国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第11卷第4期 1989年12月 Vol.11 No.4

顺序PROLOG机存储组织研究

李 艮 艮

（电子计算机系）

摘 要 YH—SIM是一种正在研制中的顺序PROLOG机。本文介绍该机存储组织的设计考虑[9,11,12]。根据PROLOG过程执行的具体特点，文中提出了一种多专用 Cache 的存储子系统结构，旨在支持深度优先加回溯的顺序PROLOG求解机制。即按照不同的访问方式，设立常规Cache和栈式 Cache，分别支持存储空间中的随机访问区域和栈式访问区域。作为一个典型的子部分，文中详细讨论了选择点Cache的控制和调度策略。

关键词：PROLOG机，存储组织，多重CACHE

分类号：TP33

1 顺序PROLOG机硬件组织的研究概述

硬件组织的研究，在于寻求有效支持系统结构的硬件和固件。与其它语言相比，PROLOG过程在具体执行中有如下特点[1,2,10]：

（1）数据类型的不确定性，以及以参数匹配作为参数传递手段；

（2）过程调用的不确定性，即用试探和回溯作为执行过程的控制手段，故而要求大量存储调用环境；

（3）求解过程的深度递归和嵌套，主要地规定了指令和数据访存特性。

相应地，对硬件组织的设计有如下要求[3,4,6]：

（1）在执行部件中，除了传统的ALU之外，要求增加支持一致化操作的快速类型匹配和测试部件；

（2）提供大容量的寄存器文件，支持硬件栈、回溯和跟踪操作；

（3）在控制部件中，加强指令预取；支持多种形式的多分支转移，支持丰富的 PROLOG内部过程；

（4）在存储部件方面，提供大容量存储空间；采用多种途径提高存储效率和频带。

本文着重于解决 PROLOG 执行过程中的访存瓶颈问题。作者研究了一种多专用Cache的存储子系统结构，旨在支持深度优先加回溯的顺序 PROLOG求解机制。在YH-SIM的存储子系统中，按照不同的访问方式，设立了常规Cache和栈式Cache，分别支持

国家自然科学基金资助项目

学术论文 1989年3月30日收稿

存储空间中的随机访问区域和栈式访问区域。作为一个典型的子部分，文中详细讨论了选择点Cache的控制和调度策略。

作者假定读者一般地了解顺序PROLOG的实现技术，如WAM模型[1,6,8,12]。

2 YH-SIM的硬件总体结构

YH-SIM 硬件的总体结构[12]主要有三个模块，即控制部件 CU、执行部件 EU 和存储部件MU。整个CPU 按流水线方式工作。CU 采用微程序控制方式，具有控制各种转移和多路分叉功能力。EU 包括寄存器文件、草稿栈（用于复合结构的一致化）、访存接口和运算部件。MU 采用分布式层次结构，其中、CPB、TRB、DB 和 IB 在功能和物理上都是独立和专用的，分别作为顺序PROLOG机存储空间的几个功能区（代码区、跟踪栈TRAIL、全局数据栈HEAP、以及局部栈STACK）的缓冲存储器。MCP 是一个独立的处理器，负责整个存储子系统的控制，完成主存与Cache间的自动调度和协调。

关于EU和CU的设计方案和细节，见诸于许多文献和报告，如文[3]、[4]、[5]、[6]。本文着重介绍存储控制器MCP的控制算法[12]。

3 一种多专用Cache的存储子系统结构

3.1 研究Cache的必要性

Cache方法是计算机系统结构设计中的重要技术之一。设立Cache主要取决于如下几个因素：

（1）访存的频度（高）；

（2）存储访问周期（长、$T_{mm} \gg T_{cpu}$）；

（3）存储访问冲突（仲裁）。

Benchmarks	Tak	Boyer	Deriv	Puzzle
访问数据/指令	0.63	1.68	1.93	1.65
访问指令/指令	1.16	1.16	1.34	1.22

