メッセージ通信を最適化する配列アラインメント処理の並列化コンパイラへの実装

辰巳 尚吾

平成 8 年 2 月 15 日提出

内容梗概

次世代の高性能計算機として並列計算機に大きな期待が寄せられている。しかし、一般に並列計算機向きのプログラムの記述は困難であるため、逐次計算機向きに記述されたプログラムを自動的に並列化する、自動並列化コンパイラの実現が求められている。

自動並列化コンパイラの第一の目的は、与えられた逐次計算機向きのプログラムから、問題の並列性を最大限引き出すことである。しかし、メッセージ交換方式による分散メモリ型並列計算機においては、プロセッサ間のメッセージ交換のオーバヘッドが大きいため、問題の並列性の抽出だけでなく、プログラムやその処理の対象となるデータの最適な分割を決定することも重要である。ここで言うデータ分割の決定とは、配列変数の各要素をどのプロセッサに割り付けるかということを決定することである。

最適なデータ分割の決定は、並列実行時のメッセージ通信のオーバヘッドを予測しなくてはならないため、一般的には困難なことである。それにも関わらず、現在では、データの分割は、ユーザが指定しなければならないことが多い。

そこで本研究では、与えられた逐次計算機向きに記述されたプログラムからメッセージ通信量を最小にするようなデータ分割を自動的に求めることを目的とした。

一般的にデータ分割は、プログラムに現われるすべての配列変数に対するアラインメントと、それらをどのようにプロセッサにマッピングするかを指定すれば一意に決定することができる。

本研究で採用した手法では、アラインメントの決定を更に、配列変数の次元の対応づけ (以下、配列次元アライメントと呼ぶ) の決定と、ブロックサイズの決定との 2 つのフェーズに分ける。配列変数の次元の対応づけは、与えられた逐次プログラムの配列変数どうしの代入文を解析して、添字式間のアフィン関係を抽出し、CAG(Component Affinity Graph) を作成することにより、グラフの問題に帰着できることが知られている。本研究ではこの CAG の作成にあたり、配列変数の添字式間のアフィン関係の抽出を両辺に配列変数を含む代入文に限らず、スカラー変数を介して代入される配列変数どうしにまで拡張した。更に代入文の実行される回数を CAG のエッジの重みづけに反映させるようにした。

さらに、本研究で採用した配列次元アライメントの決定フェーズを並列化コンパイラ TINPAR へ組み込む形で実装した。TINPAR は我々が現在開発中のメッセージ交換方式の分散メモリ型並列計算機のための並列化コンパイラである。現在はデータ分割の指定を加えた逐次計算き向きのプログラムを owner computes rule に従って、並列化を行い、SPMD(Single Program Multiple Data stream) 形式のプログラムを生成する。本研究で実装したフェーズは、ユーザが指定していたデータ分割を自動的に決定する上で重要なステップである。

実装した手法の評価のために、実際のプログラムに本研究の分割手法を適用し、それと単純な分割方法と比較した。これらのプログラムを並列計算機 AP1000 で実行した結果、本研究の分割手法によるプログラムは、単純な分割によるものに比べて数百倍の性能であることが明らかになった。

また、数値演算ライブラリ BLAS3 の dgemm に対して、ソースプログラムの代入文解析部の拡張なしの手法を適用して求まる複数の配列次元アライメントと、本手法による拡張ありの手法で求まるただ 1 つの配列次元アライメントに対して、通信量を比較した。すると、本手法によって求まった配列次元アライメントの方が通信量が少ないことが判明した。このことから、代入文の解析部の拡張によって得られる情報が通信量をより少なくする配列次元アライメントを求めるのに有効であることが分かった。

An Implementation of Array Alignment for a Parallelizing Compiler to Minimize Interprocessor Communication

Shougo TATSUMI

Abstract

Parallel computers are regarded as the next generation supercomputers. However, parallel programming is generally difficult. For this reason, there is a requirement for parallelizing compilers which automatically generate the parallelized code for multiprocessors from the given sequential programs.

In parallel programming, it is important to extract the parallelism which exists in the problem so that many processors can run in parallel. Communication by the message-passing is much more expensive than computation on processors. Hence, it is also essential to determine an appropriate data distribution pattern so that a processor can compute as long as possible without any communication.

The determination of optimal data distribution pattern is generally difficult because it needs a very complicated analysis to predict the message-passing overhead. However, such a difficult determination is made by the user, or programmer, now a day.

To release the user from this hard work, we have implemented a system which automatically specifies the data distribution pattern with minimum number of communications from the given sequential programs.

In general, to specify the data distribution, we must specify the alignment of all arrays existing in the source program and the layout of those arrays in the target parallel machine.

The specification of array alignment consists of two phases: specification of the alignment of arrays dimensions and specification of block sizes. It is known that the problem of the specification of alignment of arrays dimensions can be replaced to the grouping problems of graph nodes by constructing a CAG(Component Affinity Graph) after the analysis of affinity relations derived from assignment statements containing arrays in both sides. In this paper, we extend the technique of making CAG in two ways. The first one is to extend the analysis into the assignment statements that not only containing arrays in both sides but also in either or none of them. And the second one is to consider the number of execution times of assignment statements in constructing the CAG.

We implemented this extended technique of constructing the CAG to specify the array alignment on TINPAR. TINPAR is a parallelizing compiler for message-passing multiprocessors which is under development. TINPAR parallelizes sequential programs written in an extended version of Tiny Language following the owner computes rule, then outputs a SPMD(Single Program Multiple Data stream) program.

To evaluate the effectiveness of extensions in making the CAG, we compared the execution times of the following two programs on Fujitsu AP1000: 1) a program with the distribution pattern which derived automatically from our system and 2) one with a simple distribution pattern. The performance of the former program was hundreds times faster than the latter one.

And we also estimated the amount of messages of two distribution patterns, one was automatically derived from our system and the other was derived using previously proposed technique. The amount of messages of the distribution pattern derived from our system was less than the other. According to these evaluations, we could conclude that our system with the extension in making the CAG effectively specifies the optimal array alignment.

1 はじめに

現代、スーパーコンピュータの主流となっているベクトル計算機は、演算が高速であるとはいえ、流体計算、天気予報、構造解析、分子工学などさまざまな分野における計算には、数日から数十日かかることも珍しくない。こういった計算を高速に実行する次世代の高性能計算機として、近年、並列計算機に大きな期待が寄せられている。並列計算機による処理を実用化するには、並列計算機用のアプリケーションの充実が必要不可欠である。そのため、並列計算機専用のアプリケーションの開発だけでなく、今までに蓄積された逐次プログラムを、並列計算機用に変換する並列化コンパイラの実現が望まれている。

並列計算機によって効率よく問題を解くには、問題の持つ並列性を最大限引きだすことや、プロセッサ間の負荷バランスをとることが重要である。しかしメッセージ交換型の分散メモリ並列計算機は、各プロセッサが固有のメモリを持ち、プロセッサ間のメッセージ通信を伴うデータアクセスに要する時間がプロセッサ内のローカルメモリへのデータアクセスに要する時間に比べて長いため、並列性の抽出やプロセッサ間の負荷バランスだけでなく、メッセージ通信量の低減を図ることも必要である。しかしプログラムを実行する際に、データ分割方法によって、処理をするプロセッサが必要なデータを所有しない場合がある。この場合、必要なデータを他のプロセッサからメッセージ通信によって参照しなければならない。従ってデータの分割は、その方法によって必要となるメッセージ通信量が異なるので、性能を大きく左右する。即ち、メッセージ通信量を最小にするという観点で最適なデータ分割を求めることが重要である。しかし、こういったデータ分割の決定は、並列に実行した場合のプロセッサ間でのデータの参照関係を追跡しなければならないため、必ずしも容易ではない。

