共有メモリ式SIMD型並列アルゴリズム解析ツール

6D－7

坂本 圭 †　藤本 典幸 †　魚井 宏高 †　萩原 兼一 ‡　首藤 勝 †

†大阪大学 基礎工学部　‡奈良先端科学技術大学院大学

1 はじめに

本研究では並列アルゴリズムの中から最も理論的な研究の進んでいる PRAM [1] に代表される共有メモリ式SIMD型並列アルゴリズムに注目し，共有メモリ式SIMD型並列アルゴリズム解析ツール PRASS の構築を行なった．PRASS では本研究室で開発中の並列処理言語 ELPA [2] を用いて並列アルゴリズムを記述し，並列アルゴリズムの解析を行なう．そのための機能として，プログラムの実行制御や，プログラムの内部状態の調査・変更，また，プログラムの実行時間と仕事量 (work) [1] の計測といった機能を備えている．これらの機能により並列アルゴリズムの開発・理解を効果的に行なうことができると考えられる．

2 PRASS の機能

PRASS は UNIX の標準的なワークステーション上で実現されており，端末から入力されたコマンドに応じて動作を行ない，結果を端末やファイルに出力する．PRASS の持つ機能は次の3種類に大きく分かれている．

- シミュレータ的機能
- デバッガ的機能
- アナライザ的機能

以下にこれらの機能についてそれぞれ述べていく．

2.1 シミュレータ的機能

並列処理言語 ELPA を用いて記述された並列アルゴリズムを実行する機能である．ELPA は，C 言語に並列動作を記述するための parado 文を追加したものである．

```
parado (式1 to 式2 step 式3)
    task (int 識別子) <= (識別子のリスト)
        { parado 文本体 }
```

図1．parado 文の構文

parado 文は同じ動作 (parado 文の本体) を並列に実行するための文であり，parado 文が実行されると (式$_2$ − 式$_1$)/式$_3$ + 1 個のタスクが生成され，それぞれのタスクが parado 文の本体を実行する．各タスクは共有変数にアクセスすることにより通信を行ない，同期して命令の実行を行なう (SIMD)．なお，task の横に書かれている "識別子" はタスクの識別用変数の変数名であり，"識別子" のリストはタスクがアクセスできる外部変数の変数名である．図2はプレフィックスサム [1] を計算する関数 prefix を ELPA を用いて記述した例である．

```
 1 void prefix(int n, int x[], int s[])
 2 {
 3     int i, y[10000], s2[10000];
 4
 5     if (n == 1)
 6         s[1] = x[1];
 7     else {
 8         parado (1 to n/2)
 9             task(int i) <= (y, x)
10                 { y[i] = x[2*i-1] + x[2*i]; }
11         prefix(n/2, y, s2);
12         parado (2 to n step 2)
13             task(int i) <= (s, s2)
14                 { s[i] = s2[i/2]; }
15         parado (1 to n step 2)
16             task(int i) <= (s, s2, x)
17                 { s[i] = s2[(i-1)/2] + x[i]; }
18     }
19 }
```

図2．関数 prefix

シミュレーション時にプログラム内で起動されるタスク数に制限はない．PRASS ではシミュレーションを仮想計算機の動作をシミュレートすることにより行う．このため，ELPA で書かれたプログラムは仮想計算機のアセンブラコードに対応した中間コードにコンパイルされてから実行される．従ってプログラムの文法的な誤りはシミュレーションを行なう前に検出される．なお，このアセンブラコードは端末やファイルに出力することができ，PRASS が ELPA プログラムを内部でどのように実行しているかを知ることが出来る．

図3はシミュレーションの例であり，上で例としてあげた関数 prefix を含むプログラム prefixsum を読み込み，実行した様子を表している．なお，プログラム prefixsum ではプレフィックスサムの入力の要素 x[i] の値をそれぞれ i とし，演算は加算としている．

```
(prass) open prefixsum
Compile...
done.
(prass) run
Input:   1  2  3  4  5  6  7  8  9 10
Result:  1  3  6 10 15 21 28 36 45 55

Time = 187, Work = 332
```

図3．プログラムの実行例

2.2 デバッガ的機能

PRASS ではプログラムの実行の中断・再開やステップ実行の機能がブレークポイントを用いることにより実現されている．ブレークポイントやエラーの発生によってプロ

An analyzer for parallel algorithms of SIMD shared memory computers
Kei SAKAMOTO †, Noriyuki FUJIMOTO †, Hirotaka UOI †, Kenichi HAGIHARA ‡, and Masaru SUDO †
†Faculty of Engineering Science, Osaka University
1-1 Machikaneyama,Toyonaka,Osaka 560,Japan
‡Nara Institute of Seience and Technology
8916-5, Takayama, Ikoma, Nara 630-01, Japan
E-mail: {sakamoto, fujimoto, uoi, hagihara, sudo}@ics.es.osaka-u.ac.jp

