムのある部分のデータ最適配置が他の部分の最適配置と競合する場合どのように調整するのか。
- 最適化に際して何をもって最適とするのか。

データ分割配置の問題を「プログラマが行うべき知的作業の1つ」と見るむきもありますが、一方これを自動化しようとする試みもなされています。例えば、データ参照関係がコンパイル時に静的に解析可能なプログラムに対し、プログラム中の各ループで望まれるデータ分割配置に関する制約と、それが満たされた（或いは満たされない）場合に実行時間に与える影響を求めておき、プログラム全体で実行時間を最小化するような制約間の調

---

[3] 配列データを各プロセッサに分割配置した場合に、自分のローカルメモリにデータが配置されているプロセッサがそのデータへ代入を行う代入文の右辺の計算と代入を行うとする規則。

[4] 処理の分割方式の１つで、各プロセッサは同じプログラムを実行するがプロセッサ ID などに基づき異なるコード（異なるイタレーションや異なるプログラムセグメントなど）を実行するモデル。

整を行うことにより、データ分割配置を自動的に決定するという手法が提案されています。

現在のところ、自動データ分配手法の効果が期待できるものは、コンパイル時に静的なデータ依存パターンを解析できるようなプログラムに限られます。しかし、科学技術計算に於けるプログラムに関しては、このような性質を持つものが多いので、十分効果が期待できるものと考えられています。

5　具体的なコンパイラの紹介など

以下に具体的なコンパイラについて、上に述べたことがどのように実現されているか紹介します。

5.1　並列化コンパイラ TINPAR

TINPAR は、富田研究室が開発している、メッセージ交換型並列計算機のための並列化コンパイラです。TINPAR は owners computes rule に従って、拡張 Tiny Language で書かれた逐次プログラムから C 言語による SPMD プログラムを生成します。TINPAR による並列化は次の 3 つのステップから構成されています。

1. 前処理 :並列化に適した形にプログラムを変換します。逐次プログラムには、そのままでは並列化に適さない部分があります。そこで、ここでは、総和演算のイディオム変換、リモートデータに対する一時的なローカルコピーの作成 ·利用などを行ってプログラムの変換を行います。
2. 並列化 :owner computes rule に従って単純な並列化を行います。
    (a) if 文の挿入
    分割されたデータに対する代入は当該データの所有者プロセッサのみが行います。そこでこのようなデータに対する代入文には、所有者プロセッサを判定するための if 及び endif 文を前後に挿入します。
    (b) send/receive 文の挿入
    分割されたデータに対する参照は、当該データの所有者プロセッサと参照者プロセッサの間のメッセージ交換で実現されます。そこでこのようなデータに対する参照には、所有者プロセッサによる送信のための send 文、参照者プロセッサによる受信のための receive 文、及び両者を判定するための if/endif 文を挿入します。
3. 最適化 :不要な通信コードの削除や通信コードの移動を行います。並列化部で用いられている並列化手法は極めて単純であり、コード的に無駄が多く効率も良くありません。そこで本モジュールでは、不要な if 文の削除や send 文のリスケジューリングなどの最適化を行います。

5.1.1　最適化手法について

TINPAR は C によるオブジェクトコードを生成します。この為対象並列計算機に依存した低レベルの最適化は C コンパイラにまかせることができます。そこで、TINPAR の最適化部では、並列化に密接した最適化のみを行います

```
for i = 0, n-1 do
        if OWNER(a(i)) == GET_CELL_ID() then
                a(i) = ...
        endif
endfor

                    ▽

for i ∈ {i|OWNER(a(i)) == GET_CELL_ID(), 0 ≦ i ≦ n-1} d
        if OWNER(a(i)) == GET_CELL_ID() then
                a(i) = ...
        endif
endfor
```

図 2: ループの実行範囲の縮小

- ループの実行範囲の縮小

並列化部の生成するコードには、図 2上のような for ループが多数現れます。このようなループでは、if 文の条件式が成立しない場合には無意味なイタレーションを実行することになります。

そこで図 2下のように、そのような条件式が成立する範囲にループの実行範囲を狭めることによって空回りとなるイタレーションを削除します。

- 恒真/恒偽となる if 文の削除

  並列化部の生成するコードに含まれる、分割されたデータの所有者プロセッサを判定するための if 文の条件式には、除算や剰余算といった重い演算が含まれます。一方、ループの実行範囲の縮小を行った結果、そのような条件式が恒真または恒偽となる場合があります。そこでここでは、コンパイル時に if 文の条件式を静的評価することによって、このような不必要な if 文を削除します。具体的には、if 文の条件式を成立させるループインデックス値の条件とループインデックスのとりうる値を比較し、条件式が恒真/恒偽であるかどうかを調べます。

- if 文を削除するためのループ分割

  恒真/恒偽となる if 文の削除は、図 3上のような場合には当然適用できません。しかしこのような場合にも、ループインデックスのとりうる範囲を分割することによって、if 文の条件式を恒真または恒偽とすることができます。具体的には、ループ内の各文が実行されるべきループインデックス値の条件を求め、実行されるべき文が共通となるループインデックス値の範囲毎に当該ループを分割します。