これらのデータ分割決定の困難さを克服するためのアプローチとして、並列化されたプログラムの実行状態を推測するための情報を提示することによって、ユーザによるデータ分割決定を支援しようとするツールの開発もされている [11, 7]。しかしこのアプローチも、ツールの支援によって得られた実行状態に関する情報をユーザが理解し、適切なデータ分割を決定しなければならない。これらを自動化するシステムの実現が望まれている。

そこで本報告書では、最適なデータ分割を逐次プログラムから自動的に決定するための 1 つの段階である、配列アラインメントの決定手法に関して提案を行う。

既に Li [5] や Gupta[4] によって提案されている手法では、この配列アライメントの決定を配列変数の次元の対応づけとブロックサイズの決定の 2 つのステップに分ける。以下では、配列変数の次元の対応づけのことを配列次元アライメントと呼ぶことにする。配列次元アライメントは、逐次ソースプログラムの代入文の解析を行い、それに対応したグラフである CAG(Component Affinity Graph) を作成し、グラフの問題として解くことによって求まる。本研究で採用した配列次元アライメントの決定手法の概略はこれと同じである。しかし提案されている方法では、配列次元アライメントを決定する際に作成される CAG に通信量を反映できない。また、スカラ変数を介した配列変数の代入文に対する解析が行なわれていない。そこで、すでに提案されている手法から、代入文の解析部を拡張し、通信量とスカラ変数を介した代入文の解析結果を CAG に反映できるようにした。この拡張を加えた配列次元アライメント決定手法を、現在計算機工学講座において開発中の並列化コンパイラ TINPAR のデータ分割決定フェーズとして実装した。

まず、第 2章では、並列化コンパイラとデータ分割決定部の関係について述べる。次に第 3章で、配列次元アライメント決定手法について述べる。第 4章で実装した手法に対する評価をし、最後に第 5章でまとめる。

2 並列化コンパイラにおけるデータ分割手法

本章では、与えられた逐次プログラムから、対象並列計算機の実行可能オブジェクトが得られるまでの流れのなかでの、データ分割決定部と並列化コンパイラの関係について述べる。

2.1 並列化コンパイラとデータ分割決定部の関係

現在の並列化コンパイラでは、データ分割はユーザによって指定され、ディレクティブの形でコンパイラに指示される。本研究の目標である自動的なデータ分割決定部は、このユーザによる指定を代行することを目的とする。即ち、並列化される前の逐次ソースプログラムから、配列データがどのように相互参照されているのかを解析し、最適な分割を求める。その結果求まった分割を指定するディレクティブと逐次プログラムを並列化コンパイラに渡すのである (図 1)。ただし、ここでいう、ある配列データが他の配列データを参照する、とは、前者の配列データを持つプロセッサが、後者の配列データを持つプロセッサに対して参照要求を行うことを表す。この 2 つのデータが同じプロセッサにある場合は通信が必要ないが、別のプロセッサにある場合は通信が必要になる。

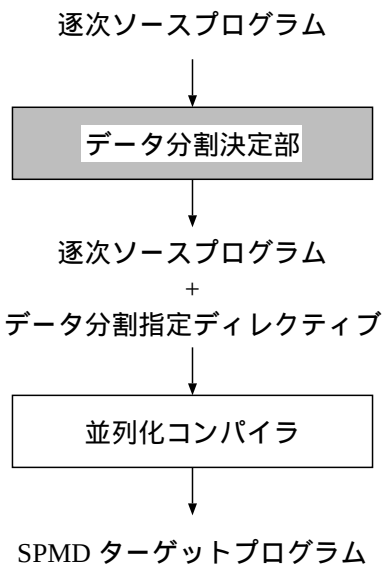


図 1: 並列化コンパイラとデータ分割決定の関係

2.2 最適データ分割決定問題

一般的にメッセージ交換型分散メモリ並列計算機を用いて問題を解く時は、データを各プロセッサが固有に持つメモリに分散して配置する。データ分割決定とは、この分散配置を決定することである。並列計算機では、あるプロセッサが演算に必要なデータを所有してなかった場合、これを所有する他のプロセッサに送信してもらわなければならない。従って通信量はデータ分割によって異ってくる。しかし通信のコストは、ローカルに持つメモリに対するアクセスのコストよりも大きいので、高速化のためには、通信量を最小化する必要がある。そこで、データ分割決定部はこのような最適なデータ分割を求めることを目的とする。

現在の分散メモリ型並列計算機のための並列化コンパイラのほとんどは数値処理プログラムを処理対象としている。数値処理プログラムはループ文のような規則的なデータアクセスパターンを持ち、処理内容に多くの並列性を持つ。このため、多くの並列化コンパイラは owner computes rule を並列化手法として用い、SPMD(Single Program Multiple Data stream) 型のコードを出力することによって比較的良い性能を引き出すことができる。owner computes rule とは、各プロセッサはそれぞれが所有する配列要素のデータを得るための演算を担当する、という規則である。これらのデータアクセスパターンの規則性のため、規則的な分割方法を適用することが有効となってくる。そこで、本研究ではこういった特徴をもつ数値処理演算を対象とする並列化コンパイラを対象とするので、規則的な分割のみ施すことを前提とする。

最適なデータ分割決定は、これらの規則性のため、2 つのステップに分けられる (図 2)。

1. 配列変数ごとのデータ分割の関係づけを行う。

2. 1 で求まった関係を保ちながら、各配列変数の規則的分割を行う。

既存の並列化コンパイラではすでに述べたように、これらはユーザが行なわなければならない。1 は、ALIGN ディレクティブによる配列変数のテンプレート配列へ対応づけ、2 における各配列変数の規則的分割は、DISTRIBUTE ディレクティブによるテンプレート配列の分割を指定によって行われる。これらのディレクティブによるデータ分割指定法を次節に述べる。

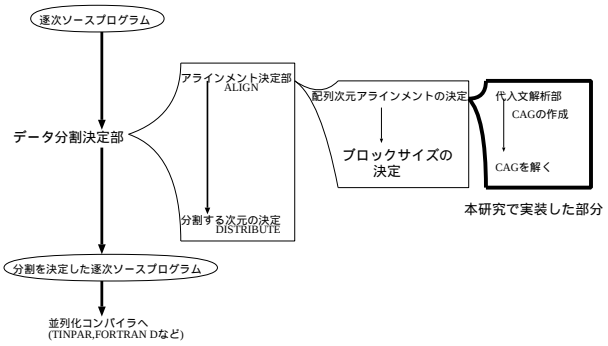


図 2: データ分割決定部の流れ

```
1:  REAL X(N,N)
2:  DECOMPOSITION A(N,N)
3:  ALIGN X(I,J) with A(I,J)
4:  ALIGN Y(I,J) with A(J,I)
5:  ALIGN Z(I) with A(I,0)
```

図 3: アライメントの例

2.3 ユーザによるデータ分割の指定方法

ユーザによる規則的な分割の指定方法の例として、Fortran D[2, 3] で採用されているディレクティブを使用した指定方法をについて述べる。

2.3.1 テンプレート配列の宣言 ~ DECOMPOSITION 文

DECOMPOSITION 文は、テンプレート配列の名前、次元数、サイズを宣言するものである。テンプレート配列とは、配列要素の対応づけ (alignment) を表現するための抽象的な配列であり、実際にはメモリは割り当てられない。

```
DECOMPOSITION A(N)
DECOMPOSITION B(N,N)
```

この例では、A は 1 次元のサイズ N のテンプレート配列を宣言している。また、B は N×N のテンプレート配列である。

2.3.2 アラインメントの指定 ~ ALIGN 文

すべての配列変数の各要素は、ALIGN 文によって、テンプレート配列の要素に割り当てられる。この割り当てを行うことをアラインメントと言う。Fortran D ではこれを ALIGN 文によって指定する。

