

ブロック同期に基づく並列アルゴリズムアニメーションシステム

3 Q-4

太田英憲 岩間一雄

九州大学工学部

1 はじめに

一般にアルゴリズムの動作を理解したり、他人に説明してその正しさを納得させることは容易でない。そこで、データ構造や動作に伴うデータの変化をグラフィックディスプレイを用いて可視化させ、直観的な理解を容易にする試みがなされてきた。このようなシステムはアルゴリズムアニメーションと呼ばれ、アルゴリズムの理解や検証及び教育において大変役に立つ。

直列アルゴリズムに対してはプログラム化して実際に計算機上で走らせるという手法が使える。デバッガなどの支援ツールも多数存在し、Balsa-II^[1]やTango^[2]といったアルゴリズムアニメーションシステムも存在する。ところが、並列アルゴリズムに対しては、アルゴリズムが対象とする並列計算機が存在しない場合もある。もちろん、アルゴリズムの段階でシミュレーションによって直列アルゴリズムに変換した後上述の直列アルゴリズムに対する手法を適用することは可能であるが、単純なものに限られるし、本来の目的に反する。

そこで、我々はP-RAM(並列ランダムアクセス機械)上での並列アルゴリズムを自動的に直列プログラムに変換してシミュレートし、さらに可視化までを行なうシステムを開発しているので報告する。

2 ブロック同期方式

P-RAM アルゴリズムの記述はその細部まで行なわれることはまれで、ほとんどの場合概略のみが与えられ、その理解はアルゴリズム開発者が有する共通の知識や約束および習慣に大きく依存している。例えば、定数ステップで実行可能部分は、それが5ステップであろうと100ステップであろうと、単にその動作を言葉で説明するだけで済ましてしまうことも少なくない。

一方P-RAMは完全同期式のモデルであって、全てのプロセッサで機械語命令が正確に1ステップずつ同時に実行される。そこで上記のような5ステップと100ステップの部分を同期させて実行させようとするなら、5ステップの部分に95ステップのNOP命令を挿入する必要がある。この作業は見かけよりもずっと大変なばかりでなく、それによってアルゴリズムの本質的部分を分かりづらくする恐れさえある。そこで我々はアルゴリズム記述の為にブロック同期という概念を導入した^{[3][4]}。これは

```
block { ... }
```

という構造を持ち、全てのプロセッサはブロック単位に同期して動作すると仮定する。例えば、2つの繰り返しループが共に3個のブロックからなる場合、個々のブロックの大きさは(定数ステップの範囲で)どうであろうと、同じ長さのループとみなされ完全に同期する。こうしてアルゴリズム開発者はブロックさえ明確にすればあとは従来通りの“大雑把な記述”を行なうこともできるし、高級言語も使用できる。

ブロック内部では同一アドレスのglobal変数に対してreadを先に行ないwriteするならば一度だけしか許されない。全てのwriteはそのブロック内での位置とは無関係に同時に実行される

Parallel Algorithm Animation Based on Block Synchronization

Hidenori Ohta, Kazuo Iwama
Kyushu University

と仮定する。1つのブロックは定数時間で実行可能でなくてはならず、条件ブロックで使用されるときにはアルゴリズム開発者が全体の動きを管理する必要がある。

3 システムの概要

並列アルゴリズムを可視化するためのアニメーションシステムは次の部分に分けられる。

- 入力ジェネレータ
- 並列アルゴリズムシミュレータ
- 出力コンバータ
- ビジュアライザ

入力ジェネレータはアルゴリズムが処理するためのデータを作成するライブラリの集まりであり、再コンパイルの必要無しに使える。これはアルゴリズムに対して様々な種類のデータを与えたい場合、アルゴリズムを記述しているプログラムを変更することなくデータを変更できるという利点、また、同じデータを別な手法のアルゴリズムに対して使用できるという利点がある。

並列アルゴリズムシミュレータは前章で述べたブロック同期方式を用いて記述されたアルゴリズムをシミュレートするもので、システムを中心となる(5章で述べる)。

出力コンバータはシミュレータから観察したいデータを得、それを変換してビジュアライザに渡す。そのためにプロトコルを定めてデータを出力フォーマットに従って変換し、それを送る。但し、出力コンバータとビジュアライザとの通信は完全に可視化部の内部だけに限られるため、シミュレータやユーザにとっては出力コンバータとビジュアライザは同じ物として認識する。

ビジュアライザは実際に画面をディスプレイに描画するもので、出力コンバータが変換した円、直線、色指定など表示するための基本的データの列からなる描画コマンドを受け取る。コマンドを受け取ると画面サイズなどのパラメータに従い、適切な画面を表示する。このように出力部をアルゴリズム部と分ける理由は、入力ジェネレータと同じようにアルゴリズムを変更することなく出力フォーマットを変えることができる、同じフォーマットを様々なアルゴリズムに対して用いることができるなどである。

4 ユーザインタフェース

並列アルゴリズムシミュレータはP-RAMをモデルとするため、通常のプログラミング言語に何らかの拡張を施さなければアルゴリズムを記述することができない。このシミュレータはUnixをOSとするワークステーション上で動作させることを目的としているため、C言語を元に以下の機能を加え拡張した言語を仮想してアルゴリズムを記述する。

- プロセッサに対してのglobal変数とlocal変数の概念
- プロセッサの同期をとるための制御構造
- 自分のプロセッサ番号を示す変数

プロセッサに対するglobal変数を表すためにpara_global, para_localという2種類の識別子を定義する。para_globalが指定されると全てのプロセッサに共通の変数(いわゆる共有メモリ)であることを示し、

```
para_global local int i;
```