- 不要な send/receive 文の削除

  メッセージ交換型並列計算機では通信処理にかかるコストは一般に大きいため、不要な通信は可能な限り削除しなければなりません。そこでここでは、データフロー解析を行うことによって、不要な通信コードを削除します。具体的には、不要な代入操作となる receive 文をデータフロー解析によって検出し、対応する send または broadcast 文とともに削除します。

  通信レイテンシは、演算に必要なデータを他のプロセッサから受信するための待ちによって生じます。TINPAR は最適化手法として以下の事柄を採用しています。

- send 文のリスケジューリング

```
for i = 0, n - 1 do
        if OWNER(a(i)) == GET_CELLID() then
                a(i) = ...
        endif
        b(i) = ...
endfor
                         ▽
for i ∈ {i|OWNER(a(i)) == GET_CELL_ID(), 0 ≦ i ≦ n - 1} do
        a(i) = ...
        b(i) = ...
endfor
for i ∈ {i|OWNER(a(i)) != GET_CELL_ID(), 0 ≦ i ≦ n - 1} do
        b(i) = ...
endfor
```

図 3: if 文を削除するためのループ分割

  メッセージ交換型並列計算機では、送信側プロセッサから送られたデータは、実際に使用されるまで受信側でバッファリングされます。そこで、send 文をリスケジューリングすることによってデータ送信を可能な限り先行させます。これにより、通信レイテンシをある程度抑えることができます。

- send/receive 文をともに含むループのループ分配

  send 文のリスケジューリングは、制御及びデータ依存関係による制約を受けるため、通信レイテンシを完全に隠蔽するには十分ではありません。例えば、図 4上のような場合には、データの送信から受信までの時間間隔は非常に短く、通信レイテンシが生じると予想されます。しかしこのコードに図 4下のようなループ分配を行うと、送信から受信までの時間間隔は前半ループの実行時間程度となり、レイテンシはほとんど隠蔽されます。

- MSG_BUFFER_FLUSH 文の挿入

  オブジェクトコードにリンクされる専用通信ライブラリは動的ベクトル化機能を持ちます。しかしこのようなバッファリングを行う場合には、バッファを適切にフラッシュしなければその利点を十分に生かせません。そこで、連続する send 文の直後に MSG_BUFFER_FLUSH 文を挿入することによって、通信ランタイムシステムの送信バッファを適切にフラッシュさせます。

上記の最適化手法を施すことにより、TINPAR は 64 プロセッサの AP1000 を用いて、行列積で 42.3 倍、ガウス消去法で 29.6 倍、SOR 法で 27.6 倍の加速率を達成しました。

5.2 FORTRAN D

FORTRAN D はデータパラレルプログラムの 2 つの基本的な段階である problem mapping と machine mapping をサポートするように設計された FORTRAN の一種です。

データパラレルな問題の並列化には次に挙げる 2 つのレベルがあります。

- 計算に内在する構造により引き起こされる問題 (problem mapping)—細粒度の並列性に依存
- 現実の計算機の資源プロセッサ数などにより引き起こされる問題 (machine mapping)—粗粒度の並列性

```
for i = 0, n -1 do
        if OWNER(a(i)) == GET_CELL_ID() then
                SEND(OWNER(b(i)),a(i))
        endif
        if OWNER(b(i)) == GET_CELL_ID() then
                RECV(OWNER(a(i)),tmp_a)
                b(I) = tmp_a
        endif
endfor
```

▽

```
for i = 0, n -1 do
        if OWNER(a(i)) == GET_CELL_ID() then
                SEND(OWNER(b(i)),a(i))
        endif
endfor
for i = 0, n -1 do
        if OWNER(b(i)) == GET_CELL_ID() then
                RECV(OWNER(a(i)),tmp_a)
                b(i) = tmp_a
        endif
endfor
```

図 4: send/receive 文をともに含むループのループ分配

に依存

5.2.1 Problem Mapping

- DECMPOSITION 命令

  DECMPOSITION 命令は使用する decomposition の次元・大きさを宣言します。この命令は抽象的なアラインメントを示し、メモリなどは割り付けられません。例えば、

  DECOMPOSITION A(N)
  DECOMPOSITION A(N,N)

  とある時、前者はサイズ N の１次元の decomposition を宣言し、後者はサイズ NxN の 2 次元で宣言しています。

- ALIGN 命令

  ALIGN 命令は decomposition に配列を割り付ける命令です。decomposition への割り付けは、I,J,K など次元内の位置を示す添字を介して決定されます。

5.2.2 Machine Mapping

FORTRAN D では、decomposition を物理的な並列計算機にマッピングするために DISTRIBUTE 命令を使用します。うまく分配するためには、プロセッサの数、各々のプロセッサのメモリや通信コストを考慮に入れなければなりません。また、分配される配列の大きさや計算構造も重要になります。さらに、data parallelism には規則的なものと不規則的なものがあり、不規則な場合には実行時などに於いて配慮が必要です。