図 3で指定されたアライメントでは、各配列変数要素がテンプレート配列に図 4のように配置される。X の 1 次元目と A の 1 次元目、X の 2 次元目と A の 2 次元目、Y の 1 次元目と A の 2 次元目、Y の 2 次元目と A の 1 次元目、Z の 1 次元目と A の 1 次元目がそれぞれ配列次元アライメントされることになる。このように、ALIGN 文の添字式に

テンプレート配列A

2次元目

1次元目

X(0,0) Y(0,0) Z(0)	X(0,1) Y(1,0)	X(0,2) Y(2,0)	X(0,3) Y(3,0)	X(0,4) Y(4,0)	X(0,5) Y(5,0)	X(0,6) Y(6,0)	X(0,7) Y(7,0)
X(1,0) Y(0,1) Z(1)	X(1,1) Y(1,1)	X(1,2) Y(2,1)	X(1,3) Y(3,1)	X(1,4) Y(4,1)	X(1,5) Y(5,1)	X(1,6) Y(6,1)	X(1,7) Y(7,1)
X(2,0) Y(0,2) Z(2)	X(2,1) Y(1,2)						X(2,7) Y(7,2)
X(3,0) Y(0,3) Z(3)	X(3,1) Y(1,3)						X(3,7) Y(7,3)
X(4,0) Y(0,4) Z(4)	X(4,1) Y(1,4)						X(4,7) Y(7,4)
X(5,0) Y(0,5) Z(5)	X(5,1) Y(1,5)						X(5,7) Y(7,5)
X(6,0) Y(0,6) Z(6)	X(6,1) Y(1,6)						X(6,7) Y(7,6)
X(7,0) Y(0,7) Z(7)	X(7,1) Y(1,7)	X(7,2) Y(2,7)	X(7,3) Y(3,7)	X(7,4) Y(4,7)	X(7,5) Y(5,7)	X(7,6) Y(6,7)	X(7,7) Y(7,7)

図 4: アライメントの例 (図解)

使用する添字変数の順番によって配列次元アライメントを定めることができる。

さらに、この添字式に、ある添字変数 1 つに関する 1 次式を使用することができ、このことによって、配列変数とテンプレート配列の対応づけを一定量ずらすようなオフセットや、一定間隔おきに配列要素を割り当てるストライドなどの指定が可能である。

2.3.3 プロセッサへの割り当て ~ DISTRIBUTE 文

DISTRIBUTE 文によって、テンプレート配列の各次元に一つの属性を与える。属性は、次元ごとに独立して設定することができる。与えられる属性は、BLOCK、CYCLIC、BLOCK_CYCLIC、* である。プロセッサの数を P、テンプレート配列 A の当該次元の要素数を Nとしたとき (ただし、$N \bmod P = 0$ とする)、それぞれの属性ごとに、次のような分割が行われる (図 5)。

- BLOCK— テンプレート配列 A の先頭から、$\frac{P}{N}$ 要素ずつを同じプロセッサに割りあてる (図 5(a))。($\frac{N}{P}$ を BLOCK 分割のブロックサイズと言う。)
- CYCLIC— テンプレート配列 A の添字番号を Pで割った余りが同じである要素を同じプロセッサに割りあてる (図 5(b))。
- BLOCK_CYCLIC(M)—この分割は、パラメータ M を持つ。テンプレート配列 A の添字番号を M で割り、余りが 0 の要素から M 個の要素を一つのグループとし、これを CYCLIC と同じ方法でプロセッサに割り当てていく (図 5(c))。(M を BLOCK_CYCLIC のブロックサイズと言う。)
- *— 当該次元は分割しない。

これらの属性の組み合わせにより、さまざまな分割方法が指定できる。

3 配列次元アラインメントの決定

配列次元アラインメントを決定するために、まず、逐次ソースプログラムに現われる配列変数どうしの代入文を解析する。次に、この解析情報から、CAG を作成する。本節では、ここで作成される CAG が何を表現したものであるか、メッセージ通信を最適にする配列次元アラインメントを求めることと、CAG の問題を解くこととはどのように対応しているのか、について述べる。

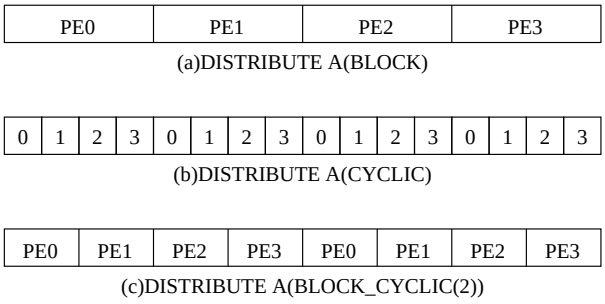


図 5: 1 次元の分割:テンプレート配列 A が 0 から 15 までの要素を持つ場合

3.1 配列次元アラインメントと CAG

ループ文内に、そのループ変数を添字式に含む配列変数どうしの代入文があった場合、owner computes rule に従い、左辺の配列要素を持つプロセッサが、右辺の配列要素を持つプロセッサからそのデータを送信してもらわなければならない。しかし、これらの 2 つの配列要素が同じプロセッサに割り当てられていれば、このデータ送信は必要でない。即ち、ループ変数の変化に伴い、どの配列要素どうしが同時に必要とされるのかを配列変数どうしの代入文から解析し、これらの配列要素を同じプロセッサに割り当てるような分割が最適な分割である。

これらの解析を行なう際に、配列変数の添字式間のアフィン関係というものに注目する。本報告書では、アフィン関数、アフィン関係という用語を以下の意味で用いる[1]。

- アフィン関数— $f(i) = a \times i + b$ の形であらわせる関数 f。ただし、a, b は整数、$a \neq 0$。i は、ループ変数。
- アフィン関係— 変数 (上の関数 f では i) が同

[1] 一般的なアフィン関数は、ベクトル変数から整数への写像として表され、アフィン関数もこのアフィン関数に対して定義される

じアフィン関数どうしを、互いにアフィン関係にあるという。

さて、CAG を作成するために、まず代入文の左辺に現われる配列変数と右辺に現われる配列変数のそれぞれの次元の添字式の組について、アフィン関係を調べる。例えば、図 6のようなプログラムの断片があったとする。

```
for j = 0 , n - 1 do
  for i = 0 ,n/2 - 1 do
    a(j,4*i) = b(3*i,j)
  endfor
endfor
```

図 6: 配列変数どうしの代入文

配列変数 a の 1 次元目と配列変数 b の 2 次元目、配列変数 a の 2 次元目と配列変数 b の 1 次元目がそれぞれアフィン関係にある。この場合、i,j の可能な組み合わせに対して、a(j,4*i) と b(3*i,j) ができる限り同じプロセッサに割り当てられるようにするには、

```
ALIGN a(I,J) with T(12*I,3*J)
ALIGN b(I,j) with T(4*J,12*I)
```

とすればよい。この場合、すべての i,j の組に対して a(j,4*i) と b(3*i,j) は同じテンプレート配列要素にアライメントできている。

この例から、添字式の間にアフィン関係があった場合、次のようなアライメントをすればいいことが分かる。

1. 添字式がアフィン関係にある次元は、テンプレート配列において、同じ次元に配列次元アライメントをする。
2. ストライドは、添字式のループ変数の係数の比によって定める。

これらは、2 次元の配列変数に限らず、何次元の配列変数に対してもいえる。更に、添字式に定数項がある場合も、アライメントのオフセットを定めることによって、同時に参照される配列要素をテンプレート配列の同じ要素にアライメントすることができる。この 1、2 は、それぞれ配列次元アライメントの決定、基本となるブロックサイズの決定に相当する。本報告書で取り扱う配列次元アライメントは、この 1 の部分を反映させようとするものである。

