一种基于C扩展的SIMD的并行程序设计语言*

景晓军 方滨兴

(哈尔滨工业大学计算机系 哈尔滨 150001)

摘要 SIMC(SIMD C)是通过对C语言进行语法扩展(未进行语义扩展)得到的支持SIMD (single instruction multiple data)并行程序设计的并行语言.SIMC可方便地描述SIMD并行算法,具有SIMD计算机系统结构定义能力,可支持多种系统结构上的并行算法研究.SIMC语言的模拟执行系统已在单机上实现,并作为作者研究开发的SIMD计算机程序设计及性能评价模拟环境的并行程序设计语言,用于SIMD计算机算法及结构的性能评价.

关键词 SIMD,并行程序设计,并行程序设计语言,并行计算机性能评价,模拟.

SIMD(single instruction multiple data)计算机从构型上来说有2种.[1]一种是处理单元(PE)到处理单元的连接形式,又被称为分布式局部存储(Distributed Local Memory)结构.在这种结构下,PE是由CPU和局部存储器(Local Memory)组成,PE间通过互连网络相互连接,完成数据通讯.这种构型在已实现的SIMD机器中占大多数.另一种是处理单元到存储器的连接形式.这种机器中,PE不含局部存储器,而是若干存储器模块通过对准网络与PE相连,使得各存储模块为各PE共享.这种结构又称作共享存储(Shared Memory)结构的SIMD计算机.

尽管与MIMD计算机相比较,SIMD计算机在求解问题上有一定的局限性,但目前它仍是较成功的商业化产品.Fox[2]研究表明有46%的已被成功并行化的实际应用问题是严格同步问题,这些严格同步问题比较适合于在SIMD计算机上求解.作为并行算法设计中重要的一类,有许多影响SIMD机上并行程序效率的复杂因素需要研究,并且SIMD计算机比较专用,算法效率与系统结构的联系比较紧密.所以,开发可运行SIMD并行程序的模拟系统为研究者提供一个仿真的实验环境,避免过多的"纸上谈兵"是很有意义的,这一点对缺乏实际并行系统的研究单位尤为重要.国际上目前也有许多关于SIMD计算机的模拟系统在开发和利用之中.[3~5]

在开发模拟系统时,确定其使用的并行程序设计语言是系统成功的关键.模拟系统所用语言应当既便于使用,又有足够的程序设计能力,同时应当便于编译实现.

---

* 作者景晓军,1967年生,博士,主要研究领域为并行机系统结构,并行机性能评价,模拟,并行程序设计语言.方滨兴,1960年生,教授,主要研究领域为并行处理,SIMD计算机,计算机安全.

本文通讯联系人:景晓军,哈尔滨150001,哈尔滨工业大学计算机系

本文1995-05-22收到修改稿

SIMC(SIMD C)是作者实现的SIMD计算机程序及系统结构性能模拟环境的并行程序设计语言.该语言是基于标准C语言的扩展.SIMC中,通过定义新的向量数据类型变量,引入新的并行语句完成对向量数据类型变量的并行操作.关于该语言扩展部分的完备性,主要参考了Rice的研究工作.[6]

1 SIMC语言

在SIMC中,有2类扩展语句.一类是说明语句的扩展,主要完成系统结构和向量数据类型变量的定义.另一种是可执行语句的扩展,主要是在CU或PE中完成的对向量进行的并行操作.

1.1 扩展说明语句

在SIMC中,说明语句包括标量说明、并行变量说明、混合变量说明、屏蔽变量说明和系统结构说明.标量说明定义了标量变量,这类变量只在CU中可访问.其说明方法、格式以及所允许的数据类型与标准C语言中相同(形式描述省略).除此之外的其它变量说明都是扩展说明语句.SIMC并行程序结构形式与标准C语言相同,在此只对扩展说明语句的语法用BNF加以描述.

其中"//"之后是解释、说明文字;黑体词是新定义的系统内部保留字;有关C语言中变量说明形式在此省略.