また、decomposition に対して計算機マッピングの手法を明記しコンパイラはそれを適用するため、各々の配列に対してマッピング手法を指定する必要はありません。

- DISTRIBUTE 命令

  decomposition のそれぞれの次元に対する属性を受け、それに従って分配を行います。

- サポートしない分配

  FORTRAN D による規則的分配は、コンパイラによる比較的簡単な生成を意図しているので、次のような簡単なルールに従っています。

  - 次元間の関係が直交している。
  - 境界を除いた区分けが規則的である。
  - 割り当てが規則的である。

5.2.3 その他の機能

- 動的なアラインメントと分配

  データマッピングは場合によって変わりうるので、動的なリアラインメントや再分配もサポートしています。

- 手続き

  分配された配列を引数として呼ぶことが許されています。また、DECOMPOSITION, ALIGN, DISTRIBUTE などの命令はスコープルールにより適用範囲が制限されています。

- FORALL ループ

  ループ命令に於いて、ループ運搬依存は並列化の妨げとなります。その改善策として FORALL 命令によるループを行います。

  FORALL 命令において、それぞれの反復はループ前か同一反復内で宣言された変数のみを使うことができます。FORALL ループの最後に、それぞれの反復で新しく代入された値のうち、反復順に見て最も新しい値が残されます。

- リダクション

  リダクションとは、数集合を計算し、結果的により少ない集合にする作業をいいます。FORTRAN D では REDUCE 命令で明示的にリダクションを提供し、コンパイラが並列化をし易くしています。

- ON 節

  FORTRAN D では ON 節によりループの反復をどのプロセッサが実行するかを指定することができます。

5.3 V-Pascal/DM

V-Pascal/DM は津田研究室で研究開発中の自動ベクトル化コンパイラ V-Pascal を拡張した自動並列化コンパイラです。

プログラマは、大規模配列データの分割の指示を含む逐次プログラムを記述します。コンパイラは、プログラマの指示に従いデータを分割し、各要素計算機に割り当てます。コンパイラは更に、プログラム中の並列実行可能なループを検出し、その繰り返し実行を並列に処理するようにプログラム変換を行います。

並列化されたループ中の演算は、owner computes rule に基づいて実行されます。更に、自動ベクトル化コンパ

イラで用いられている諸技法を応用することで、IF 文や一次巡回演算を含むループの自動並列化処理も実現されます。

演算に必要なデータが他の要素計算機により保持されるとき、その値を転送するようにコンパイラが通信命令を挿入します。通信のレイテンシとオーバヘッドを削減するため、通信処理はブロック化して行われます。並列計算機の通信能力がその演算能力に比べて十分でないとき、通信処理がボトルネックとなりプログラムの処理速度が向上しない事態が生じることになり得ます。V-Pascal/DM では、通信量を削減するために次のような手法を採用しています。

- プログラム中に複数の分割配列があり、ある定義のために他の配列が引用されるとき、配列の分割法と要素計算機への割り付けを工夫することで、通信量を削減することができます。
- 分割が等しく、異なる次元について等しい添字式が記述された二配列の参照が存在するとき、分割小配列のジオメトリ値を転置した値で要素計算機に割り付ければ通信量を減らせる場合があります。
- ループ中にループ不変項の引用やループ不変式が存在するとき、その処理をループ外で行い、一回の放送ですますことができます。
- 通信量の削減が困難な場合でも、要素計算機の結合トポロジに着目し、分割された配列の要素計算機への割り付けを工夫することで、計算機の通信能力を最大限に活用できる場合があります。

6　終わりに

以上自動並列化コンパイラの技術、及びその具体例について解説してきました。分散メモリマルチプロセッサシステムのための並列化技術や大域的並列性の検出は、超並列処理システムの実現に向けての重要な課題であり、今後の研究が期待されています。

参考文献

[1] 笠原博徳並列処理技術コロナ社

[2] 本多弘樹,　自動並列化コンパイラ　情報処理,Vol.34,No9,pp. 1150-1157,Sep 1993

[3] 三吉郁夫、前山浩二、後藤慎也、森眞一郎、中島浩、富田眞治 メッセージ交換型並列計算機のための並列化コンパイラ TINPAR –最適化手法と性能評価–

[4] 三吉郁夫、前山浩二、後藤慎也、森眞一郎、中島浩、富田眞治 並列化コンパイラ TINPAR における専用通信ライブラリの実装と性能評価

[5] Geoffrey Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, Min-You Wu　FORTRAN D Language Specification January 22,1992

[6] 後藤慎也論文紹介　FORTRAN D Language Specification June 9,1994

[7] 梅田憲 データパラレルアプローチに基づいた分散メモリ型並列計算機用自動並列化コンパイラ February,1994