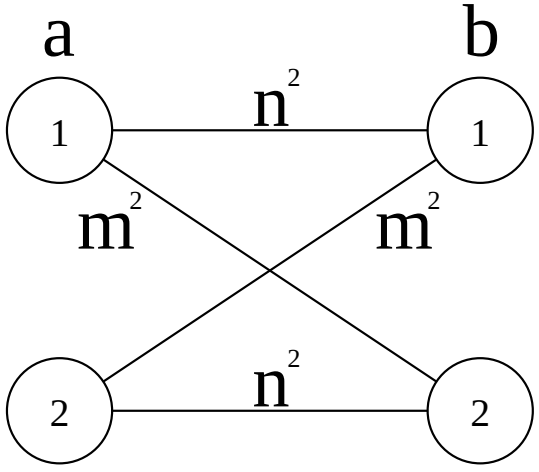
しかし、必ずしもこのように最適な配列次元アライメントが求まるわけではない。

```
for j = 0 , n - 1 do
  for i = 0 , n - 1 do
    a(i,i) = b(i,j)
  endfor
endfor
```

図 7: 同じ代入文での競合の例

図 7のプログラムの例では、b の 1 次元目は、a の 1 次元目 2 次元目の両方とアフィン関係にある。この場合、この代入文で同時に必要とされる配列要素すべてを同じテンプレート配列の同じ要素に割り当てる配列次元アライメントはない。また、図 8では、1 つ目の代入文と 2 つ目の代入文では、それぞれを局所的に見た場合の最適な配列次元アライメントは異る。これら場合、2 つの候補が競合しているが、最終的にはどちらかの候補を採用しなければならない。

このように、配列次元アライメントの候補が競合している場合、メッセージ通信量を最適化するという観点から、プログラム全体として最適な配列次元アライメントを求めなければならない。このため、配列変数どうしの代入文を解析した情報から CAG を作成し、競合候補から 1 つ候補を選



```
for j = 0 , n - 1 do
  for i = 0 , n - 1 do
    a(i,j) = b(i,j)
  endfor
endfor
      ⋮
for j = 0 , m - 1 do
  for i = 0 , m - 1 do
    a(i,j) = b(j,i)
  endfor
endfor
      ⋮
```

図 8: 異なる代入文間での競合の例

択する問題をグラフの問題に帰着するのである。

3.2 CAG の作成方法

CAG のノードは配列変数の次元に対応し、エッジはソースプログラムの代入文の解析の結果、異る配列変数の次元間のアフィン関係があった場合、それぞれの次元に対応するノードの間につくられる (図 3.1)。前節で述べたように、配列次元アライメントは、配列変数の次元のアフィン関係によって求めることができるので、このようにして作られる CAG は、配列次元アライメントを求めるのに必要な情報を表現したデータ構造になっている。

本節では、本研究において改良を加えた部分であるエッジの重みづけ方法と代入文の解析方法を含め、CAG の作成方法を説明する。

3.2.1 CAG のエッジの重みづけ方法

ループ文内にある代入文は、そのループが実行される回数によりそこから得られる添字式のアフィン関係の持つ重要性が異なる。図 8のプログラムでは、ループ文の実行範囲を指定する定数 m、n の大小によって、それぞれの中に含む代入文の実行回数が異なる。もし、$m \ll n$ であれば、当然最初のループ内の代入文の実行回数の方が多い。この場合、最初の代入文の解析から得られる配列次元アライメントを優先させるべきである。なぜならば、実行回数が多いということは、その代入文によって決められるアクセスパターンに従ったデータの移動が多いということになる。そのため、この解析から得られる配列次元アライメント以外の候補が採用された場合、メッセージ通信の量が多くなるからである。従ってよりよりアライメントを決定するためには、代入文の実行回数を考慮に入れなければならない。

そこで、本手法では、最適なアライメントを求める際に、代入文の実行回数までを考慮に入れられるように、それを CAG に反映させた。

具体的には、CAG を作成する際の配列変数の代入文から次元のアフィン関係を解析しエッジを作成する時に、その重みとしてその代入文の実行回数を付ける。ただし、コンパイル時の静的な解析のみでは実行回数が決まらない場合がある。この場合は、それ以上の解析は不可能であるので、便宜的に、1 回だけ実行されるものとして処理する。

3.2.2 解析対象とする代入文

配列変数のある次元方向の最大値を求めたり、配列変数ある次元方向の総和を求めたり (図 9) するようなリダクション演算では、配列要素の値を一旦スカラー変数へ代入したり、それよりも 1 つ次元数の少ない配列変数に代入したりする場合が多い。LU 分解のピボッティングルーチン (図 10) もその 1 つである。

```
for j = 0 , n-1 do
  for i = 0 , n-1 do
    s = s + a(i,j)
  endfor
  sum(j) = s
endfor
```

図 9: リダクション演算の例:総和演算

このようなリダクション演算においては、演算の結果を格納する配列変数のすべての次元 (図 9では sum の 1 次元目) と、その対象の配列変数の縮退しなかったすべての次元 (図 9では a の 2 次元目) との間で配列次元アライメントを求めることによって、メッセージ通信を最適にする配列次元アライメントを求めることができる。つまり図 9 においては、a(i,j) から sum(j) への代入がある

```
1 :for k = 0 , n-2 do
2 :  local_dmax(k) = abs(a(k,k))
3 :  for i = k + 1 , n - 1 do
4 :    candidate_a = abs(a(k,i))
5 :    dmax = local_dmax(k)
6 :    if (candidate_a > dmax) then
7 :      local_dmax(k) = candidate_a
8 :    endif
9 :  endfor
10:endfor
```

図 10: LU 分解におけるピボッティングルーチン

として解析すべきである。従って、sum の 1 次元目と a の 2 次元目を対応づけるのが最適な配列次元アライメントである。

これを確かめるために、図 9のプログラムのデータのアクセスパターンを追跡してみる (図 11)。まず、ある j に対して、内側の i ループでは、a の 1 次元目方向にデータが参照され、それらが足されていく。その結果が一旦スカラ変数 s に代入されたあと、sum(j) に代入される。結局 sum(j) の計算には、a の 2 次元目が j のデータだけが必要になる。従って、a の 2 次元目と sum の 1 次元目をアライメントし、この次元で分割することで通信なく演算を進めることができる。即ちこの配列次元アライメントは最適なものである。

しかし、配列変数どうしの代入文だけを解析の対象とした場合、図 9のプログラムからは何の情報も得られない。これらの情報を得るためには、リダクション演算に見られるような、配列変数の値を一旦スカラー変数に代入し、更にそのスカラー変数から別の配列変数への代入がある場合、前者の配列変数から後者の配列変数へ代入されているとして解析するのが適当である。これをスカラー変数を介した配列変数どうしの代入と呼ぶことにする。

このように、解析の対象として、スカラー変数を介した配列変数どうしの解析が有効である。そこで本手法では、この解析を CAG 作成の際に行ない、CAG に反映させた。具体的な解析方法は、3.2.3節のアルゴリズムによって説明する。

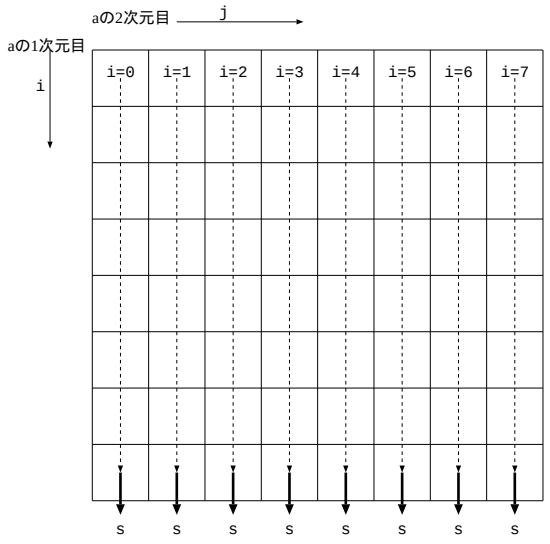


図 11: 総和演算のアクセスパターン (n=8)

