Communication-Parallelism Graph による データ自動分割手法

齊藤哲哉　　城和貴　　福田晃

奈良先端科学技術大学院大学 情報科学研究科

逐次のアプリケーションをワークステーションクラスタのような分散環境の上で分散化しようとするとき、データをどのように分割し配置するかが大きな問題となる。分散化を行うソフトウェアの多くは、これらの作業を利用者に任せているが、自動的に行う手法もいくつか提案されている。本稿では、データの分割と配置を一度に決定することができる通信-並列性グラフ (Communication-Parallelism Graph) を用いて、逐次のアプリケーションをリモートプロシージャコールベースの分散化コンパイラ SCIDDLE でコンパイルできる形式に変換するトランスレータの実装方針について述べる。

Automatic Data Partitioning Strategy with Communication-Parallelism Graph

Tetsuya Saito　　Kazuki Joe　　Akira Fukuda

Graduate School of Information Science
NARA Institute of Science and Technology

Data distribution is a key aspect when sequential application is distributed on the distributed environment such as network of workstations. Though most of distributed software leaves this task to users, several strategies to process this task automatically are proposed. One of these strategies is the Communication-Parallelism Graph(CPG), which process data alignment and distribution at the same time. In this paper, we discuss our strategy with CPG to translate sequential applications into distributed programs which can compile with an RPC-based distributed compiler, SCIDDLE.

1 はじめに

近年、スーパーコンピュータはもとより、ワークステーションやパーソナルコンピュータの性能の向上は著しく、低廉な価格で高性能なコンピュータを容易に入手できるようになってきている。特に、パーソナルコンピュータに至っては、数年前のワークステーションクラスの性能を持つものも出てきており、個人でかなり大規模なアプリケーションを実行することができる環境が整いつつある。

しかし、1台のコンピュータという枠組の中では、どんなにCPU等の性能が向上しようとも、いずれ物理的な限界が見えてくるだろう。

そんな中、LANやインターネット等のネットワークに接続している複数のコンピュータをあたかも一台の並列計算機のように見たてて利用できるようにするソフトウェアが現れてきている。PVM [4] はその代表的なソフトウェアの一つである。

これらのソフトウェアによって構築される仮想的な並列計算機というのは、スケーラビリティに優れているであるとか、眠っている資産を有効利用できるという利点がある反面、まったく共有空間を持たないという特性のために、データをいかに分割して持たせるかという問題に直面する。なぜなら、このような環境では、ネットワークを介して互いにデータをやりとりしなければならないため、その通信にかかるコストは非常に高く、データ分

割の方法いかんでシステム全体の性能が大きく左右されるからである。また、データの送受信であるとか、データのマーシャリング、アンマーシャリングといった低いレベルの処理をユーザ自身が明示的に指定する必要があるという欠点がある。

スイス連邦工科大で開発された分散化コンパイラ SCIDDLE [6] も仮想的な並列計算機を構築するためのソフトウェアのひとつである。SCIDDLE の場合、リモートプロシージャコールを用いることで、それらの低いレベルの処理をユーザから隠すことに成功している。しかし、SCIDDLE でもその他のソフトウェアと同様に、タスクの分割やデータ分割などはユーザが明示的に指定する必要がある。

これらの繁雑な作業が、並列/分散処理を行おうとするユーザのネックになっていると考えられる。新たにアプリケーションを構築するときはもちろん、既存の複雑で大規模なアプリケーションの構造を理解してうまく分散させるには、それ相当の知識と熟練が必要となるだろう。

そのような問題を解決するために、既存のアプリケーションを自動的に分散化を行うツールの開発が行われている。

そこで我々は、逐次のアプリケーションを自動的に分散化するために、プログラム分割やデータ分割を自動的に行い、分散化コンパイラ SCIDDLE でコンパイルできる形にして出力するツールの開発を行っている。

もちろん、自動的に最適な分散化を行うのは困難であるが、分散化コンパイラへの出力の形を取ることで、ユーザが後に変更や保守をしようとしたときに有利に働くと考えられる。

そこで本稿では、Communication-Parallelism Graph と呼ばれるグラフを用いて、SCIDDLE で自動的にデータ分割を行う枠組について考えてみることにする。

2 分散化コンパイラ SCIDDLE

SCIDDLE は、クライアント-サーバモデルに基づくリモートプロシージャコール (RPC) システムである。SCIDDLE では、プロセス間通信はすべて RPC によって行われる。RPC は、呼び出し側 (クライアント) と呼び出された側 (サーバ) のプログラムが異なる 2 つのアドレス空間 (あるいは、別々の 2 台のコンピュータ) 上で実行されることを除けば、通常の手続き呼び出しと同じである。SCIDDLE では、通常の同期 RPC による分散処理だけでなく、非同期 RPC による並列処理もサポートしている。