グラムの実行が中断された場合，プログラムの内部状態を調べることができ，現在の行番号や，各プロセスの状態やローカル変数の内容，タスクの数，および実行時間と仕事量について知ることができる．図4が2．1節で述べたプログラム prefixsum における表示例であり，関数 prefix の１０行目で実行を中断した時のプログラムの状態を様子を表している．表示から，現在プログラムは２６行目で停止しており，タスクは6個あるが，0番のタスクは停止状態にあり活動状態にあるのは１～５のタスクだけである，ということが分かる．

```
current line is 26.
Task 1 : STOP : i = 0, x[], s[], num = 10,
 n = 10, x[], s[], i = 0, y[], s2[]
Task 2 : RUN  : i = 1
Task 3 : RUN  : i = 2
Task 4 : RUN  : i = 3
Task 5 : RUN  : i = 4
Task 6 : RUN  : i = 5
Running/Total Task = 5/6
Time = 17, Work = 17
```

図4．プログラムの内部状態表示例

また，これらの情報を実行中に絶えず端末やファイルに出力することもできる．この場合，上で述べた情報に加えてプロセスの実行した命令も出力されるため，これを実行履歴として利用することもできる．

プログラムの実行中断時には，プログラムの内部状態を調べるだけでなく，それを変更することも可能であり，変数の内容の変更機能と，その場でプログラムのコードを与えて実行させる機能が実現されている．

2.3 アナライザ的機能

PRASS では，プログラムが正常に終了した場合，その実行時間と仕事量を知ることができる．実行時間 (仕事量) は中間コードから得られた仮想計算機が実行した命令数にもとづいて計算される．実行時間 (仕事量) がカウントされる命令としては変数への参照・書き込み，算術計算，ジャンプ命令などがあるが，すべて1実行時間かかるものとして統一されている．また，仕事量は命令を実行したタスクの数を全ステップについて合計したものである．

例えば a[i] = b は次のようなコードに変換されるが，

```
LOAD b;   LOAD i;   STORE a[]
```

このコードの実行には変数 b,i への参照がそれぞれ1回ずつ，変数 a への書き込みが1回行なわれるため，3実行時間かかる．

ただし，プロセッサ数がタスク数よりも少ない場合には，1命令の実行に2実行時間以上かかってしまう．現実の並列計算機ではプロセッサが有限の数になっているが，これに合わせてプロセッサ数を初期値の無限大から有限の数に変更した場合，1命令にかかる実行時間は，タスク数 ／ プロセッサ数 となる (端数切り上げ)．これはプロセッサが時分割でタスクを切替えて命令を実行するためである．ただし，この切替えは非常に高速に行なわれるため，余分な実行時間はかからないものとしてる．図5は proc コマンドを使ってプロセッサの数を変更しながらプログラム prefixsum の実行時間 (仕事量) を測定している例である．プロセッサの数の初期値は無限大としてある．

```
(prass) run
    <略>
Time = 187, Work = 332
(prass) proc 2
(prass) run
    <略>
Time = 251, Work = 332
(prass) proc 1
(prass) run
    <略>
Time = 332, Work = 332
```

図5．プロセッサ数を変更した例

プログラムによっては，データの初期化や実行結果の表示などにかかる実行時間 (仕事量) のため，アルゴリズム本体の実行時間 (仕事量) が分からなくなってしまう場合がある．このため，本システムでは実行時間 (仕事量) の計測に関する制御情報をプログラム内に埋め込み，アルゴリズム本体の実行時間 (仕事量) だけを計測することができる．制御情報の埋め込みは次の3つの疑似ライブラリ関数の呼び出しをプログラム内に追加することにより行なわれる．

creset	実行時間 (仕事量) のカウント値を0に戻す．
cstart	cstop 関数によって中断された実行時間 (仕事量) のカウントを再開する．
cstop	実行時間 (仕事量) のカウントを中断する．

図6はプログラム prefixsum とプログラム prefixsum から疑似ライブラリ関数の呼び出しを取り除きアルゴリズム本体以外の部分の実行時間 (仕事量) も一緒に計測するようにしたプログラム prefixsum2 の実行時間 (仕事量) を比較した例である．このように両者には実行時間と仕事量に大きな違いがあり，計測の制御がアルゴリズムの解析に不可欠であることが分かる．

```
(prass) open prefixsum
    <略>
Time = 187, Work = 332
(prass) open prefixsum2
    <略>
Time = 2643, Work = 2833
```

図6．計測の制御による効果

3 おわりに

本システムはオブジェクト指向により設計・実装を行なったが，これにより仮想計算機の各構成要素を直接オブジェクトとして表現でき，シュミレーションを簡潔に実現することができた．

本システムで提供しているプログラムの内部状態表示機能は，表示方法がテキストベースのものであるため，データサイズやプロセッサ数が多い場合にはどうしても理解が困難になってしまう．この問題を解決するため，グラッフィクスを利用してプログラムの内部状態の表示を行なうことが今後の課題としてあげられる．

本研究は一部文部省科研費 (平成5年度重点領域研究 (1)04235104) の補助による．

参考文献

[1] Joseph JaJa: “An Introduction to Parallel Algorithms” Addison-Wesley Publishing Company (1992).

[2] 藤本典幸：“ELPA 言語仕様第1版” 大阪大学基礎工学部情報工学科首藤研メモ No. 50 (1992)