〈扩展说明〉::=〈结构说明〉〈并行变量说明〉〈混合变量说明〉〈屏蔽变量说明〉

〈结构说明〉::=architecture{pe_number=〈正整数〉;sp_memory=〈存储容量说明〉;pe_memory=〈存储容量说明〉;network=〈网络名〉;mask=〈屏蔽机制〉;}

〈存储容量说明〉::=〈正整数〉〈存储单位〉

〈存储单位〉::=k | K | m | M

〈并行变量说明〉::=pe_var{〈变量定义〉}

〈混合变量说明〉::=cu_var{〈变量定义〉}

〈屏蔽变量说明〉::=mask_var{〈屏蔽寄存器定义〉|〈屏蔽变量定义〉}

〈屏蔽寄存器定义〉::=mask_REG 〈变量名〉

〈屏蔽变量定义〉::=int 〈变量名〉

〈屏蔽机制〉::=register | address

〈网络名〉::= //〈用户定义的互联网络名〉

〈变量定义〉::= // 变量定义的方法与C语言中的相同,只是允许的数据类型是"int,float,char,double"以及相应的修饰"unsigned,short,long"

〈变量名〉::= //C语言中允许的标识符(省略)

表1是各扩展说明语句的实例.

并行变量:在PE中才能访问的变量,只出现在并行指令中.

混合变量:是在CU和PE中都可以访问的变量,在标量指令和并行指令中都可出现.主要用于存储公共数据,同时也可存放从PE收集的数据.其变量类型限制与PE变量类型相同.

屏蔽变量:用于控制屏蔽状态寄存器的变化,说明活动的PE集合.变量类型有2种:"REG_mask"和"int".2种类型对应于不同的屏蔽机制,"mask_REG"定义寄存器屏蔽机制下CU中的屏蔽寄存器变量;"int"定义地址屏蔽机制下的屏蔽控制变量.

表1 扩展说明语句实例

名称	体系结构定义	并行变量定义	混合变量定义	屏蔽变量定义

实例	architecture {
	pe_number=16;
	cu_memory=2k;
	pe_memory=1k;
	network=hypercube;
	mask=address;}
		pe_var{
		int A,B;
		float C[10];
		}
			cu_var {
			int cu,fg;
			float aa[10];
			}
				mask_var{
				int mk;
				}或
				mask_var {
				REG_mask mk;
				}

1.1.1 系统结构说明

SIMC语言的程序设计是基于一个虚拟的SIMD计算机,程序设计者通过语言中的系统结构说明来限定虚拟机的一些系统参数来模拟某一特定结构的SIMD计算机.该虚拟机的结构如图1.

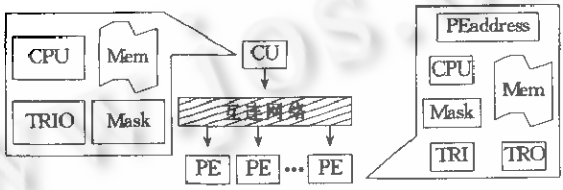

图1 SIMD虚拟机结构

该虚拟机主要由控制部件CU(control unit),互连网络和处理单元PE(processing element)组成.系统中PE带有局部存储器,PE之间通过互连网络相互连接.其中CU和PE中的ALU可完成C语言中的任何运算操作.通过结构说明可定义CU和PE中的存储器(Mem)的大小;PE中还有一个屏蔽寄存器(Mask),用于控制该PE是否活动;一个PE地址寄存器(PEaddress)用于索引该处理器;数据传输输入(TRI)及输出(TRO)寄存器,用于PE间以及PE和CU间的数据传输.CU中有屏蔽寄存器(Mask,大小为N-bits,N是PE个数),此寄存器的作用由结构说明语句中的屏蔽机制决定;数据传输寄存器TRIO,用于CU与PE间的通讯.互连网络可由用户定义嵌入.这种虚拟机结构可以描述分布式局部存储结构类型的SIMD计算机的特点,有较广泛的适用性.