3.2.3 CAG 作成方法の詳細

本節で述べた代入文の解析の拡張を含め、本研究で実装した、メッセージ通信を最適にするアラインメント決定のための CAG 作成アルゴリズムを以下に示す。

1. 入力として加えられる逐次ソースプログラムのすべての代入文のリストを作成する。
2. リストのノードを先頭から順に引数として、手続き Construct_CAG を呼び出す。

Construct_CAG は引数として代入文 (Assignment) をもつ。
手続き **Construct_CAG(Assignment)**

1. 代入文 Assignment の左辺が配列変数の場合、以下の処理を行う。
   - 左辺の配列変数に対してグラフのノードを作成する$_{(a)}$。
   - 右辺の式に現われる配列変数それぞれに対して、以下の処理を行う。
     - その配列変数に対してグラフのノードを作成する$_{(a)}$。
     - 左辺の配列変数に対するノードと右辺の配列変数に対するノードのすべ

ての組み合わせに対してそれらが互いにアフィン関係にあるかどうかを調べる$_{(b)}$。アフィン関係である場合は、その代入文が何回実行されるかを計算$_{(c)}$し、それを重みとしたエッジを与えられたノード間に作成する。

2. 代入文 Assignment の左辺がスカラー変数で、右辺に配列変数を含む場合、右辺の配列変数それぞれに対して以下の処理を行う。(ここで、右辺の配列変数を A(..)、左辺のスカラー変数を S とする。即ち、S=..A(..).. という代入文があったものとする。)

   - 代入文 Assignment 以降の代入文をたどり、スカラー変数 S を右辺に含む代入文 (V=..S.. とする) すべてに対して、V=A(..) という代入文を引数として、Construct_CAG を呼び出す。ただし、V=A(..) という代入文は、代入文 V=..S.. の直後にあるものとするが、実際には、リストには加えない。

(a) 与えられた配列変数に対するグラフのノードを作成する

- 既にノードが作られていた場合:現在調べている代入文の各次元の添字式を対応するグラフのノードに記録する。
- まだノードが作られてない場合:配列変数に対応するグラフのノードを作成する。現在調べている代入文の添字式をグラフのノードに記録する。

(b) アフィン関係にあるかどうか調べる
本手法で扱うアフィン関係は、3.1節で述べたように、$f(i) = a \times i + b$ の形をしたアフィン関数に対してのみである。即ち、添字式には、ループ変数が高々1 つしか現われないものとして判定することとなる。このことから、2 つの添字式を比べ、同じループ変数があるかどうかを見ることによって、アフィン関係を調べることができる。

(c) 代入文が何回実行されるかを計算する
実行回数が、実行時にしか決まらないものについては、解析できないので、便宜上実行回数は 1 回であるとする。また、条件分岐などのため、必ず実行されるとは限らない代入文については、必ず実行されるものとして計算する。

以上が CAG の作成法である。スカラー変数を介した配列変数どうしの代入の解析は、Construct_CAG の 2 の部分にあたる。図 12によって説明する。まず、代入文 S=..A(..).. について解析が行われる (図中 1)。次に、それ以降の代入文を調べ、S を代入文の右辺に含む代入文 V1=..S.. を発見し、これらから、V1=..A(..).. という代入があるものとして、以降の代入文の解析を行う (図中 2)。これを繰り返し、最終的に B(..)=..A(..).. という代入文があるものとして、CAG のノード及びエッジを作成する。このように、スカラー変数を介した配列変数どうしの代入の解析は、コントロールフローを無視した最も単純なデータフロー解析であるということもできる。

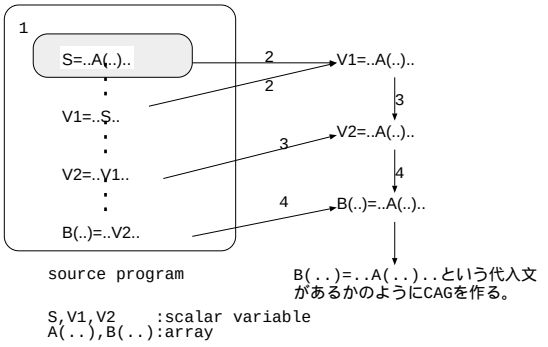


図 12: スカラー変数を介した配列変数どうしの代入

3.3 CAG による最適配列次元アライメント決定手法

ある配列次元アライメントを求めるということは、CAG のノードを次のようにグループ分けすることに相当する。

n をソースプログラムに現われる配列変数の最高次元数としたとき、すべてのノードを n 個の排他集合に分ける。ただし、それぞれの集合に属するある 1 つ配列変数の次元に対応するノードは、高々1 つである。

これらの可能なアライメントの中から、通信量の最も少なくなるようなものを求めることになる。エッジの重みは、その次元どうしがアライメントされた場合に得られる利益を表している。すなわち、エッジの重みは、アフィン関係の度合を示すものであり、それら 2 つの次元を同じグループに属させる度合を表す。このことから、CAG において上の方法に従ってあるグループ分けをしたとき、異なるグループ間に存在するエッジの重みは、その分だけ、メッセージ通信が必要になるということになる。逆に同じグループ間に存在するエッジの重みは、その重みの分だけ配列変数のローカリティを反映したアライメントになっている、ということになる。

従って、最適なアライメントを求めるということは、CAG において、グループ間を結ぶエッジの重みを最小とするようなグループ分けを求めるということに帰着されるのである。

これらの CAG とメッセージ通信量を最適にする配列アライメント決定の関係をまとめると、図 13のようになる。

CAG	配列次元アライメントの決定
ノード	ある配列変数のある次元
エッジの重み	エッジの両端のノードに対応する次元がアライメントされたときの配列変数のローカリティの反映度合
ノードのグループ分け	ある配列次元アライメントを求める
グループ間を結ぶエッジの重みを最小とするグループ分け	メッセージ通信を最適にするアライメント

図 13: CAG とアライメントとの対応

このようにして帰着されたグラフの問題は、NP 完全問題になることが知られている [5]。本研究では、全ての可能なグループ分けに対して探索を行っている。そのため、多数の配列変数が現われるプログラムに対しては、問題を解くのに時間を要する。文献 [5] では、これらの問題を解くヒューリスティックアルゴリズムが提案されている。本研究においては、この部分の改良は今後の課題である。

3.4 配列次元アライメント決定後のステップ

今回実装はしなかったが、アライメントの決定をするには、配列次元アライメントを決定したあと、各配列変数をどういったストライドでテンプレート配列に割り当てるのかを決定しなければならない。さらに最終的な分割を決定するには、分割する次元、分割の属性 (BLOCK,CYCLIC,BLOCK_CYCLIC) を決定する必要がある。

図 6で見たように、ある代入文にとって最適なアライメントとするためのストライドは、添字式のループ変数の係数の比によって定めればいい。次のような代入文

$\mathtt{a}(\mathtt{a}_0*\mathtt{i}_0,\mathtt{a}_1*\mathtt{i}_1,\cdots,\mathtt{a}_n*\mathtt{i}_n)=$
$\mathtt{b}(\mathtt{b}_0*\mathtt{i}_{k_0},\mathtt{b}_1*\mathtt{i}_{k_1},\cdots,\mathtt{b}_n*\mathtt{i}_{k_n})$