利用者は、簡単な宣言言語によって、リモートインターフェース定義と呼ばれるクライアントとサーバ間のインターフェースを記述するだけで、それに伴うデータの送受信等の通信にかかわるコード (スタブと呼ばれる) を自動的に生成することができる。図 1は、行列積を求めるプログラムのためのインターフェース定義の例である。

```
INTERFACE MatMult;
CONST
    Max = 100;
TYPE
    Matrix = ARRAY [Max, Max] OF REAL;
PROCEDURE Multiply(
    IN n, lo, hi    : INTEGER;
    IN A[n, n]      : Matrix;
    IN B[n, lo:hi] : Matrix;
    OUT C[n, lo:hi]: Matrix) : ASYNC;
END
```

図 1: 行列積のインターフェース定義

インターフェース定義は大きく分けて、サービス名宣言部 (INTERFACE)、定数宣言部 (CONST)、型宣言部 (TYPE)、コンテキスト変数宣言部 (CONTEXT)、手続き宣言部 (PROCEDURE) の 5 つの部分から構成される。

サービス名宣言部 クライアントが接続するサービスの名前を宣言する。

定数宣言部 型宣言に必要な定数を宣言する。

型宣言部 手続きの引数として渡される変数の型を宣言する。

コンテキスト変数宣言部 サーバの静的データ空間に送られる変数を宣言する。コンテキスト変数は、サーバの大域変数となり、RPC の行われている間ずっと存在する。

手続き宣言部 クライアントからの依頼によってサーバが実行する手続きの型 (同期 RPC(SYNC)) か非同期 RPC(ASYNC))、引数の型や方向属性を指定する。

このリモートインターフェース定義から、クライアントスタブ、サーバスタブを生成する。スタブには、手続きの引数をクライアントとサーバ間でやりとりするために必要なコードが含まれている。よって、ユーザは、インターフェース定義とクライアントプログラム、サーバプログラムを記述するだけで、分散アプリケーションを記述できるわけでである。

インターフェース定義から生成されたスタブと、ユーザが用意したクライアントプログラムとサーバプログラムは、SCIDDLE のユーザライブラリと一緒にリンクされる。このライブラリには、サーバが提供するサービスの初期化やサーバプロセスの起動や停止、非同期呼び出しの管理といったルーチンが用意されている。また、SCIDDLE では、C、C++、Fortran

77 がプログラミング言語としてサポートされている。

図 2は、その様子を示したものである。

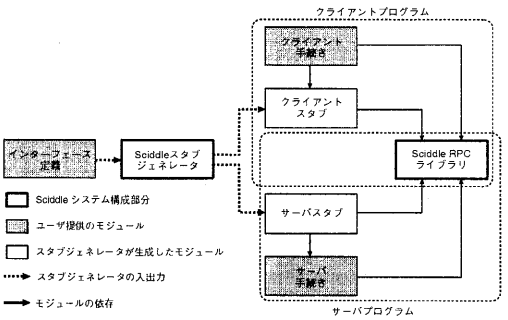

図 2: SCIDDLE のアプリケーションの構成

SCIDDLE の大きな特徴の一つは、配列を効率良く、容易に扱うことができるような機構が用意されていることである。

まず、図 1の例の型宣言部分に現われているように、配列型が用意されており、16 次元までの配列を扱うことができる。これらの配列には、**PACKED**、**UNPACKED** という属性と、**STATIC**、**DYNAMIC** という属性を付けることができる。前者は配列の添字空間に対するものである。**UNPACKED** の場合、元の配列の添字空間を保ったまま転送されるが、**PACKED** の場合、配列の添字空間を、要素のすべての添字が「ゼロ」になる点 (**(1,1,1,1, ...)**) に向かってシフトされる。後者は、記憶クラスに関する属性であり、配列を静的に確保するのか、動的に確保するのかを指定することができる。

次に、クライアントからサーバへ RPC として発行される手続きの引数に対して、方向属性を付けることで、データの送信方向を指定することができる。指示属性には、**IN**、**OUT**、**INOUT** の 3 種類が用意されている。クライアントからその変数を送信するが、サーバからその変数を返してもらう必要のない場合は **IN**、クライアントからはその変数を送信しないが、サーバからその変数を返してもらう必要がある場合は **OUT**、ともに必要な場合は **INOUT** を指定することで、配列の分配を効率よく行うことが可能である。