按照Siegel提出的SIMD计算机的数学模型[7],1台SIMD机器形式上可以表示成一个四元组(N,F,I,M),其中N表示SIMD机器中PE数目;F是互连函数集;I是一组机器指令;M是一组屏蔽方式.这个模型概括了决定一个SIMD计算机系统的要素.在SIMC中,通过说明虚拟机的这些参数,来定义一个SIMD计算机的结构.

SIMD计算机屏蔽机制一般分为2类,屏蔽寄存器控制机制和地址屏蔽控制机制[7],关于2种屏蔽机制在此不赘述.SIMC中,允许选择上述之一作为屏蔽机制,控制处理器的活动.当"mask=register"时,表示系统采用寄存器屏蔽机制;当"mask=address"时,表示系统采用地址屏蔽机制.

1.1.2 互连网络的定义

互连网络是决定SIMD计算机拓扑结构最重要的因素之一,直接影响着问题求解的效率.在SIMC中,用户可说明程序中使用的互连网络名称.可用C语言对欲使用的互连网络的置换函数加以描述,然后在程序模拟执行中使用定义的互连网络.这样,用户可根据需要

描述自己设计的特殊互连网络,并在模拟执行中加以使用.从而可对在不同互连结构下求解问题的程序性能进行比较,选择适合于问题的互连结构.

在系统中,使用C语言对互连函数描述的格式如下:

int NetworkName(int Direction,int PEaddress)

其中,Direction入口参数控制互连网络的传输方向,PEaddress表示通讯源PE地址,函数的返回值是通讯的目的PE地址.

例如:对ILLIAC IV机器的互联网络,用上述定义形式可描述如下:

ILLIAC IV是由64个处理器以64*64的mesh结构排列的,其置换函数为

```
int ILLIACNetWork(int control,int node)
{ switch(control){
  case 0: /* up */
  return((node<64)? (node+4031):(node-64));
  case 1: /* down */
  return((node>4031)? (node-4032):(node+64));
  case 2: /* left */
  return((node==4095)? 0:(node+1));
  case 3: /* right */
  return((node==0)? 4095:(node-1));
}}
```

在SIMC语言中使用互连网络时,需在远程传输语句中说明调用互连函数的传输方向,即说明函数定义中的Direction参数值,如$a[net(i)]:==a+b$;语句中,"i"变量就是调用互连函数时的Direction形参的入口参数.

1.2 可执行语句的扩展

在SIMC中,可执行语句包括标量操作语句和并行操作语句.标量操作语句只在CU中执行,对公共的数据进行操作并且控制程序流,这部分指令集与标准C语言的语句相同.向量指令在PE或CU内执行,对分布在各PE内的数据进行操作,还完成PE与CU之间数据相互传送,这部分是扩展指令,对扩展语句的BNF范式表示如下:

```
〈并行操作〉::=〈并行指令〉; | 〈并行指令〉,〈地址屏蔽表达式〉;
〈并行指令〉::=〈非条件并行指令〉|〈条件并行指令〉|〈屏蔽指令〉
〈非条件并行指令〉::=〈内部赋值语句〉|〈远程赋值语句〉|〈广播语句〉|〈收集语句〉
〈内部赋值语句〉::=〈并行变量〉:=〈并行表达式〉
〈远程赋值语句〉::=〈并行变量〉[net(〈传输方向控制〉)]:==〈并行表达式〉
〈广播语句〉::=〈并行变量〉:<==〈混合表达式〉
〈收集语句〉::=〈混合变量〉 <-- 〈并行表达式〉
〈条件并行指令〉::=where〈并行逻辑表达式〉do {〈非条件并行指令〉}
                  | where〈并行逻辑表达式〉do {〈非条件并行指令〉}
                  elsewhere{ 〈非条件并行指令〉 }
〈屏蔽指令〉::=〈屏蔽广播语句〉 | 〈屏蔽收集语句〉
〈屏蔽广播语句〉::=〈屏蔽寄存器变量〉 ::= 〈屏蔽表达式〉
〈屏蔽收集语句〉::=〈屏蔽寄存器变量〉::;
〈地址屏蔽表达式〉::=[〈位域赋值〉;〈位域赋值〉|〈位域赋值〉]
〈位域赋值〉::=〈位域表达式〉:〈位域表达式〉=〈屏蔽值〉
〈并行表达式〉::= // 表达式中出现的变量都是并行变量的C语言表达式
〈混合表达式〉::= // 表达式中出现的变量全部是混合变量的C语言表达式
```

