一种基于汇编代码的单重循环向量化方法*)

陆洪毅　戴　葵　王志英
(国防科技大学计算机学院　长沙410073)

A Single Loop Vectorization Method Based on Assemble Code

LU Hong-Yi　DAI Kui　WANG Zhi-Ying
(Computer College, National University of Defence Technology, Changsha 410073)

Abstract　Through loops vectorization in instruction sequence, the vector power provided by hardware can be fully utilized. This paper analyzes the RISC instruction set, and presents a single loop vectorization method that is based on assemble code, it can efficiently detect single loops in instruct sequence and vectorize them.
Keywords　Vectorization, Assemble code, Loop, RISC

1　引言

在微处理器设计中，开发指令级并行(ILP)以提高微处理器系统的性能受到了很大的限制。研究更大发射的超标量微处理器已经是一件极其复杂而没有意义的事[8]。但是如果在开发指令级并行的同时，开发数据级并行，理论分析表明可以显著提高微处理器的性能，微处理器的等效IPC(每个时钟周期发射的指令条数)和超标量微处理器相比可以提高20～40多倍[6,7]。因此，在微处理器系统设计中开发数据级并行具有重要的理论意义和实用价值。

开发数据级并行(DLP)关键是识别程序中的可向量化成分。基于源程序的向量化方法已经取得了显著的成效，通过对算法本身的改进，可以极大地提高程序的并行执行效率[1,2]。但是在指令代码一级开发程序的可向量化成分的研究还较少，而该研究对于保证研制的微处理器系统和某种已有的指令级结构二进制代码兼容具有重要的现实意义。

通过将存储器和处理器单元集成到同一个芯片上面而形成的PIM(Processor In Memory)技术，将极大地提高存储器和处理器单元之间的带宽。研究表明，在处理单元中采用向量处理部件将会取得较好的效果[3～5]。因此，研究在指令代码一级实现程序的向量化将是一个有意义的课题。

研究表明，基于指令源代码的向量化分析主要集中在指令序列的循环部分，尤其是在数值应用程序中，更是如此。本文将以RISC指令级结构为基础，针对指令源代码的单重循环，在不考虑向量资源受限的情况下，给出一种有效的向量化方法。文中通过一些DLX单重循环指令代码的向量化实例验证了该方法的有效性，并对向量化后的指令代码的执行性能进行了综合评价。

2　基本定义和操作

定义1　指令 i 可以定义为如下三元组：

$i=\langle op, SR, dR\rangle$

其中：op 为指令 i 的操作码，SR 为指令 i 的源操作数集合，dR 为指令 i 的目的操作数。

定义2　单重循环指令序列集合 I_S 是由 n 条顺序执行的标量指令序列组成的，也即：

$I_S=\{i_j^S \mid j\in N\}$

定义3　向量指令序列集合 I_V 是由 m 条顺序执行的向量指令序列组成的，也即：

$I_V=\{i_j^V \mid j\in N\}$

定义4　存储器载入操作码集合 M_L 是由某种RISC指令集结构中的所有 k 存储器载入操作码组成，也即：

$M_L=\{op_L^1, op_L^2, \cdots, op_L^k\}$

定义5　标量/向量操作码对照表 SV_{op} 是由下面的二元偶对所组成的集合：

$SV_{op}=\{\langle S_{op}^i, V_{op}^i\rangle \mid i\in N\}$

其中：S_{op}^i 为指令集结构中的第 i 个标量操作码，V_{op}^i 为与 S_{op}^i 相对应的向量操作码。

定义6　标量/向量操作码对照表 SV_{op} 的查表操作定义为：给定一个标量操作码，在标量/向量操作码对照表 SV_{op} 中找出与之相对应的向量操作码，也即：

$V_{op}^i=F_V(S_{op}^i)$

定义7　标量/向量寄存器分配操作 VA 定义为：给定一个标量寄存器 R_i，按照某种映射规则将其映射到向量寄存器 V_j，也即：

$V_j=VA(R_i)$

定义8　标量/向量寄存器对照表 SV_R 是由下面的二元偶对所组成的集合：

$SV_R=\{\langle R_i, VA(R_i)\rangle \mid i\in N\}$

其中：R_i 为指令集结构定义的第 i 个标量寄存器，$VA(R_i)$ 为其相对应的向量寄存器。

定义9　标量/向量寄存器对照表 SV_R 的查表操作 Q_V 定义为：给定一个标量寄存器名称，在标量/向量寄存器对照表 SV_R 中找出与之相对应的向量寄存器名称，也即：

