数理解析研究所講究録
992巻 1997年 122-125

CCSに基づく並列処理言語の実装

九州大学システム情報科学研究科　原 淳　(Atsushi Hara)
九州大学システム情報科学研究科　森 雅生　(Masao Mori)

1　はじめに

ここ数年の間に並列コンピュータや並列／分散処理機構を備えた計算機ネットワーク (これらをまとめて 並列／分散コンピュータ と呼ぶ) が目覚しく発達してきた.

それを用いて1台のコンピュータでは処理しきれないような大きいデータを, ほとんど通常のプログラム言語を使っているのと変わらないような感覚で分散/並列計算を行なうプログラムを記述することを目的として,CCS[2] が提案している並列プログラム言語の意味論を基礎に改良した並列分散プログラム言語を定義し, 実装した.

2　言語の定義

言語は,C 言語や PASCAL 言語風の手続き型言語に並列演算子 (**par**) を付け加えたものとし, 文法を以下のように定義した.

X	$::=$	$X \vert Y \vert \cdots$	変数
F	$::=$	$+ \vert - \vert \cdots \vert 0 \vert 1 \vert \cdots$	関数
C	$::=$	$C;C'$	直列合成
		$X := E$	代入
		$\mathbf{if}(E)\mathbf{then}\{D_v;C\}\mathbf{else}\{D'_v;C'\}$	条件分岐
		$\mathbf{while}(E)\mathbf{do}\{D_v;C\}$	繰り返し
		$\{D_v;C\}\mathbf{par}\{D'_v;C'\}$	並列合成
		$\mathbf{call}\ \ \mathbf{G}(E,Z)$	手続き呼び出し
		$\mathbf{input}\ \mathbf{X}$	入力
		$\mathbf{output}\ \mathbf{E}$	出力
D_v	$::=$	$D_v;D'_v$	
		$\mathbf{var}\ \ \mathbf{X}$	変数宣言
D_p	$::=$	$D_p;D'_p$	
		$\mathbf{proc}\ \ \mathbf{G}(in\ X;out\ Y)\mathbf{is}\ \{D_v;C\}$	手続き
E	$::=$	X	変数
		$F(E_1,E_2,\cdots,E_n)$	関数適用

定義した言語を実装するために, 並列言語から CCS の言語への変換の関数 $M[-]$ を次のように定義した.

$$
\begin{array}{lcl}
M[C;C] & = & M[C]BeforeM[C'] \\
M[X := E] & = & \overline{up}_X.M[E]Into(x)(\overline{put_X}x.\overline{down}_X.Done) \\
M[\textbf{if}(E)\textbf{then}\{D_v;C\}\textbf{else}\{D_v';C'\}] & = & M[E]Into(x)(\textbf{if}x\textbf{then}M[D_v;C]\textbf{else}M[D_v';C']) \\
M[\textbf{while}(E)\textbf{do}\{D_v;C\}] & = & W\ ,\ where\ W \stackrel{def}{=} M[E]Into(x) \\
 & & (\textbf{if}x\textbf{then}M[D_v;C]\ Before\ W\textbf{else}\ done) \\
M[\{D_v;C\}\textbf{par}\{D_v';C'\}] & = & M[D_v;C]ParM[D_v';C'] \\
M[\textbf{input X}] & = & \overline{up}_X.in\ x.\overline{put_X}x.\overline{down}_X.Done \\
M[\textbf{output E}] & = & M[E]Into(x)(\overline{out}x.Done) \\
M[\{D_v;C\}] & = & (M[D_v]|M[C])\backslash ACC_{D_v} \\
 & & \\
M[\textbf{var X}] & = & Sem_X|Loc_X \\
 & & \\
M[\textbf{call G}(E,Z)] & = & M[E]Into(x).(name_G g.\overline{call_{G,g}}x.return_{G,g}z.\overline{put_Z}z.Done) \\
M[\textbf{proc G}(inX,outY)\ \textbf{is}\ D_v;C] & = & W(0)\qquad ,where \\
\qquad W(g) & \stackrel{def}{=} & \overline{name_G}g.(G_g|W(g+1)) \\
\qquad G_g & \stackrel{def}{=} & call_{G,g}x.(Loc_X|Loc_Y|\overline{put_X}x.M[D_v;C].get_Y y.\overline{return_{G,g}}y.0)\backslash L_X\backslash L_Y \\
M[X] & = & get_X x.\overline{res}x.0 \\
M[F(E_1,\cdots,E_n)] & = & (M[E_1][arg_1/res]|\cdots|M[E_n][arg_n/res] \\
 & & |M[F])\backslash\{arg_1,\cdots,arg_n\} \\
M[F] & = & arg_1x_1.\cdots.arg_nx_n.\overline{res}(f(x_1,\cdots,x_n)).0
\end{array}
$$

ここで,

$$
\begin{array}{rcl}
Done & \stackrel{def}{=} & \overline{done}.0 \\
P\ Before\ Q & \stackrel{def}{=} & (P[b/done]|b.Q)\backslash b \\
P\ Par\ Q & \stackrel{def}{=} & (P[d_1/done]|Q[d_2/done]| \\
 & & (d_1.d_2.Done + d_2.d_1.Done))\backslash\{d_1,d_2\}) \\
P\ Into(x)Q & \stackrel{def}{=} & (P\ |res(x).Q)\backslash res
\end{array}
$$