〈并行逻辑表达式〉::=// 表达式中出现的变量都是并行变量的 C 语言表达式

〈位域表达式〉::=//表达式中出现的变量都是屏蔽变量的 C 语言表达式

〈并行变量〉::=//并行变量说明中定义的变量

〈混合变量〉::=//混合变量说明中定义的变量

〈屏蔽变量〉::=//屏蔽变量说明中定义的地址屏蔽变量

〈屏蔽寄存器变量〉::=//屏蔽说明语句定义的屏蔽寄存器变量

〈屏蔽值〉::= 1 | 0 | *

关于 C 语言表达式的描述在此省略.

表 2 是对各扩展并行语句语义的描述.对扩展语句语义描述是基于图 1 所示的 SIMD 抽象机,其中 TRI,TRO,Pmask,TRIO 分别对应于抽象机中 PE 的传输输入、传输输出、屏蔽寄存器和 CU 中的数据传输寄存器.描述中出现的各种运算符含义同 C 语言中的运算符;Pvar,Cvar 分别表示定义的并行变量和混合变量;Pexp,Cexp,Mexp 分别表示并行表达式,混合表达式和屏蔽表达式;Mval 表示屏蔽值;Active 表示置屏蔽寄存器为活动;PEnum 表示系统中 PE 的个数;Network 表示按照前文定义的互连函数;$Pvar_k=Pexp_k$ 表示系统内第 k 个 PE 内在完成约定的运算. k. range 表示 k 的 range 所限定位的取值. range 代表位操作的范围.

表 2 扩展并行指令语义说明

名称	语义说明	示例
内部赋值: *Pvar*:=*Pexp*,*Mexp*;	If (*Pmask*$_k$==*Active*) 　*Pvar*$_k$=*Pexp*$_k$;	*a*:=*b*+*c*;

远程赋值:
Pvar < *net* (*dir*) > = *Pexp*,
Mexp;
	TRO$_k$=*Pexp*$_k$;
	for(*i*=0;*i*<*PEnum*;*i*++)
	　TRI[*Network*(*dir*,*i*)]=*TRO*[*i*]
	If(*Pmaskk*==*Active*)
	　Pvar$_k$=*TRI*$_k$
		a[*net*(*i*)]:==*a*+*b*;

广播指令: *Pvar*<==*Cexp*,*Mexp*;	If(*Pmask*$_k$==*Active*) { *TRIO*=*Cexp*; *TRI*$_k$=*TRIO*; 　*Pvar*$_k$=*TRI*$_k$; }	*pe*<==*c*1+*c*2;

收集指令:
Cvar<−−*Pexp*,*Mexp*;
	If(*Pmaskk*==*Active*)
	{ *ActCount*++; *TRO*$_k$=*Pexp*$_k$;
	　TRIO=*TROk*; *Cvar*[*k*+1]=*TRIO*;
	}
	Cvar[0]=*ActCount*;
		cu<−−*a*+*b*

条件指令:
where (*Condiction*){
　Inst−*Set*1;
}
elsewhere{
　Inst−*Set*2;
}
	IF(*Condiction*$_k$==*TRUE*)
	　Pmask$_k$=*Active*;
	IF(*Pmask*$_k$=*Active*)
	　{*Inst*−*Set*1$_k$; }
	IF(*Condiction*$_k$==*FALSE*)
	　Pmask$_k$=*Active*;
	IF(*Pmask*$_k$=*Active*)
	　{*Inst*−*Set*2$_k$; }
		where (*a*==*b*) do
		{ *true*−*body*}
		elsewhere
		{ *false*−*body* }