のように使用される。para_localが指定されていれば各プロセッサで独立の変数であることを示す。

プロセッサ同期のための制御構造は2章で述べた通りであり、プロセッサ番号を知るためには `extern int proc_id` という変数を定義する。

アルゴリズムを実行する際、ユーザはビジュアライザに対してどの変数(群)を可視化するかを指示しておく必要がある。その指示はプログラム中に

```
visual()
```

という関数の形で指定する。これは引数を2つ持ち、最初はビジュアライザに対するイベントの種類で、次は可視化したい変数のアドレスである。イベントとはそれを通知された時のビジュアライザの動作を示すもので、例えば、`new`, `set_v`, `set_e` がある。出力コンバータはイベントの種類により転送されたデータのタイプを知り、ビジュアライザに描画コマンドを発行する。例えば、

```
typedef int[] Vlist;
typedef struct {int e1,e2} Elist[];
typedef struct {
    int n,m;
    Vlist vlist;
    Elist elist;
} Gdata;
```

```
Gdata gdata;
```

```
;
```

```
visual("new",gdata)
```

のように `visual()` が呼び出されると、ビジュアライザでは `gdata` をグラフデータとして扱いディスプレイ上にグラフを表示する。注意しなければならないことは、プログラム中に記述した `visual()` は単にビジュアライザに対してデータを与え、画面を更新するきっかけを与えるに過ぎないということで、実際のデータの変更は `visual()` を呼び出す前にアルゴリズム中に記述されていなければならない。また、画面を更新するタイミングもアルゴリズム記述者に委ねられている。

5 並列アルゴリズムシミュレータ

実際にシミュレータでプログラムを実行するためには、何らかの形で直列で動作するよう変換する必要がある。そこで、我々はアルゴリズムを記述したプログラムを実行する前に、前処理として通常のC言語に変換する。

その手順は、まず、`global` 変数は基本的に通常の変数として扱うことができ、`local` 変数はプロセッサ数分の要素を持つ配列変数として扱えるため、そのように変換する。但し、プログラムを実行する時は1ブロックずつプロセッサ番号順に実行するため、`global` 変数への `write` の結果をそのまま書き込むとプロセッサ間で一貫性が取れなくなる。従って、`global` 変数への `write` があるブロックではそのための一時変数を用意し、`write` はその変数に対して行ない、実際の `global` 変数への書き込みは全てのプロセッサが実行を終えた時点で(同時書き込みを許す)P-RAMの規則に従って行なわれる。

各プロセッサが実行するブロックは異なっている可能性がある。そのため、プログラムを実行するためにはアルゴリズムの動きを管理する機構が必要となる。そこで、シミュレータ内に管理プログラムを用意する。管理プログラムではプロセッサが実行すべきブロックを管理し、ブロックを実行する他に、`global` 変数の管理などを行なう。各ブロックをシミュレートするに当たっては、プログラム中に記述された `block { ... }` の部分を、前処理で関数として呼び出せる形に変換して、管理プログラムはその

関数を呼び出すことによってブロックを実行する。

シミュレータはプロセッサ数が N の場合、以下の手順でプログラムを実行する。

Step 1: 各プロセッサの実行位置を最初のブロックに設定 ($i=0$)

Step 2: プロセッサ番号 i の実行位置に従い1ブロックを実行し、次の実行ブロックを設定

Step 3: i を1増やし $N-1$ まで **Step 2** を繰り返し実行

Step 4: グローバル変数への `write` の結果を実際に書き込み、**Step 2** へ

6 ビジュアライザ

ビジュアライザについてはその第1版が完成している。グラフアルゴリズムを対象として、グラフのデータをディスプレイ上に表示することができる。

グラフをディスプレイ上に表示する上での問題は、いかに頂点や枝が重ならないように配置するかということである。配置にはいろいろな方法があると思われるが、第1版では二つの方法を考案した。

一つは、ディスプレイに仮想的に円を描きその円周上で等間隔に頂点を配置する方法である。この方法では、枝が円の内部を通るので頂点と重なることがない反面、多数の頂点を表示するには広い面積が必要である。

もう一つは、仮想的に格子を描きその交点上に互い違いに頂点を配置する方法である。この方法では表示空間全体で頂点が均等に配置されるので、単位面積あたりの表示可能頂点数は第一の方法より多いが、次数が大きくなると頂点と枝が重なり合ったり、枝同士が完全に重なったりするという欠点がある。

グラフの属性データの授受は、とりあえずファイルを通じて行なうこととした。そのフォーマットは、表示ステップ毎に全頂点の配列データのイメージをダンプしたものであり、データをファイルから読み込むとそれらのグラフ問題における意味やデータ相互の関連性については考えないで、ただ表示するためのデータとして扱う。この方法は原始的であり表示するデータをアルゴリズム側で作成するため、複雑な計算を必要とする場合はアルゴリズム側にかかる負担が大きくなるが、分かりやすく一般性が高いと思われる。

7 むすび

現在、対象とする並列アルゴリズムをグラフアルゴリズムに限定して開発を進めている。上で述べたように、ビジュアライザの部分の開発が先行しているが、シミュレータの部分についてもプロトタイプが出来上がりつつある。

参考文献

- [1] M.H.Brown, "Exploring Algorithms Using Balsa-II," IEEE Computer, Vol. 21, No. 5, pp.14-36, May 1988.
- [2] John T.Stasko, "Tango: A Framework and System for Algorithm Animation," IEEE Computer, Vol. 23, No. 9, pp. 27-39, Sep. 1990.
- [3] 太田, 岩間, "ブロック同期方式に基づく並列アルゴリズムシミュレータ," 平成4年度電気関係学会九州支部連合大会, 1327, pp.797, 1992.
- [4] 富田, 岩間, "ブロック同期方式による並列アルゴリズムの記述とそのプログラム化," 情報処理学会第46回(平成5年前期)全国大会