この変換関数は，文献 [2] で定義されているものを基礎に改良したものになっている．特に，手続き呼び出しに関しては，実装時の効率を考えて大幅に変更を行なった．

手続き起動プロセス

文献 [2] で行なわれている定義では, 手続きプロセスは常に 1 個以上が呼び出し待ち状態にあるため, メモリの使用効率が落ち, また手続きプロセスの生成, 消滅毎に変数プロセス以外の全てのプロセスに呼び出し可能なプロセスのプロセス ID を通知しなければならず不必要な通信が行われるため, それぞれの手続きプロセスについて手続き起動プロセスを作り，手続き呼び出しはそれを通して行なうようにした．

この場合，手続き呼び出しは呼出側が，手続き起動プロセスと通信して，新たに生成された手続きプロセスの番号を受けとり，この番号を用いて手続きの呼び出しを行なうようになっている．

手続き起動プロセスは, 式の $W(n)$ にあたる．

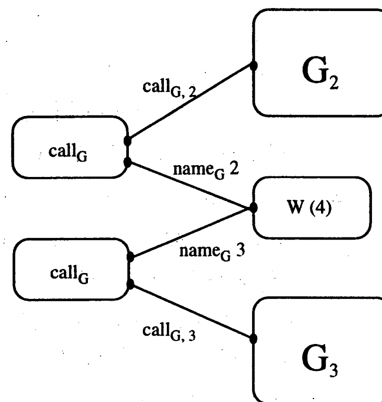

図 1: 手続き起動プロセス

3 言語の実装

2節で定義した言語の仕様に基づいて実装を行った.

コンパイラは, 並列言語のソースプログラムからC言語への変換プログラムと, 変換されたプログラムをコンパイルするときに必要となる通信関係のライブラリからなっている.

また, 作成したプログラムで共通して使用するプロセスの作成も行った.

C言語への変換には Perl を, 通信には PVM を用いている.

3.1 PVM の概要

PVM(Parallel Virtual Macine)[1] は, ネットワークに接続された異機種UNIXコンピュータ群を, 単一の並列コンピュータとして利用することを可能とするソフトウェアシステムである.

PVM のシステムの構成は, 以下の2つに分けられる.

デーモン 分散メモリ型並列コンピュータを構成する全てのコンピュータ上に常駐し, 実際の通信等を行なう.

ライブラリ PVM が提供する機能へのインタフェイスルーチンのライブラリであり,PVM を利用するアプリケーションは, このライブラリをリンクする必要がある.

ライブラリへの言語インタフェイスとしては,C 言語と Fortran 言語が用意されており, 今回はC言語を用いた.

PVM では, 各プロセスにタスク ID と呼ばれる番号がついており, プロセス間の通信では, これを用いて相手の指定をする.

言語の実装には,PVM により提供される機能のうち, プロセスの生成/消滅, プロセス間の通信の機能を利用している.

3.2 実装にともなう変更

実装では, 前節の定義では不都合な部分が出てきたため, 一部変更を行ない, 新たにプログラム起動／終了プロセスを追加した.

プログラム起動／終了プロセス

プログラム全体は，標準入出力を行なうプロセスを $Stdio$ として，

$$M[prog] = M[D_v; D_p; C] = M[D_v] \mid M[D_p] \mid M[C] \mid Stdio$$

と書くことができるが，ここまでの定義のままでは，プログラムを終了したとき，Cの部分のプロセスは終了して消えるが，D_v, D_pの部分のプロセスは，終了しない．このため，プログラム全体を終了させるために，Cが終了した時点で全てのプロセスを終了させる部分を追加して，

$$M[prog] = M[D_v; D_p; C] = M[D_v] \mid M[D_p] \mid M[C]\ Before\ Killall \mid Stdio$$

とした．

この，残ったプロセスを終了させるプロセスと，標準入出力の部分をまとめてプログラム起動プロセスとした．このプロセスはそれ以外に，全域変数のプロセス，手続きのプロセス，プログラム本体のプロセスを起動と，起動した各プロセスへのタスク ID の通知をおこなっている．

4 まとめ

本論分では,CCS が提案している並列プログラム言語の意味論を基礎に改良し並列プログラム言語を定義, 実装した.

実装した言語は, 最低限の機能しか持たないものであり, また,1 変数を扱うにも百数キロバイトものメモリを消費し, 手続きの再帰呼び出しを行なうときに, 毎回同じプロセスをたちあげるなど非常に効率の悪いものであった.

今後は, 変数に関しては, プロセスローカルに扱えるものは変数プロセスを用いないようにして速度, メモリの効率を向上し, 非ローカルな変数に関しても,1CPU につき 1 個のプロセスで変数を管理するような, 変数マネージャープロセスを用いることによってメモリ効率の向上を行ないたい. さらに, 配列や, 実数変数の実装も行ないたい.

参考文献

[1] Al Geist et al,"PVM:Parallel Virtual Macine",The MIT Press, 1994.

[2] Robin Milner,"Communication and Cocurrency",Prentice Hall, 1989.