$V_j=Q_V(R_i)$

3　向量化算法

问题描述：现假设有一单重循环指令序列 I_S，对其进行向量化的任务就是将其中可向量化指令提取出来，形成由标量指令序列和向量指令序列组成的指令序列集合。

算法描述：

1)设：集合 $SR=\Phi$，$DR=\Phi$；

对所有的指令 $i\in I_S$，如果 $i.op\in M_L$，那么：

*)本课题受自然科学基金支持(60173040)。陆洪毅　博士，主要研究方向是计算机系统结构、高性能微处理器设计、向量微处理器设计。戴　葵　副教授，主要研究方向是计算机系统结构。王志英　博士生导师，主要研究方向是计算机系统结构。

$DR=DR+i.dR$

$SR=SR\cup i.SR$

2)设:可向量化标量寄存器集合 $M=\Phi$;

对 SR 中的所有标量寄存器 R_i,如果存在指令 $i\in I_S$,$i.op\notin M_L$,且 $R_i=i.dR$,则:

$M=M+R_i$

3)设:标量/向量寄存器对照表 $SV_R=\Phi$;

对 M 中的寄存器 R_i,对其相应的存储器载入指令 i,如果:$i.dR\in DR$,则:

$SV_R=SV_R+\langle i.dR,VA(i.dR)\rangle$

4)设:向量指令序列 $I_V=\Phi$,Flag=0;

对所有的指令 $i\in I_S$,有:

·如果对 $R_j\in i.SR$,如果 $Q_V(R_j)\neq\Phi$,则:

$R_j=Q_V(R_j)$

Flag=1;

·如果:$Q_V(i.dR)\neq\Phi$,则:

$i.dR=Q_V(i.dR)$

Flag=1

·如果:$Q_V(i.dR)=\Phi$,且 $i.dR\notin M$,Flag=1,则:

$SV_R=SV_R+\langle i.dR,VA(i.dR)\rangle$

$i.dR=Q_V(i.dR)$

Flag=1

·如果:Flag=1,则:

$i.op=F_V(i.op)$

$I_V=I_V+i$

$I_S=I_S-i$

Flag=0

4 向量化实例

实例:假设有如下实现如图1所示的 DAXPY 的循环指令序列。

```
LOOP: LD      F1, 1000(R2);          //载入 X
      LD      F2, 5000(R2);          //载入 Y
      MULTD   F3, F1,      R1;       //计算 a*X
      ADDD    F4, F3,      F2;       //计算 a*X+Y
      SD      F4, 5000(R2);          //保存 Y
      ADD     R2, R2,      #8;       //循环控制变量递增
      SUB     R5, R2,      #504;     //判断循环结束
      BNEZ    R5, LOOP;              //分支循环
```

图1 DAXPY 循环指令序列

$SV_{op}=\{\langle LD,LDV\rangle,\langle SD,SDV\rangle,\langle ADDD,ADDDV\rangle,\langle MULD,MULDV\rangle\}$,$M_L=\{LD\}$

经过向量化过程之后,该单重循环的指令序列如图2所示。

```
LOOP: ADD     R2, R2,      #8;       //循环控制变量递增
      SUB     R5, R2,      #504;     //判断循环结束
      BNEZ    R5, LOOP;              //分支循环
      LDV     V1, 1000(R2)           //向量载入
      LDV     V2, 5000(R2)           //向量载入
      MULTDV  V3, V1,      R1        //向量乘法
      ADDDV   V4, V3,      V2        //向量加法
      SDV     V4, 5000(R2)           //向量保存
```

图2 向量化后的 DAXPY 指令序列

值得注意的是,上述的向量代码并非是一种完全正确的表示,因为控制循环次数的寄存器变量(R2)决定了向量操作的次数或者长度。对于简单的单重循环,可以通过简单的方法跟踪循环控制变量的改变来计算所需的向量操作次数或者长度。基本算法如下:

1. 检查指令流中的跳转指令。在实例程序中,即为 BNEZ。

2. 检查跳转指令所操作的寄存器。在实例中,即为 R5。

3. 检查指令流中修正了 R5寄存器并影响跳转指令的地方。在实例中,即为 SUB。

4. 判断 R5的初始值,以及修改的步长。在实例中,即为 504与8。

5. 求出循环的次数即为向量的长度。在实例中,即为64,因此向量的长度就为64。

因此,我们可以将代码转换成如图3所示的形式。

```
SETVL   VL,   #64              //设置向量长度
LDV     V1,   1000(R2)         //向量载入
LDV     V2,   5000(R2)         //向量载入
MULTDV  V3,   V1,    R1        //向量乘法
ADDDV   V4,   V3,    V2        //向量加法
SDV     V4,   5000(R2)         //向量保存
```

图3 调整后的 DAXPY 向量指令序列

实例中,由于是一个完全的数值运算,因此可以完全消除循环。在实际的应用中,可能还需要在循环中处理别的指令,这时,可以将向量运算与其他指令相分离。

5 性能分析

假设在五级 DLX 流水线上实现 DAXPY 的标量指令序列,那么指令执行的时空图如图4所示。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LD	F	D	E	M	W									
LD		F	D	E	M	W								
MULD			F	D	E	M	W							
ADDD				F	D	E	M	W						
SD					F	D	E	M	W					
ADD						F	D	E	M	W				
SUB							F	D	E	M	W			
BNEZ								F	D	E	M	W		
LD											F	D	E	M

图4 执行 DAXPY 标量指令序列的流水线时空图

可以看出上述循环需要(8×64)+(2×63)=638个时钟周期。

对于后面的向量指令序列而言,假设其功能部件的执行序列如图5所示。

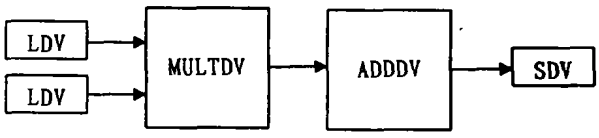

图5 DAXPY 向量指令序列的执行框图

如果各个功能部件的启动时间均是5个时钟周期,并且各个功能部件能够流水处理,那么显见实现上述向量指令序列需要84个时钟周期。那么整个向量化的加速比将是638/84=7.60。

(下转第124页)

的内容。

参考文献

1 National Security Agency. Information Assurance Technical Framework (IATF), Version 3.0. http://www.iatf.net, 2000
2 Rushby J. Security Requirements Specifications: How and What. Symposium on Requirements Engineering for Information Security (SREIS), 2001
3 Helmer G, et al. A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System. Symposium on Requirements Engineering for Information Security (SREIS), 2001
4 Lutz R R. Software Engineering for Safety: A Roadmap. Future of Software Engineering(FoSE) at ICSE'00, 2000
5 The International Organization for Standardization. Common Criteria for Information Technology Security Evaluation. ISO/IEC15408:1999(E), 1999
6 McDermott J, Fox C. Using Abuse Case Models for Security Requirements Analysis. In: Proc. of the 15th Annual Computer Security Applications Conf. 1998
7 Leiwo J, Zheng Yuliang. A Formal model to aid documenting and harmonizing of information security requirements. In: Proc. of the IFIP TC11 13th Intl. Conf. on Information Security (SEC'97), 1997. 25～38
8 Leiwo J, Gamage G, Zheng Yuliang. Harmonizer- A Tool for Processing Information Security Requirements in Organizations. In: Proc. of the Third Nordic Workshop on Secure IT Systems (NORDSEC'98), 1998
9 U.S. DEPARTMENT OF COMMERCE. Small Business Innovation Research Program, ABSTRACTS OF AWARDS FOR FISCAL YEAR 1999. http: // www.rdc.noaa.gov/～ amd/abstracts1999.pdf
10 Payne S C. A Guide to Security Metrics. http://www.sans.org/infosecFAQ/audit/metrics.htm, 2001
11 Nielsen F. Approaches to Security Metrics. A Report of the Workshop Held at the National Institute of Standards and Technology (NIST) In conjunction with the Computer System Security and Privacy Advisory Board (CSSPAB) Meeting, 2000
12 Bodeau D J. Information Assurance Assessment: Lessons-Learned and Challenges. http: // philby.ucsd.edu/～ cse291_IDVA/papers/rating-position/Bodeau.pdf, 2001
13 Jelen G F, Williams J R. A Practical Approach to Measuring Assurance. 14th Annual Computer Security Applications Conf. Dec. 1998
14 Williams J R, Jelen G F. A Framework for Reasoning about Assurance. National Institute for Standards and Technology, DRAFT, 1995
15 Eames D P, Moffett J. The Integration of Safety and Security Requirements. Safecomp'99, 1999. 27～29
16 Pfleeger C P. Security in Computing. Second Edition. London: Prentice Hall PTR, 1997. 462～471
17 Lutz R R, Shaw H-Y. Applying Adaptive Safety Analysis Techniques. In: Proc. of the Tenth Intl. Symposium on Software Reliability Engineering, 1999
18 蒋慧，吴礼发，陈卫卫. UML设计核心技术. 北京：北京希望电子出版社，2001. 9～21
19 SSO. The Systems Security Engineering Capability Maturity Model (SSE-CMM). http: // www.sse-cmm.org/model.htm, 1999
20 Williams J R, Jelen G F. A Framework for Reasoning about Assurance [R]. Document Number ATR 97043, Arca Systems, Inc., 1998
21 何全胜，姚国祥. 网络安全需求分析及安全策略研究. 计算机工程，2000, 26(6): 56～58
22 周之英. 现代软件过程(中)－基本方法篇. 北京：科学出版社，2000. 1～26