に対して、G を $\mathtt{a}_0,\cdots,\mathtt{b}_n$の最小公倍数とする。例えば $k_0 = t(0 \leq t \leq n)$ であった場合、a の t 番目と b の 0 番目の次元はアフィン関係にある。この場合、a の t 番目に対するテンプレート配列の次元のストライドは $G/\mathtt{b}_0$、b の 0 番目の次元に対応するストライドは $G/\mathtt{a}_0$ とすればいい。どの次元ともアフィン関係にない次元に対応するテンプレート配列の添字式の係数は G とすればいい。

次に、求まった配列アライメントから分割する次元を決定しなければならない。SPMD 型のコードを出力する並列化コンパイラでは、配列変数の複製を許さない場合、どの配列変数も必ず分割される。このためもしプログラム中に 1 次元配列変数があれば、すべての配列変数は、その次元に配列次元アライメントされた次元で分割されることになる。この場合は配列次元アライメントを決めると分割する次元まで求まる。配列変数の複製を許す場合、複製することによってデータの一貫性を保つための通信が必要になるかどうかの解析が

必要となる。これは、イタレーション間の依存解析によって行なうことができる。これはまた、アレイエクスパンションの可能性の解析ということもできる。この後に分割の属性を決定するのである。

3.5 関連研究との比較

Li[5] による CAG 作成法では、CAG のエッジの重みとして、1 と$\epsilon(0 < \epsilon \ll 1)$ のみを用いている。この重みづけ手法は次のとおりである。

- 代入文を解析し、1 つの代入文から生ずる競合があった場合はそれぞれの候補に対してエッジを作り、重みをϵとする。
- 1 つの代入文に対して、競合のないアライメントが求まるならばそのアライメントに対応するエッジの重みを 1 とする。

しかし、この CAG の作成方法では、代入文の実行回数は CAG には反映できていない。これでは、実行回数を考慮に入れた最適な配列次元アライメントを求めることはできない。

また、3.2.2節で述べたように、スカラ変数を介した代入の解析は、配列次元アライメントを求めるための有効な情報を得ることができるにも関わらず、解析対象とする代入文は、両辺に配列変数を含む代入文だけである。

4 並列化コンパイラ TINPAR への実装と性能評価

4.1 並列化コンパイラ TINPAR への実装

我々は現在並列化コンパイラ TINPAR の開発を行っている [12]。TINPAR は拡張 Tiny Language によって記述された逐次プログラムを並列化する。拡張 Tiny Language とは、ループ再構成ツール Tiny[8] のための言語 Tiny Language を拡張したプログラム記述言語である。

本研究では、これまでに述べてきた配列次元アライメントを求める手法を並列化コンパイラ TINPAR に実装した。本手法を実装するにあたって重要となる、TINPAR の特徴について以下に述べる。

Fortran D では、テンプレート配列の各次元に与える属性として、BLOCK、CYCLIC、BLOCK_CYCLIC があった。しかし、BLOCK、CYCLIC はそれぞれ BLOCK _CYCLIC のブロックサイズを N/P、1 にしたものである (Pはプロセッサ数、Nは当該次元の大きさ)。そこで、TINPAR では分割属性としてブロックサイズのみを指定する。また、Fortran D では、配列変数を一旦テンプレート配列に ALIGN 文によってアライメントし、テンプレート配列に分割属性を与える。しかし TINPAR では、各配列変数をテンプレート配列にアライメントすることなく直接に指定する。この指定方法では、Fortran D によってアライメントの段階で指定できるオフセット、複製などの指定はできない。

このように、TINPAR の分割指定ディレクティブの機能が限定的であるため、その他のより一般的な並列化コンパイラと違い、以下のような実装上の制限が生ずる。

1. 配列変数の次元の添字式が、定数項が 0 でないアフィン関数で表される場合、2 つの配列を一定量だけずらすようなオフセットつきアライメントが求まるが、それを TINPAR に指定することができない。しかし、配列次元アライメントを求めるステップまでにはこれらの制限は影響を与えない。

2. TINPAR では配列変数の複製を許してないためソースプログラムに 1 次元配列変数があった場合、その配列変数の次元は必ず分割される。このため、アライメントを求めただけで、分割する次元は自動的に決まる。後は、分割のブロックサイズを決めるだけでよい。そこで実装の際には、配列変数の複製を許さないということを前提とした。配列次元アライメントの決定にはこれらの条件は影響しない。

4.2 性能評価

本章では、配列次元アライメントの決定手法の有効性と代入文解析部の拡張の有効性を示す。

まず、配列次元アライメント決定手法の有効性を示すために、本手法を適用して得られた分割と単純な分割の並列計算機 AP1000 での実行時間を比較する。

次に、本手法で拡張した代入文の解析部を適用して得られる配列次元アライメントと、拡張がないことによる情報不足から生ずる複数の配列次元アライメントとの理論的通信量を比べることによって、代入文解析部の拡張の有効性を示す。

4.2.1 配列次元アライメント決定手法の有効性

配列次元アライメント決定手法の評価を行なうために、livermore kernel No.8 のプログラム (図 14) に対して処理を行った。このプログラムの場合、スカラー変数と配列変数の代入文がないため、スカラー変数を介した代入文の解析は出力する CAG に何の影響も与えない。

```
1 :const n = 512
2 :for kx = 1, n-2 do
3 :  for ky = 1, n - 2 do
4 :    du1(ky)=
5 :      u1(kx,ky+1,nl1)-u1(kx,ky-1,nl1)
6 :    du2(ky)=
7 :      u2(kx,ky+1,nl1)-u2(kx,ky-1,nl1)
8 :    du3(ky)=
9 :      u3(kx,ky+1,nl1)-u3(kx,ky-1,nl1)
10:    u1(kx,ky,nl2)=
11:      u1(kx,ky,nl1)+a11*du1(ky)
12:      +a12*du2(ky)+a13*du3(ky)
13:      +sig*(u1(kx+1,ky,nl1)
14:      -2.0*u1(kx,ky,nl1)+u1(kx-1,ky,nl1))
15:    u2(kx,ky,nl2)=
16:      u2(kx,ky,nl1)+a21*du1(ky)
17:      +a22*du2(ky)+a23*du3(ky)
18:      +sig*(u2(kx+1,ky,nl1)
19:      -2.0*u2(kx,ky,nl1)+u2(kx-1,ky,nl1))
20:    u3(kx, ky, nl2) =
21:      u3(kx,ky,nl1)+a31*du1(ky)
22:      +a32*du2(ky)+a33*du3(ky)
23:      +sig*(u3(kx+1,ky,nl1)
24:      -2.0*u3(kx,ky,nl1)+u3(kx-1,ky,nl1))
25:  endfor
26:endfor
```

図 14: livermore kernel No.8 のリスト

このリストに対して、本研究で実装したアライメント決定部を適用すると、図 15のような CAG が求まり、更に、図 16のようなアライメントが求まる。このアライメントをアライメント 1 とする。

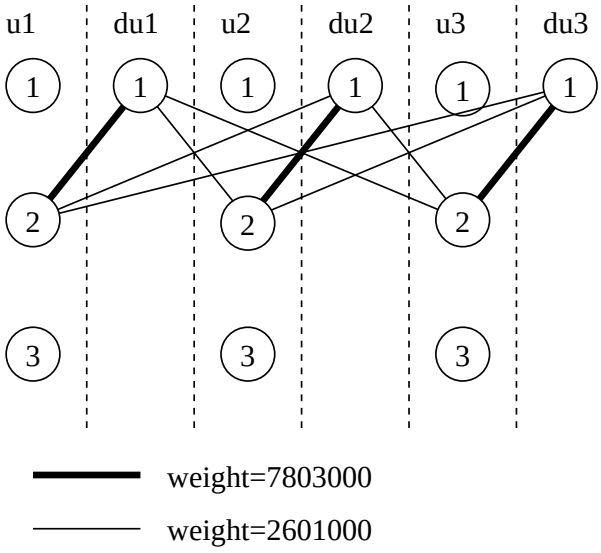


図 15: kernel No.8 の CAG

次に、比較のために、これとは別のアライメントとして、図 17 のような単純なアライメントを考える。これをアライメント 2 とする。

また、アライメント 1 は、u1,u2,u3 の 2 次元目、du1,du2,du3 の 1 次元目をそれぞれブロック分割した。アライメント 2 は、u1,u2,u3 の 1 次元目、du1,du2,du3 の 1 次元目をそれぞれブロック分割した。