图 1 几个典型PROLOG程序的动态访存频度(次/指令)[7]

根据对指令动态执行的统计（见图1），PROLOG程序的访存频度相当高，平均一条指令访存2至3次（读指令、读写数据）。其次，PROLOG执行的空间开销很大，现已发表的PROLOG模型机，其主存容量均在16M字以上，与超级计算机相当。在合理的性能价格比之下，较大的存储容量意味着较大的主存周期与主机周期之差异，即$T_{mm} \gg T_{cpu}$. 再者，即便在访存速度较快的情况下，仍然可能需要设置Cache. 例如在采用总线结构的 VAX—11/780 中，$T_{mm}=4*T_{cpu}$，但仍然设置了指令Cache和数据Cache，旨在避免因总线冲突而导致的仲裁开销。

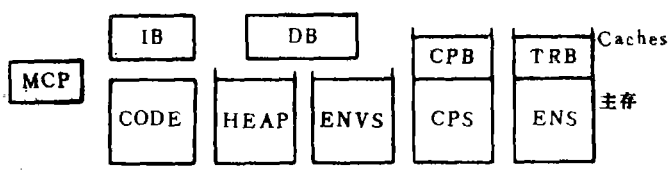

图 2 YH-SIM硬件组织中的存储子系统结构

3.2 访问方式与Cache的结构

Cache的有效性与Cache的命中率紧密相关，Cache的命中率又取决于被缓冲对象的局部性和所采用的调度算法。首先讨论顺序PROLOG机中存储空间的访问方式，及其存

储对象的分布特征，从而确定有效的Cache结构形式和调度算法。

YH-SIM存储空间中的诸功能区域的访问方式，主要分两类[8,12]。

（1）栈访问方式

跟踪栈TRAIL完全按栈方式访问，总是访问 TRAIL 栈顶元素；若以选择点为基本访问单位，则选择点的访问也是按栈方式进行的，即通过选择点寄存器 B 访问当前选择点。按栈方式访问的区域，设法将相应的栈顶部分缓存在 Cache 中，显然是最理想的。因此，应把Cache设计成相应栈区的逻辑延续。

（2）随机访问方式

由于过程调用、一致化成功或失败、子句索引等，都将导致执行流的转移和分叉。因此，代码区的访问是随机的。另外，环境体（STACK内）和全局数据区HEAP的访问也是随机的，这是因为存在变量约束链的缘故。对于随机访问的存储区域，须按常规设计Cache.

按照不同的访问方式，设立四个专用Cache（见图 2），其中，原STACK分成两个栈，即CPS和ENS,分别用作选择点栈和环境栈；指缓IB和数据缓存DB均是常规Cache；选择点缓存CPB和跟踪栈缓存TRB是栈式Cache,它们是相应功能栈（CPS和ENS）的逻辑栈顶。所有的访问将首先经过Cache层次，只在Cache不命中时才访问主存并进行相应的调度。