(上接第117页)

在我们自行设计的高性能嵌入式微处理器银河TS-1[9]中，通过测试可以获得如图6所示的数据。

		非向量版本	向量版本	加速比
指令条数		13	8	
执行周期数	VLR=4	294	132	2.23
	VLR=8	578	210	2.75
	VLR=16	1146	360	3.18
	VLR=32	2282	666	3.43
	VLR=64	4554	1272	3.58

图6 向量化性能测试结果

由此可见，向量化之后对程序的执行性能提高是极有好处的。

结束语 如果在微处理器设计中，在微处理器内部设置相关的向量部件，并且在编译过程中对程序进行向量化，可以提高程序在微处理器上的执行性能。本文提出的单重循环向量化方法实验结果表明，可以在二进制代码兼容的条件下达到优化程序的执行性能。将此向量化方法运用在我们自行设计的银河TS-1高性能嵌入式微处理器[9]中，结合硬件的向量处理功能，取得了良好的效果[10]。

同时，由于在处理器中引入了向量处理部件，将增大访存单元的负担。特别是算法所需要的支持非连续存储器单元访问的功能(LDV、SDV)，是实现这一算法的难点。但是，由于执行代码的向量化，可以使得访存单元能够提前得到有关数据地址的信息，有利于连续读入/写入相关的数据。另外，由于向量部件可以在较长的时间内充分利用功能部件，因此可以在硬件上实现更多的存储单元访问操作。

本文所考虑的情况是基于硬件资源不受限的单重循环向量化，如何开发多重循环的向量化成分，如何在向量处理部件资源受限的条件下，在向量化过程中进行资源的分配也是我们在今后的工作中将要着重研究的问题。

参考文献

1 李晓梅，蒋增荣，等. 并行算法. 湖南科学技术出版社，1992
2 张丽君，金绥更. 向量算法与并行算法. 国防工业出版社，1993
3 Lopez D, Valero M, et al. Increasing Memory Bandwidth with Width Bus: Compiler, Hardware and Performance Trade-offs. ACM, 1997
4 Burger D, et al. Memory Bandwidth Limitations of Future Microprocessors. ACM, 1996
5 Saulsbury A, et al. Missing the Memory Wall: The Case for Processor/Memory Integration. ACM, 1996
6 Quintana F, Espasa R, Valero M. Performance Advantages of Merging Instruction- And Data-Level Parallelism (Extended Version)
7 Espasa R, Valero M. Exploiting Instruction and Data Level Parallelism in Future High Performance Processors
8 Wall D W. Limits of Instruction-Level Parallelism: [David W. Wall, WRL Technical Note TN-15]. Western Research Laboratory, Dec. 1990
9 陆洪毅，赵学秘，王蕾，戴葵，王志英. 一种高性能的嵌入式微处理器：银河TS-1. 电子学报，2002. 10
10 宋辉，戴葵，王志英. 基于银河TS-1的高性能量子计算. 电子学报，2002. 10