以上の分割を Fortran D の記法に従って表記するとそれぞれ図 18、図 19 のようになる。

これをそれぞれ TINPAR によって並列化し、並列計算機 AP1000 で実行した。AP1000 は分散メモリ並列計算機である。本実験では 64 台のプロセッサが接続されたシステムを利用しプロセッサ数 1、2、4、8、16、32、64 台のそれぞれの実行時間を計測した。

図 20にそれぞれの台数と実行時間の関係のグラフを示す。図より、本手法によって得られたアライメント 1 をもとにした分割方法が明らかに性能がよいことが分かる。実際、64 プロセッサで比べると、アライメント 1 による分割は、アライメン

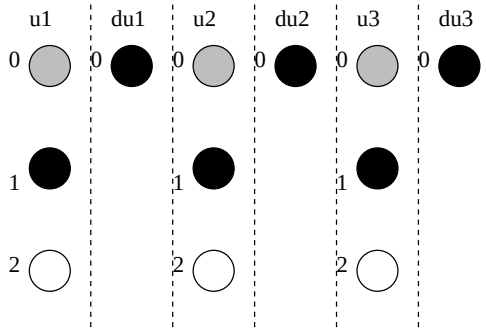


図 16: アライメント 1

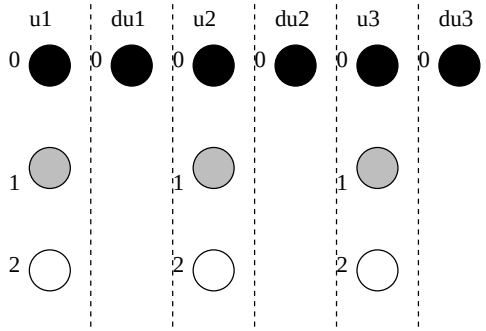


図 17: アライメント 2

```
DECOMPOSITION T(n-1,n-1,1)
ALIGN u1(I,J,K) with T(I,J,K)
ALIGN u2(I,J,K) with T(I,J,K)
ALIGN u3(I,J,K) with T(I,J,K)
ALIGN du1(J) with T(0,J,0)
ALIGN du2(J) with T(0,J,0)
ALIGN du3(J) with T(0,J,0)
DISTRIBUTE T(*,BLOCK,*)
```

図 18: アライメント 1 をもとにした分割

```
DECOMPOSITION T(n-1,n-1,1)
ALIGN u1(I,J,K) with T(I,J,K)
ALIGN u2(I,J,K) with T(I,J,K)
ALIGN u3(I,J,K) with T(I,J,K)
ALIGN du1(J) with T(J,0,0)
ALIGN du2(J) with T(J,0,0)
ALIGN du3(J) with T(J,0,0)
DISTRIBUTE T(BLOCK,*,*)
```

図 19: アライメント 2 をもとにした分割

ト 2 の分割の実に 500 倍程度の性能差がある。

これらの原因を考察する。まず図 14の 4 行目が実行される様子を見る。アライメント 1 から求まる分割方法では、$1 \leq \mathtt{ky} \bmod P \leq 510$($P$ はプロセッサ数) の場合 `du1(ky)` と `u1(kx,ky+1,nl1)` は同じプロセッサにあるため、通信が全く必要でない。通信が必要となるのは、$\mathtt{ky} \bmod P = 0$ または $\mathtt{ky} \bmod P = 511$ の時だけである。一方アライメント 2 から求まる分割方法では、ある `kx` に対して、`u1(kx,ky,nl1)` のうち、$512/P$個の `ky` に対応するデータは、`du1(ky)` の計算の際に通信なく得られるが、残りの $512(P-1)/P$個の `ky` に関しては `u1(kx,ky,nl)` のデータは通信によって得なければならない。これがすべての `kx` に対して毎回行なわれるので、通信量がアライメント 1 に比べはるかに多くなる。これは 5、6 行目に対してもいえる。

同様に、7-9 行目の代入文が実行される様子を見る。アライメント 1 から求まる分割方法では、全く通信なく演算を進めることができる。一方、アライメント 2 から求まる分割方法では、`du1,du2,du3` と `u1` とは、ほとんどの `kx,ky` の組に対して同じプロセッサにはないので、通信が必要となる。

これらの追跡から、アフィン関係にない次元どうしをアライメントしたアライメント 2 は、通信が頻発して性能が悪くなったと考えられる。このように、アフィン関係を調べることによって、アライメントを求め、それをもとに決定した分割の方が性能がいいことが確かめられた。

4.2.2 代入文解析部の拡張の有効性

図 21に BLAS 3[9] の dgemm のリストを示す。これは、一般行列 A, B, C と実数α, βに対して $C \leftarrow \alpha AB + \beta C$を計算する数値演算ライブラリである。これに対し本手法を適用すると、図 22のような CAG が求まり、これから、図 23のような配列次元アライメントが一意に定まる。

一方従来のスカラ変数を介した配列変数どうしの代入の解析を行なわない手法では、図 22において、c の 2 次元目と b の 2 次元目の間にあるエッ

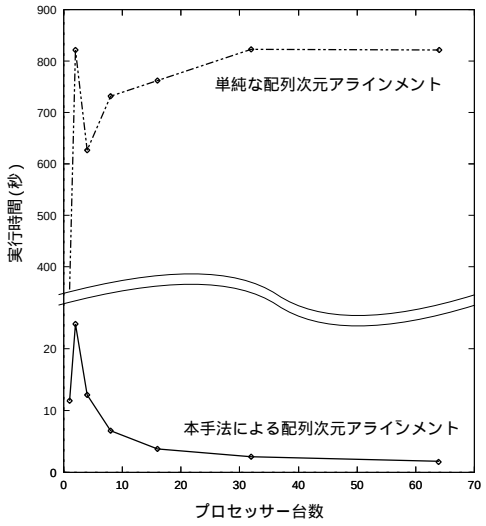


図 20: livermore kernel No.8 の実行時間

ジは作成されない。このため、配列次元アライメントは一意に定まらない。候補として、b の 1 次元目を c の 2 次元目にアライメントするものと、c の 2 次元目にアライメントするものの 2 つがある。前者は本手法によって一意に定まる配列次元アライメントと同じであるが、後者の配列次元アライメントが採用される場合もあり得る。

そこで、本手法で一意に定まる配列次元アライメント (図 23、アライメント 3 とよぶ。図 24) と、従来の手法で採用される可能性のある配列次元アライメント (アライメント 4 とよぶ。25) とで、通信量を見積もり、それらを比べることによって本手法による代入文解析部の拡張の有効性を示す。