最後は、ビュー (View) と呼ばれる機構である。これは、部分配列を各サーバに分散させる手段で、タスクの実行には使用しない配列要素を転送する際に生じる通信のオーバヘッドを避けるため用意されている。ビューは、正規表現による部分配列の指定であって、配列の次元ごとに<開始インデックス>:<幅>:<終了インデックス>という組の集合で構成される。次元 d に対するビューの添字の範囲は 1 から n_d である。ただし、n_d は次元 d における要素数である。また、ビューの範囲を選択する際に、RPC として発行される手続きの引数リストにある変数を使用することもできる。これを境界変数と呼んでいる。ただし、その変数の方向属性は **IN** にしておかなければならない。図 1の場合、**n**、**lo**、**hi** が境界変数である。境界変数を利用することで、実行時に動的に部分配列を選択することが可能となる。

この章では、SCIDDLE の、特にデータ転送に関する項目についてかなり詳細に説明した。これらは CPG を実装するのに、非常に有用なものが多いと考えられる。これについては 4 章で説明する。

3 通信-並列性グラフ

通信-並列性グラフ (Communication-Parallelism Graph; 以下 CPG と略す) は、Universitat Politécnica de Catalunya の Jordi Garcia らによって提案されたものである [1]。

CPG は重みつき有向グラフであり、フェーズと呼ばれるブロック内にある配列への代入文を解析して構築する。ここでフェーズとは、ループ本体に含まれる配列参照の添字の位置に帰納変数があるループネストのことを言う。このフェーズには、その帰納変数を定義する周辺のループも含まれる。各フェーズでは、それぞれ異なるデータ分割戦略を持ち、フェーズ間ではデータの再配置も行われる。

CPG の頂点は列ごとに構成され、それぞれの列はフェーズ内の配列を示している。また、各列には、プログラムに現われるすべての配列の中で一番大きい次元数 d だけ頂点がある (d より少ない次元数しかない配列の場合には、足りない分だけ頂点を追加する)。CPG の辺には、データの移動と並列性を表し、それぞれ、data movement edge、parallelism hyperedge と呼ばれる。

data movement edge フェーズ内にある、ある 2 つの配列のそれぞれの次元の間で、どう alignment できるかの候補を表す。候補は参照パターン (reference pattern) を解析することで選択される。読み出される配列から、更新される配列へ矢印を引く。また、重みは、その 2 つの次元が配置され、分散された場合のデータ移動にかかるコスト (実行時間) を表す記号式である。

また、data movement edge には、再配置情報も含められる。もし、フェーズ P_i で参照されている配列を、それより後にあるフェーズ P_j でも参照しているなら、$d \times d$ 本の辺で繋ぐ。重みは、同じ次元に関しては **NULL** を、違う次元には **Redistribution** プリミティブのコスト (実行時間) を付加する。

Parallelism hyperedge プログラム内でデータ依存解析をして、並列実行可能なループを見つける。そして、そのループ本体に現れる配列への代入文を解析し、添字部分に並列実行可能なループのインデックス変数が現れている次元に対応する頂点を辺で結んでいく。hyperedge とは、2つ以上の頂点を結合できるという意味である。こうすることによって、ループを並列化する際に分配しなければならない配列の次元が結ばれる。この辺に付加される重みは、与えられたプロセッサ数で並列実行したとして、削減できる実行時間を表す。

ここで、実際のコードから生成された CPG を見てみることにする。図 3がその例である。この 2 つの DO ネストがそれぞれフェーズとなり、各フェーズごとにループ本体に含まれる配列の代入文について解析を行う。図 4は、図 3 のコードから得られた参照パターンである。データの移動には、あらかじめ決めておいたプリミティブを用いる。表 1は、参照パターンとそれに対応するプリミティブを示している。

```
DO I = 2, N
  DO J = 1, N
    A(I, J) = A(I - 1, J) + B(I, J)
    B(I, J) = A(I, J) + 1
    C(I, J) = B(I, J) + 2
  ENDDO
ENDDO
DO K = 1, N
  DO L = 1, N
    B(L, K) = C(K, L) + 1
    D(K, L) = C(K, L) + 2
  ENDDO
ENDDO
```

図 3: コードの例

```
Phase 1:
    A(I, J) ← A(I - 1, J)
    A(I, J) ← B(I, J)
    B(I, J) ← A(I, J)
Phase 2:
    B(L, K) ← C(K, L)
    D(K, L) ← C(K, L)
```

図 4: コードの例の参照パターン

これらを元に CPG を記述したのが図 5である。点線の矢印は、LOCAL MEMORY ACCESS を示している。また、loop J、K、L は paralellism hyperedge を示している。