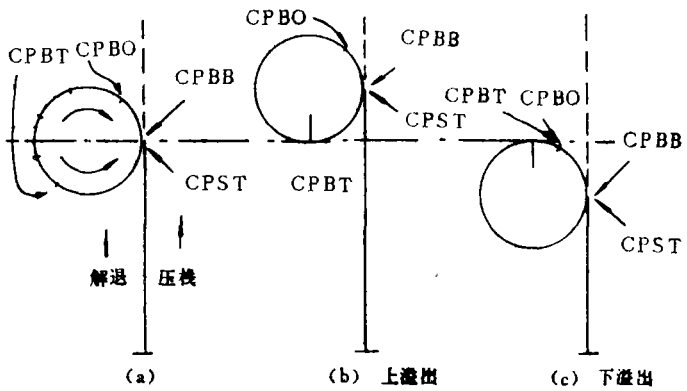

图 3　选择点Cache控制结构

CPBT：选择点指针，指向CPB中的栈顶选择点（等价于B）；

CPBO：选择点指针，指向CPB中的栈底选择点；

CPBB：栈底指针；

CPST：选择点指针，指向CPS中的栈顶选择点。

设立多个专用Cache的优越性在于：

（1）吻合不同的访式，提高Cache结构的效率；

（2）支持多路并行访问。

下面，我们略去常规型Cache 设计的讨论，单独给出有关CPB的控制和调度策略。

3.3　选择点Cache的控制策略与分析

3.3.1　以一个选择点为调度单位

以一个选择点为调度单位，CPB与CPS中的内容是不重复的。其基本调度思想是：CPB中至少缓存一个选择点（即当前选择点），当CPB上溢出时，将其中最早创建的一个（或多个）选择点迁移至CPS中，利用回收的空间创建新的选择点；而当CPB下溢出时，自动将CPS中顶部的一个选择点迁移至CPB中。

可见，CPB是按栈方式循环使用的，栈底是浮动的。图3形象地表示了CPB与CPS间的关系，其中CPS用直线表示，CPB表示成与之相切的圆。切点即CPB之栈底（CPBB）、CPS之栈顶（CPST）。创建选择点时CPB沿逆时针增长，删除选择点时CPB沿顺时针解退（图3(a)）；CPB上溢出时，将CPBO指出的选择点迁入CPS中，即CPB圆切CPS直线向上滚动（图3(b)）；而当CPB下溢出时，紧挨切点（由CPST指出）的选择点迁入CPB中，即CPB圆切CPS直线向下滚动（图3(c)）；自动将CPS中顶部的一个选择点迁移至CPB中。

下面分别就选择点访问的各种类型，具体给出相应的地址变换和调度过程，用类C语言描述。假定CPB的容量为Ncpb，注意，诸选择点指针满足如下关系式：

$$B = CPST + (CPBT - CPBB) \bmod Ncpb$$

$$CPST = CPBO \rightarrow B$$

$$CPBB = CPBO - (CPBO \rightarrow Na + Ns)$$

其中，(CPBO→Na+Ns) 系一个选择点的长度，Na系其中过程变元的个数，Ns系机器状态占用单元数，选择点的详细内容见[8]。为了实现快速调度，每个选择点内增加一个域B1，指向其后创建的选择点。这样，所有的选择点通过B域和B1域双向链接。

(1) 读、修改当前选择点

指令fail用当前选择点的内容恢复机器的状态寄存器和变元寄存器；指令retry和retry_me_else修改当前选择点中的P域。因为当前选择点总在CPB中，可通过CPBT直接访问。

(2) 建立选择点

涉及的指令有try和try_me_else.

```
{ n=Ns+Na;
  while ( (B+n-CPST) >Ncpb) /* CPB上溢出 */
      { CPST=CPST+CPBO→Na+Ns;
        *CPST=*CPBO; /* 转抄一个选择点 */
        CPBB=CPBO; CPBO=CPBO→B1;
      }
  将当前状态内容存入CPB（从CPBT指出的单元开始）;
  CPBT+=Na+Ns;
}
```

(3) 清除选择点

清除当前选择点，有关的指令为

trust和trust_me_else，其操作是 B:=B→B;

CUT操作，有关的指令为

"set_B_from Xi"，其操作是 B:=Xi;

set_B_From_CR，其操作是 B:=CR.