屏蔽控制(*Mexp*): [*range*$_1$=*Mval*$_1$, *range*$_2$=*Mval*$_2$, ..., *range*$_n$=*Mval*$_n$]	IF(*k*. *range*$_1$==*Mval*$_1$&& 　*k*. *range*$_2$==*Mval*$_2$ 　&&...&&*k*. *range*$_n$==*Mval*$_n$) 　*Pmask*$_k$=*Active*;	[*m*1:4=1,4:0=0]

收集指令中,左变量必须是一个长度为 $N+1$(N 为 PE 数)的一维数组的混合变量,收集来的所有活动的各 PE 值将被依次存放在数组下标从 1 开始的单元中,0 号单元将存放收集的数据的个数. 表 2 中 ActCount 完成收集数据数目的统计.

2 SIMC 编程实例

排序问题是以比较和传送操作为主的典型非数值计算问题. 在计算机科学中,对排序问题有许多深入的研究. 本文作为应用实例,实现了立方体网互连结构 SIMD 机器上的 Batcher 双调排序算法.[8]算法处理的排序数据规模与处理器的个数相同,满足 $N=2M$,其中 M 是立方体互连网的维数. 双调排序的 *SIMC* 程序如下:

```
#include (stdio.h)
#include (math.h)
#define M 8              /* 待排序数的个数 */
#define BitNum 3         /* 处理器地址二进制表示的位数 */
architecture{            /* 结构说明 */
  pe_number=8;           /* 处理器个数为 8 */
  pe_memory=1k;          /* PE 内存大小为 1k */
  cu_memory=1k;          /* CU 内存大小为 1k */
  network=hypercube;     /* 设置立方体网络 */
  mask=address;}         /* 采用地址屏蔽机制 */
pe_var{int A,B,P_DTR,ADDRESS,P_j,P_k,P_i;}        /* 并行变量定义 */
cu_var{int OUT[9],X[8];}        /* 混合变量定义 */
mask_var{int mv;}        /* 地址屏蔽变量定义 */
main(void)
{  int i,j,k;
   Initdata();        /*产生随机数作为排序初值,并将各数分配到处理器中,该函数省略*/
for(j=1;j<=BitNum;j++)        /*排序开始*/
{  for(i=j-1;i>=0;i--)
   {  mv=i;
      P_i:<==i;
      A:=P_DTR,[mv=0];
      P_DTR(net(i)):==P_DTR,[mv=0];
      /* 决定升序、降序排列预处理,计算 ADDRESS(m-1)^...^ADDRESS(j) */
      ADDRESS:=0;
      P_j:<==j;
      for(k=BitNum-1;k>=j;k--)
      {  P_k:<==k;
         ADDRESS:=ADDRESS^((#>>P_k)&1);        /*#处理器号索引*/
      }
      where(((P_j==3)||(ADDRESS==0))&&(((#>>P_i)&1)==0)&&(P_DTR<A)) do
      {B:=A;        /* 升序排列 */
       A:=P_DTR;
       P_DTR:=B;
      }
      where(((P_j==3)&&(ADDRESS==1))&&(((#>>P_i)&1)==0)&&(P_DTR>A)) do
      {B:=A;        /* 降序排列 */
       A:=P_DTR;
       P_DTR:=B;
      }
      P_DTR<net(i)>:==P_DTR,[mv=1];
      P_DTR:=A,[mv=0];
   }} /* 排序结束 */
```

```
    CollectAndPrint(); /* 数据收集及输出函数省略 */
}
```

3 结 论

现阶段并行程序设计语言种类很多,主要偏重于数据并行.这些语言多是在已有串行计算机程序设计语言(如 Fortran,Lisp,Pascal,C 等)基础上进行扩展得到的.如 Actus[9]是基于 Pascal 的扩展;Vector C[10],C*[11]是基于 C 语言的扩展;Fortran D[12]是基于 Fortran 的扩展.这些语言往往通过引入向量数据类型定义,修改在原语言中串行操作语句的语义来实现对向量的操作.这些语言多用于实际的并行机系统中,其编译方面的工作比较复杂.