ここで記述した CPG に、

表 1: 参照パターンとプリミティブ

パターン	プリミティブ
$i_p \cong j_p$	LOCAL MEMORY ACCESS
const $(i_p - j_p)$	ONE-TO-ONE
const (i_p)	MANY-TO-ONE
const (j_p)	ONE-TO-MANY
$i_p \neq j_p$	MANY-TO-MANY

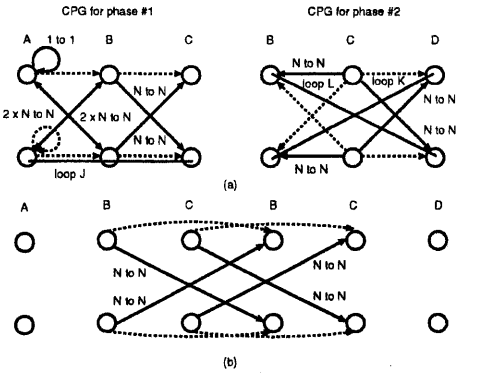

図 5: (a) 各フェーズの CPG (b) 再配置辺

- 解はパスである
- パスは各列の 1 つの頂点を通る
- パスには、選択した頂点を結ぶすべての辺を含めなければならない
- parallelism hyperedge は、パスに属するすべての頂点が含まれるときに選択される

という制約条件を課して、パスが通る data movement edge に付けられている重みの合計から、parallelism hyperedge に付けられた重みの合計を引いたものが最小になるようなパスを求める。これらを 0-1 整数計画問題として解く。求めた結果が図 6である。

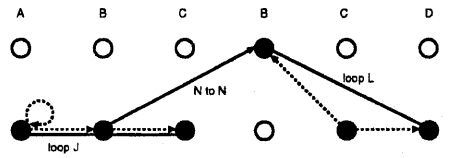

図 6: 最適な分割

このように、CPG を用いることで、なんらヒューリスティックな手法を用いることなく、通常は 2 ステップ (alignment と distribution) で行われるデータ分割が一度に行われていることがわかるであろう。

また、彼らは、静的多次元分割 [1]、動的分割 [3]、制御フローと BLOCK/CYCLIC 分割 [2] を考慮した CPG の提案を行っている。

静的分割は、動的分割における Redistribution プリミティブでコストが付けられている辺を選択しないものと同様なので、本稿では、動的一次元分割を対象としてトランスレータの実装を考慮してみることとした。

4 トランスレータの実装方針

SCIDDLE は、RPC を用いて分散化を行うシステムである。このことから、トランスレータに求められる役割の一つは、「SUBROUTINE 文を分散化させる」ことである。Fortran の SUBROUTINE 文を SCIDDLE の同期 RPC でそのまま分散化させるのは非常に簡単なことである。しかし、それではある程度の負荷分散にはなるが、性能向上という点から見ると、まったく利点がない。また、プログラム間解析は非常に困難であり、CPG では考慮されていないので、対象外とした。そこで、今回は、PROGRAM 文からなる、DO 文のネストのみから構成されるようなソースを対象に方針を考えることとする。

また、本来は、ソースプログラムの解析を詳細に行い、ソースの再構築を行うことで、非同期 RPC を用いることができるような SUBROUTINE 文を生成し、分散化することが理想であるが、これは非常に困難である。そこで我々は、イリノイ大学で開発された自動並列化コンパイラ Parafrase-2 [5] が並列化したソースプログラムを利用して分散化を行うこととした。Parafrase-2 によって、Fortran 77 で記述されたソースプログラムから、各イタレーションを並列に実行できる DO 文である CDOALL 文が生成される。これを利用すれば parallelism hyperedge も容易に検出することができるだろう。

図 7は、トランスレータによって行われる分散化の全体像を示している。

まず最初に考えなければならないのは、CPG によって最適な分割が求まった後に、どう SCIDDLE の枠組にあてはめるかという問題である。

我々は、既に f2sc と呼ばれるトランスレータの試作を完了している [7]。こちらも同様に、Parafrase-2 の出力するソースを元に分散化を行っている。f2sc では、プログラム内に現れるループネストの中で、CDOALL が一番外側にあるネストを対象として、そのループ本体を新たに SUBROUTINE 文として定義して手続きを作成し、これを非同期 RPC を用いて分散化させるという方針を取った。CDOALL 文のイタレーションはすべて独立に実行できるため、一番容易な手段として選択している。

CPG を用いる場合、f2sc と同じ方針でもよいのだが、それでは適応できる範囲がかなり限定される。そこで、CPG を用いる場合は、フェーズを手続きとしてサーバに分配することにした。