相应的控制如下，旨在确保当前选择点缓存在CPB中。

```
{ if (B≤CPST)
    { CPST=B→B;
      CPBB=0;        /* 从CPB栈底开始使用 */
      CPBT=CPBO=B→Na+Ns;
     *CPBT=*B;
    }
  else CPBT=(B-CPST+CPBB) mod Ncpb;
}
```

3.3.2　以定长选择点为调度单位

该方法的出发点是：用于过程调用的变元寄存器数目是有限的（设为Nr）。所以存在最大选择点尺寸Ncp（Ncp=Ns+Ncp)。设Ncpb为Ncp的整数倍，且以3.3.1中的指令的分类，给出相应的CPB调度过程。

(1) 读、修改当前选择点（同3.3.1(1)）。

(2) 建立选择点

```
{ if (CPBT=CPBB)      /* CPB上溢*/
    {CPST=CPST+Ncp;
     CPBB=(CPBB+Ncp) mode Ncpb;
     *CPST=*CPBB;     /* 转抄一个选择点*/
    }
}
```

(3) 清除选择点

```
{ if (B≤CPST)
    { CPST=B→B; CPBB=0; CPBT=Ncp;
      *CPBT=*B;
    }
  else CPBT=(B-CPST+CPBB) mod Ncpb;
}
```

3.3.3　以定长块为调度单位

以定长块（块长为Nb）为基本调度单位，当发生CPB上溢出时，将 CPB 栈底的 Nb 个单元存入CPS栈顶；当发生CPB下溢时，将CPS栈顶的Nb 个单元读入CPS 栈底。仍按上述指令分类，给出相应的CPB调度过程。

(1) 读、修改当前选择点

因为调度块的块长 Nb 与选择点长度不总是一致的，有可能当前选 择点只是部分缓存在CPB中。为此，在访问当前选择点中的单元（设其逻辑地址为 Si）时，需要对其是否在CPB中进行判断。

```
while (Si≤CPST)
    for (i=0; i--<Nb; )  /* 从CPS栈顶调入一块 */
    *CPBB=*CPST--; /*这里，CPBB和CPST用作单元指针*/
    CPBB=(CPBB-1) mod Ncpb;
```

(2) 建立选择点

```
Si=B+Ns+Na;
while (Si-CPST>Ncpb)
  for (i=0; i--<Nb; )  /* 向CPS栈顶存入一块 */
      CPBB=(CPBB+1) mod Ncpb;
      *(++CPST)=*CPBB;
  }
```

(3) 清除选择点

```
if (B<CPST)
for (i=0; i--<Nb; )
    *CPBB=*CPST--;
    CPBB=(CPBB-1) mod Ncpb;
```

3.3.4 讨论

在上述三种调度方案中，定长选择点方法的优缺点最为明显：控制简单、速度快，但浪费大量的选择点空间，因为它是按最大变元寄存器数规定一个选择点的占用空间的。在其它两种方法中，以选择点为调度单位的方案的主要优点是：对当前选择点的访问（修改、读），与定长选择点方案相同，都是直接在选择点Cache中进行。相比而言，按定长块调度时，对当前选择点中单元的访问，原则上都要作出是否命中选择点Cache的判断，故其响应时间将受到影响。不过，若用硬件实现Cache的访问和命中判断并行，则定长块方法和选择点方法将是可比的。这三种方法都是建立在浮动栈基础上的。

3.4 一种面向实现的选择点Cache的控制方案

3.3节一般性地描述了几种方案的控制流程。从快速实现的角度来说，需要提供专门的手段，支持Cache调度中的地址变换。例如，需要计算CPB与CPS的指针间的对应关系：

$$B=CPST+(CPBT-CPBB) \bmod Ncpb.$$

再者，在CPB发生下溢时，总是从CPB栈底（CPBB=0）开始调度，这是不必要的。在下面介绍的实现方案中，CPB容量Ncpb设计成$\log_2 Ncpb=Nb$，Nb为一正整数；表示CPB与CPS间单元对应关系用表达式：ptr_cpb=sig(ptr_cps, Nb)

其中，ptr_cpb和ptr_cps分别为CPB的CPS单元地址，sig(ptr_cps, Nb)表示截取地址ptr_cps尾部的Nb位有效位。不难看出，这正是常规Cache所采用的，即组相联数为1时的特例。借助于专用硬件线路，可自然地完成地址尾码的截取操作，没有额外开销。撤

