超並列配線マシンのソフトウェア環境

4P−5

大木由江* 河村薫 進藤達也 澁谷利行 三渡秀樹
株式会社富士通研究所

1. はじめに

並列計算機のプログラミング開発は，並列性の記述・デバッグともに難しい問題である．

我々は，並列性およびMAPLEのハードウェア的特徴を容易に表現できる記述方法として，C言語と専用のマイクロ言語を用いて記述することで実現した．また，C言語とマイクロ言語を自動的に分離するマイクロコンパイラの開発を行った．

2. マイクロ記述とマイクロコンパイラ

MAPLEのアプリケーションは逐次処理部分をC言語で，プロセッサアレイ実行部分を専用のマイクロ言語で記述する．プログラムは，図1に示すようにC言語の記述中に，二重の中括弧で括ってマイクロ記述をする．

マイクロコンパイラは，二つの言語を自動的に分離し，Cの記述はコントロールプロセッサの実行形式とし，マイクロの記述は96ビット幅のマイクロコード列に変換する．コントロールプロセッサは，マイクロ記述部分については，マイクロコード列の実行をシーケンサに命令する．以後，シーケンサは，マイクロコードの実行制御を司り，プロセッサアレイや収集演算回路に毎クロック命令を送る．コントロールプロセッサは，マイクロ命令の終了を待たずに引き続くC言語記述部分を実行する．

ハードウェアを簡素化するために，インターロックの調整はマイクロコンパイラで行っている．

3. マイクロ記述言語

表1にマイクロ記述言語の基本命令を示す．マイクロ記述では，MAPLEのアーキテクチャ特有の，隣接プロセッサ間の通信(North, East, West, South)，コントロールフラグ(CF)を用いた記述や収集演算回路を用いた記述，イベント(EVENT)の記述，ウィンドウに対する記述，コントロールプロセッサとの同期等を記述することができる．

3．1 変数宣言

変数は，MAPLE上のメモリに対して宣言することができる．内部レジスタの変数宣言，外部メモリ上の変数宣言，グローバルメモリ上の変数宣言である．図2に変数宣言の例を示す．

```
プログラム記述例()
{
  C言語
  {{
      マイクロ記述
  }}
  C言語
}
```

図1 プログラム記述例

変数の宣言は，最初にどのメモリに書くかを示す識別子，次に変数の型，最後に変数名を書く．識別子は，プロセッサエレメント内のメモリについてはstatic を，外部メモリについてはmem を，グローバルメモリについてはglobalを用いる．変数の型としては，デフォルトで用意したものはbit であり，1 bit の値を示す．この他に構造体を用いて宣言できる．

3．2 マイクロ命令

図3にプログラム記述例を示す．

表1 マイクロ基本命令

種別	命令
算術演算	ADD, SUB, MIN, MAX, COMP 上記のCARRY付き命令
論理演算	AND, OR, INV, NAND, NOR, EOR
通信	North, East, West, South
特種	PASS, ONE, ZERO, NOP
特種関数	queue(), schReq(), history(), eventReset(), load(), save(), sync(), waitPE()

Software Environment of the Massively Parallel Routing Machine
Yoshie Ohki, Kaoru Kawamura, Tatsuya Shindo, Toshiyuki Shibuya and Hideki Miwatari
FUJITSU LABORATORIES LTD., KAWASAKI

(1) は隣接プロセッサを用いて演算する例である．それぞれaは，通信を使わないで演算する手法，bは，隣接プロセッサの値を用いて演算する手法である．

(2) はコントロールフラグを用いることで，プロセッサ単位で演算をするかしないか制御する例である．CFの値が0のときコロンで区切られた左側の演算を実行し，CFの値が1のとき右側の演算を実行する．

(3) はウィンドウ制御の時に用いる例である．これは，擬似変数EVENTに1を代入することによりイベントが発生する．

(4) は収集演算回路を用いた演算の例である．

(5) はコントロールプロセッサとの同期を示す．コントロールプロセッサは，マイクロ命令列の起動を行うと，その終了を待たずに次の処理を継続実行する．そこで，マイクロプロセッサ側でマイクロ命令列の終了を待ちたい場合に同期をとる必要がある．'waitPE()'はマイクロ命令列の'sync()'の処理が終了するまで次の命令を実行せずに待ち続ける．

4. その他のソフトウェア開発環境

MAPLEとフロントエンド計算機との間で通信を行うライブラリを用意しており，入出力関数はC言語における記述法と同一のインタフェースでデータの送信を行うことができる．初期データのダウンロードやアップロード，あるいはデバッグ用の入出力を実現している．また，プロセッサ内メモリや外部メモリの任意の変数の内容をC言語から直接リードライトできるライブラリを開発し，プロセッサデータの初期値設定や，デバッグに用いている．さらに，配線状況を視覚的に捉えられるようにMAPLE-モニタを開発した．これはXウィンドウシステムでインプリメントした．

5. まとめ

MAPLEのソフトウェア開発環境として，マイクロコンパイラ，入出力ライブラリ，モニタの開発を行った．この環境下で，約7Kステップの配線ソフトウェアのデバッグを約1ヵ月で終了した．

今後は，EWS上でＭＡＰＬＥをシミュレートできるシミュレータの開発，UNIXのdbxに相当するデバッガの開発，並列動作の分析，デバッグ，チューニングを支援するビジュアライザーの開発などが必要であると考えられる．

```
struct costState {
   costT  cross;
   costT  touch;
}
static    costState    $cost;
static    bit          $currentCost;
static    bit          $newCost;
static    bit          $northCost;
static    bit          $inhibitFlag;
static    bit          $targetFlag;
mem       bit          lineWidth[5];
global    bit          globalTarget[16];
global    bit          currentTarget[16];
```

図2　マイクロ記述言語の変数宣言

```
label()
{
 C言語
 {{
  $newCost=ADD($cost.cross,$cost.touch); (1)        a
  $newCost = ADD($newCost, (North($currentCost)); b

  CF(INV($inhibitFlag)) { (2)
  $currentCost=MIN:PASS2($newCost,$currentCost);
  }
  EVENT=A.LT.B; (3)

  CF ($targetFlag)   {
   globalTarget=GMIN[PASS:ONE($currentCost)];
  }    (4)

  sync(); (5)
  }}

  waitPE();

  if (globalTarget < currentTarget)
   currentCost=globalTarget;

 }
```

図3　プログラム記述例