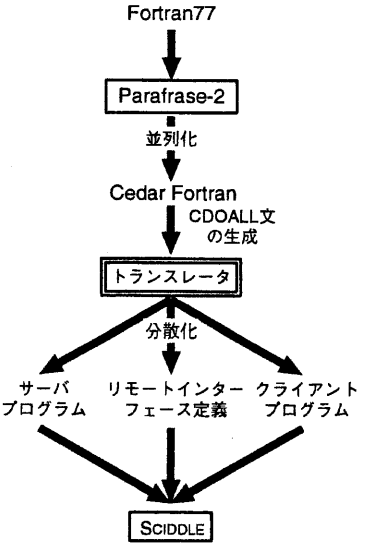

図 7: 分散化の流れ

こうすることで問題となるのは、フェーズ間での配列の再配置と、フェーズ内の代入文ごとに行われる、他のサーバとの配列の転送である。

f2sc では、すべてのデータを方向属性 INOUT でやりとりすることで、分配したデータの一貫性という問題を回避した。また、この時は単純に添字にしたがって配列の分割をしただけなので、どれが更新データで、どれが参照のデータかを考慮できずにすべてやりとりしていたため、通信が増大し、ワークステーションクラスタ上では性能向上を得ることはできなかった。

SCIDDLE では、コンテキスト変数と呼ばれる変数を宣言できることは前述のとおりである。これはサーバの大域変数となるため、owner computin rule を実装するにはよい手段になると考えられる。しかし、実際には、SCIDDLE が RPC で実装されている関係上、通信トポロジが木構造に限定されており、サーバ間の通信ができないという問題がある。そのために、サーバがそれぞれ持っているコンテキスト変数を他のサーバが参照する場合には、クライアントに一度データを戻して、また他のサーバに送るという手段を用いなければならない。すなわち、すべての正しいデータはクライアントに保持されている必要がある。したがって、フェーズ間で配列の再配置をする場合も、フェーズ内の代入文ごとに行われる他サーバとの配列の転送も、クライアント経由に行い、その分余計なデータ転送という代償を支払うことになる。

しかし、CPG によって更新されるデータと

参照されるだけのデータの種類や位置が明確に区別されるので、SCIDDLE の方向属性やビューを利用することで、サーバ間通信ができる場合よりは少し効率が落ちるが、`f2sc` ほどではなくなると考えられる。

クライアントとの通信が必ず必要となるのは、owner computing rule を実装するためには不向きであるので、SCIDDLE 自身の拡張も考慮しているところである。

最後に、辺につけられる重みの算出方法である。CPG は、プロファイルを用いることでそれぞれの辺の重み (data movement edge と parallelism hyperedge) を決定している。これについては、ある定数を仮定することで解決することとした。また、トランスレータによって、これらの値を取得できるようなコードを追加することも考えている。

5 おわりに

SCIDDLE と CPG は、ごくわずかな問題があるとはいえ、非常に親和性の高いものであると考えられる。今後は、今回考えた方針にのっとって、実装を行い、テストをして CPG を SCIDDLE に、また SCIDDLE を CPG にうまく適応させてより最適に近い自動分散化が行えるようにすることを考えている。

また、本稿では、ワークステーションクラスタのような通信コストが非常に高い環境を対象としているが、将来的には、nCUBE に SCIDDLE の実装を行い、CPG によるデータ分割を行う予定である。

参考文献

[1] Garcia, J., Ayguadé, E. and Labarta, J.: A Novel Approach Towards Automatic Data Distribution, *In Proceedings of the Supercomputing '95 conference* (1995).

[2] Garcia, J., Ayguadé, E. and Labarta, J.: Dynamic Data Distribution with Control Flow Analysis, *In Proceedings of the Supercomputing '96 conference* (1996).

[3] Garcia, J., Ayguadé, E. and Labarta, J.: A Framework for Automatic Dynamic Data Mapping, *In Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing* (1996).

[4] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press (1994).

[5] Girkar, M. B., Haghighat, M., Lee, C. L., Leung, B. P. and Schouten, D. A.: *Parafrase-2 User's Manual* (1995).

[6] Sprenger, C.: User's Guide to SCIDDLE Version 3.0, Technical Report 208, ETH Zürich, Computer Science Department (1993).

[7] 齊藤哲哉: 自動分散化コンパイラに関する研究:FORTRAN 77 から分散化コンパイラ SCIDDLE へのトランスレータの試作, 修士論文, 奈良先端科学技術大学院大学 (1996).