销专用CPB指针寄存器，如CPBT、CPBO、和CPBB等。

(1) 读、修改当前选择点

因为当前选择点总在CPB中，通过sig(T, Nb)直接访问CPBT，其中T是当前选择点中一个单元的地址。

(2) 建立选择点

```
{ n=Ns+Na;
  while ( (B+n-CPST) >Ncpb)  /* CPB上溢出 */
        for ( i=0; i++<n; )
         {CPST++; *CPST=*sig(CPST, Nb); }
  将当前状态存入CPB中（从由sig(++B, Nb) 所指的单元开始）;
}
```

(3) 清除选择点

根据不同的清除选择点指令，对B进行修改。之后，为了确保当前选择点缓存在CPB中，相应的MCP的控制如下：

```
{ if (B≤CPST) /* CPB下溢出，调入当前选择点 */
    for (; CPST!=B→B; ; )
     { *sig(CPST, Nb) = *CPST; CPST--}
}
```

4 结束语

在一般地研究了顺序PROLOG机硬件组织的特点之后，作者具体介绍了其中的存储子系统的组织方案，用多专用CACHE结构来解决严重地存在于PROLOG过程执行中的访存效率问题。基本思想是：按照具体的访存方式，为多个用途各异的存储区域，分别设置专用的CACHE（和缓存），并采用不同的调度策略。文中详细地介绍了选择点缓存的控制和调度策略。至于其它方面，如CACHE的物理尺寸等，将通过进一步的研究工作（如模拟和实验）来确定。

致　　谢

本课题的研究是在慈云桂教授的悉心指导下进行的。在此，作者向慈云桂教授和导师小组的其他老师表示感谢。

参 考 文 献

[1] Clocksin W F and Mellish C S. Programming in Prolog, Springer-Verlag, 1981

[2] Kowalski R A. Logic for Problem Solving, Elsevier North Holland Inc., New York, 1979

[3] TaKi K. Hardware Design and Implementation of the Personal Sequential Inference Machine(PSI), In Proc. of the International Conference on Fifth Generation Computer Systems, 1984

[4] Tamura N, et al. Sequential Prolog Machine PEK, In Proc. of the International Conference on FGCS, 1984

[5] Tick E and Warren D H D. Towards a Pipelined Prolog Processor, In New Generation Computing, 1984; (2): 323~345, OHMSHA.LTD. and Springer—Verlag

[6] Tick E. Sequential Prolog Machine: Image and Host Architecture, 17th Annual Microprogramming Workshop, IEEE Computer Society, Oct., 1984

[7] Tick E. Memory Performance of Lisp and Prolog Programs, In Proceedings of the 3rd International Conference on Logic Programming, July, 1986

[8] Warren D.H.D. An Abstract Instruction Set, Technical Note 309, SRI International, Oct., 1983

[9] 慈云桂，刘桂仲，孙成政，张晨曦，李良良，王志英．新一代计算机的研究——我们的工作与初步结果。全国新一代计算机学术研讨会，1987

[10] 李良良，张晨曦．顺序推理机研究中的几个重要问题、国防科学技术大学学报，1987；(3)

[11] 李良良，慈云桂．YH—SIM：一种支持PROLOG数据库操作的顺序推理机系统结构、全国新一代计算机学术研讨会．1987

[12] 李良良．一种顺序PROLOG机系统结构的研究．工学博士学位论文，国防科学技术大学，1987

A Study on the Memory Organization for a Sequential PROLOG Machine

Li Liangliang

(Department of Computer)

Abstract

In this paper presented is a new multi—Cache scheme about the memory organization of a sequential PROLOG machine. Based on the memory access modes, each memory area for different usage and with different access mode is supported with a separate cache using an associated control policy, in order to overcome the access bottleneck problem incurred in the execution of PROLOG programs. As a typical component, the cache for Choice Point stack is investigated in details with respect to the control and schedule policies.

Key words: PROLOG machine, memory organization, multiple caches