SIMC 语言的定义主要借鉴了 ACLE—SIMD 并行程序设计模拟系统[4]的语言 ACLAN,增强了结构说明能力,减少了扩展并行语句数目.这种扩展方法一方面增强了系统结构说明能力,另一方面使得开发该语言编译器的工作量大大减少.SIMC 具有如下特点:

(1)在保留了 C 语言功能强,可移植性好的特点的基础上,对已有的标准 C 语言中的说明语句及操作语句进行少量的语句扩展来实现数据的并行操作.这种方法使 SIMC 既便于使用又可提高预编译器的实现效率.

(2)独立于 SIMD 计算机的系统结构.程序设计者在使用 SIMC 时,是基于一个 SIMD 虚拟机来编写并行程序,而不考虑具体的 SIMD 机器的特点.

(3)能够定义 SIMD 体系结构.由于该语言是基于一个 SIMD 虚拟机,故在该语言中可通过限定该虚拟机的一些参数来达到在较高层次上限定 SIMD 的结构,并可在其上模拟求解问题.

基于上述特点,SIMC 语言可用于算法研究和 SIMD 结构的性能评价的研究.

目前 SIMC 程序的模拟执行环境已开发成功,该环境中还具有图形可视化的程序性能分析系统.研究者可在其上进行 SIMD 并行算法及结构的性能评价方面的研究.

参考文献

1 黄铠,F. A. 布里格斯. 计算机结构与并行处理. 金兰等编译,北京:科学出版社,1990.

2 Fox G C. What have we learnt from using real parallel machines to solve real problems. *Proceeding of the 3rd CHCCA*, 1988. 21～25.

3 Michael M, Gutzmann *et al*. PEPSIM－ST: a simulator tool for benchmarking. COMCON'93, 1993. 665～676.

4 Plata O G *et al*. ACLE: a software package for SIMD computer simulation. The Computer Journal, 1990,**33(3)**: 194～203.

5 Herbordt M C, Weems C C. An environment for evaluating architecture for spatially mapped computation system and initial results. COMPCON'93, 1993, 191－201Caltech, Report C2p－522, 1990.

6 Rice M D. Semantic for data parallel computation. International Journal of Parallel Programming, 1990,**19(6)**:477～509.

7 Siegel H J. A model of SIMD machine and comparison of various interconnection network. IEEE Trans. Computer, Dec. 1979, C－28,907～917.

8 陈国良. 并行算法——排序和选择. 合肥:中国科技大学出版社,1990.

9 Perrott R H *et al*. A language for array and vector processors. ACM Transactions on Programming Languages and

Systems, Oct. 1979. 177～195.

10 Li K—C *et al*. Vector C: a vector processing language. Journal of Parallel and Distributed Computing, May 1985. 132～169.

11 Michael Hord R. Parallel supercomputing in SIMD architectures. CRC Press, Inc. 1990.

12 Fox G C *et al*. Fortran D language specification. Technical Report TR90—141, Dept. of Computer Science, Rice University, December 1990.

EXTENSIONS TO THE C PROGRAMMING LANGUAGE FOR SIMD COMPUTERS

Jing Xiaojun Fang Binxing

(*Department of Computer Science and Engineering Harbin Institute of Technology Harbin* 150001)

Abstract SIMC(SIMD C) is a SIMD(single instruction multiple data) parallel programming language extended to the syntax of usual C language. SIMC can be easy to program SIMD algorithms. Moreover, it can be used to study SIMD algorithms on different kinds of SIMD architecture which is defined in SIMC. The simulation environment of SIMC has been implemented on a sequential computer. This environment can be used to evaluate the performance of SIMD's algorithms and architecture.

Key words SIMD, parallel programming, parallel programming language, performance evaluation, simulation.