通信量の見積もりは、テンプレート配列 T の各要素に一つの論理的なプロセッサを割りあてたものとして行う。すると、図 21の 8 行目と 13 行目は、どちらのアライメントの場合も通信なく行なえる。22 行目に関しては、アライメント 3 では $2N^3 - 2N^2$、アライメント 4 では、$2N^3 - N^2 - N$ の通信量が必要である。本手法によって定まる配列次元アライメントの方が $N^2 - N$だけ通信量が少ない。このことから、代入文解析部の拡張によって得られる情報が通信量をより少なくするものであることが分かる。

```
1 :const M=N,K=N
2 :real a(0:M-1,0:K-1),b(0:K-1,0:N-1)
3 :real c(0:M-1,0:N-1)
4 :real alpha,beta,b_temp,temp
5 :for j=0,N-1 do
6 :  if(beta == 0) then
7 :    for i=0,M-1 do
8 :      c(i,j)=0
9 :    endfor
10:  else
11:    if (beta != 1) then
12:      for i=0,M-1 do
13         c(i,j)=beta*c(i,j)
14:      endfor
15:    endif
16:  endif
17:  for l=0,K-1 do
18:    b_temp=b(l,j)
19:    if(b_temp!=0) then
20:      temp = alpha*b_temp
21:      for i=0,M-1 do
22:        c(i,j)=c(i,j)+temp*a(i,l)
23:      endfor
24:    endif
25:  endfor
26:endfor
```

図 21: BLAS3 dgemm

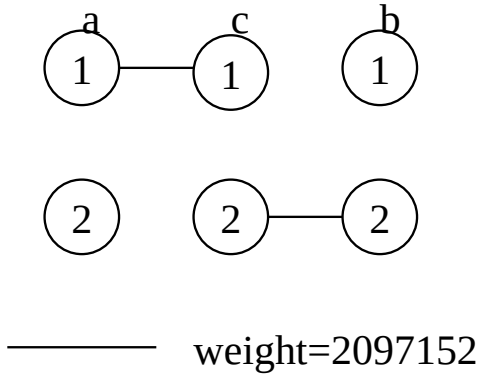


図 22: BLAS3 dgemm に対する CAG

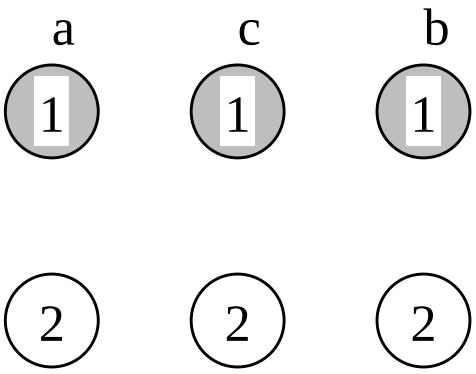


図 23: BLAS3 dgemm に対する配列次元アライメント (本手法適用)

```
DECOMPOSITION T(N,N)
ALIGN a(I,J) with T(I,J)
ALIGN b(I,J) with T(I,J)
ALIGN c(I,J) with T(I,J)
```

図 24: アライメント 3

```
DECOMPOSITION T(N,N)
ALIGN a(I,J) with T(I,J)
ALIGN b(I,J) with T(J,I)
ALIGN c(I,J) with T(I,J)
```

図 25: アライメント 4

5 おわりに

メッセージ通信を最適にする配列アライメントを求めるために、ソースプログラム中に現われる代入文を解析し、CAG を作成する手法が既に提案されている。本研究では、この手法の代入文の解析部において、代入文が実行される回数と、スカラー変数を介した配列変数どうしの代入の解析結果を従来の手法に加えて、CAG に反映させることを提案した。更に、それをメッセージ交換による分散メモリ型並列計算機用に開発中である並列化コンパイラ TINPAR に実装した。

livermore kernel No.8 に対して本手法を適用して得られる配列次元アライメントから求まる分割と、それとは別の単純な分割とをディレクティブによって指定し、並列化コンパイラ TINAPR によって並列化し、分散メモリ並列計算機 AP1000 での実行時間を比較した。その結果、本手法を適用して得られた配列次元アライメントから求まる分割を施した方が数百倍の性能であることが明かになった。このことから、アフィン関係を調べ、配列次元アライメントを決定する本手法の有効性が確認できた。

また、総和演算や最大値を求める演算などのリダクション演算見られるスカラ変数を介した配列変数どうしの代入があるプログラムに対して、従来の手法を適用した場合、配列変数どうしの代入文がプログラムに現われないため、配列次元アライメントに関する情報は何も得られない。ところが、拡張された代入文の解析を適用すると、通信が必要でない最適な配列次元アライメントが求まった。実際、数値演算ライブラリ BLAS3 の dgemm に対して本手法を適用した場合、本手法による拡張なしの解析から得られる配列次元アライメントの候補に比べ、本手法による拡張ありの解析で決まる配列次元アライメントの方が通信量が少ないことが確かめられた。このことから、代入文解析部の拡張によって得られる情報が通信量をより少なくするものであることが分かった。

今後は、CAG からノードのグルーピング問題を解く部分にヒューリスティックアルゴリズムを採用し、コンパイル時間を短縮する必要がある。その後、本研究で求められた配列次元アライメントを入力としたブロックサイズの決定と実プロセッ

サ上でのデータ分割フェーズの設計、実装を行う。得られたデータ分割を既存の TINPAR と結合することで、逐次プログラムから SPMD コードを得るという自動並列化が達成されることになる。

謝辞

最後に、本研究の機会を与えて下さり、適切な御指導を賜わった富田眞治教授、貴重なご助言を頂いた中島浩助教授、森眞一郎助教授に深く感謝致します。本研究に対して日頃から御討論頂いた窪田昌史氏をはじめとする京都大学工学部情報工学科富田研究室の諸氏に感謝致します。また、並列計算機 AP1000 の実行環境を提供して頂いた (株) 富士通研究所に感謝致します。

参考文献

[1] Anant Agarwal, David A.Kranz and Venkat Natarajan: Automatic Partitioning of Parallel Loops and Data Arrays for Distributed Shared-Memory Multiprocessors., *IEEE transactions on parallel and distributed systems*, Vol. 6, No. 9, p. 943 (1995).

[2] Geoffrey Fox, Seema Hiranandai, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, Min-You Wu: Fortran D Language Specification., Technical Report, Dept. of Computer Science, Rice University(January 1992).

[3] Seema Hiranandai, Ken Kennedy, and Chau-Wen Tseng: Evaluating Compiler Optimizations for Fortran D. ,*Journal of Parallel and Distributed Computing*, Vol.21,pp.27-45 ,(1994).

[4] Manish Gupta and Prithviraj Banerjee: Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers. *IEEE transactions on parallel and distributed systems*, Vol. 3, No. 2(March 1992).

[5] Jingke Li and Marina Chen: Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays, *The 3rd Symposium on the Frontiers of Massively Parallel Computation* , pp.424-433,(Oct 1990).

[6] Mongenet, C.: Mapping for communications minimization using distribution and alignment, *Parallel Architectures and Compilation Techniques* (1995).

[7] Vasanth Balasundaram, Geoffrey Fox, Ken Kennedy, Ulrich Kremer: A Static Performance Estimator to Guide Data Partitioning. *ACM* (1991).

[8] Michael Wolfe: Tiny A loop Restructuring Research Tool, Technical Report,Oregon Graduate Institute of Science and Technology, (June 16,1993).

[9] Jack Dongarra,Jeremy Du Croz, Iain Duuff and Sven Hammarling: A Set of Level 3 Basic Linear Algebra Subprograms, Technical Report,Illinois Mathematics and Computer Science Division, (Auguest 1998).

[10] 後藤慎也: データフロー解析にもとづく並列化コンパイラの通信最適化手法, 特別研究報告, 京都大学情報工学科 (1995).

[11] 三吉郁夫: プログラム並列化におけるデータ分割支援システム, 特別研究報告, 京都大学情報工学科 (1993).

[12] 三吉郁夫: 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, 並列処理シンポジウム *JSPP'95* , pp.51-58,(1995).

[13] 笠原博徳: 並列処理技術,(コロナ社,1991).

[14] 津田孝夫: 数値処理プログラミング, 岩波講座ソフトウェア科学 9 , 第 3 章,(岩波書店,1988).