第 35 卷　第 1 期　　计　算　机　工　程　　2009 年 1 月
Vol.35　No.1　　Computer Engineering　　January 2009
·工程应用技术与实现·　文章编号：1000—3428(2009)01—0201—03　文献标识码：A　中图分类号：TP314

SIMD 计算机的优化编译器设计

赵　辉，黄　石
（西安理工大学计算机科学与工程学院，西安 710048）

摘　要：利用处理器的相关资源，提高编译器优化性能和增强代码可适应性是 SIMD 处理器优化编译的关键。该文基于 M 语言和 LS SIMD 体系结构，结合现代编译器的编译技术，提出针对 SIMD 协处理器编译器的优化和实现方法，包括寄存器分配、单值合并、代码压缩等。实验结果表明，编译生成的目标代码准确、高效。

关键词：M 语言；LS SIMD 协处理器；编译器

Optimizing Compiler Design for SIMD Computer

ZHAO Hui, HUANG Shi
(School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048)

【Abstract】 Utilizing the resource of the processor to improve the execution efficiency of the program and to increase the flexibility of the code is the key problem of optimizing complier for SIMD processor. This paper introduces M language and Li-Shan Single Instruction Multiple Data(LS SIMD) computer architecture and presents optimization method and implementation method of the M compiler for SIMD coprocessor combining with modern compiler techniques. It includes assigning register combining single value and compressing code, etc. Experimental result shows that the M compiler can generate exact and high efficient target code.

【Key words】 M language; Li-Shan Single Instruction Multiple Data(LS SIMD) coprocessor; compiler

随着大规模集成电路技术和计算机体系结构的进一步发展，SIMD 计算机在军事侦察、天气预报、卫星图像处理等诸多领域得到越来越广泛的应用。面对应用系统的复杂性，使用串行语言的编译器设计方法已经不能满足数据并行计算机体系结构的需要。SIMD 计算机在构造上与传统的 RISC 计算机不同，其面向数据存储的特性以及一些特殊操作指令，使得由编译器产生代码的效率仍然很低。由于 SIMD 处理器的广泛应用以及集成系统芯片的趋势，开发出适合于 SIMD 优化的编译器已经成为研究的热点。

1　M 映射语言与 LS SIMD 阵列

1.1　M 映射语言

在嵌入式数据并行计算系统的设计开发过程中，针对高级语言程序一般都存在编译质量问题，汇编(机器)语言也存在兼容性与可读性差的缺点。M 语言是一种介于高级语言与汇编(机器)语言之间的中间语言。它采用面向高级语言的赋值语句与控制语句的描述形式，从而具有高级语言的可读性与多目标性，可作为汇编级的“高级语言”使用，解决高级语言的编译质量问题。采用面向机器语言的语义粒度，使 M 语言具有汇编语言的程序实时性与映射直接性。M 语言包括串行部分和并行部分，其基本框架描述如下：

```
int subname(formal parameter list)     //子函数定义
{
  body;   }
simd functionname(formal parameter) //数据并行部分函数
 {…
  r1=(moveast)r2; //阵列处理元之间流路径赋值   }
  void main()     //主函数
  {…
  i_x=i_y+i_z;   //整型变量相加
  while(f_r1<f_r3)   //浮点寄存器内容判断   { …
  f_r1=f_r1+0.1;     //浮点寄存器内容相加
  } }
```

1.2　LS SIMD 阵列

LS SIMD阵列协处理器是针对嵌入式图像理解应用而设计的一种定点阵列微处理器，由处理元PE阵列及控制部件CU构成，所有PE之间以网格互连，由沿行或沿列的播送互连[1]。LS SIMD协处理器并行处理元由 1 024 个处理元以二维网格互连方式构成 32×32 的网络。每个处理元具有 16 位运算字宽，能够执行通常的 16 位运算和逻辑运算操作。阵列互连模型如图 1 所示[2]。

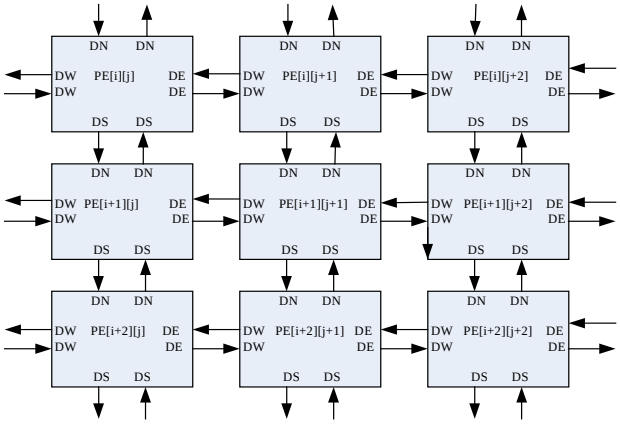

图 1　二维 PE 阵列的互连模型

作者简介：赵　辉(1976－)，男，硕士研究生，主研方向：数据并行编译器设计；黄　石，硕士研究生

收稿日期：2008-05-15　**E-mail：**zhaokuafu@163.com

在LS SIMD并行处理阵列中，PE之间采用网格互连以及播送互连以支持数据并行模型[3]。相应的传送指令有沿行或沿列的播送指令以及上下左右 4 个方向的相邻处理器之间的传送指令。LS SIMD计算机将图像阵列中的每个像素映射到一个唯一处理元上，然后将同样的操作同时施加到这个数据结构的所有或大多数元素上。

2 编译器实现技术

2.1 编译器结构

编译器可分为前端和后端。编译器前端包括词法分析、语法分析、语义分析和中间代码生成，它主要依赖于源语言，与目标机体系结构无关。编译器后端包括代码优化、代码选择、代码生成、寄存器分配等。一般而言，上述部分独立于源语言，仅依赖于中间表示和目标机器。本文设计的编译器结构如图 2 所示。

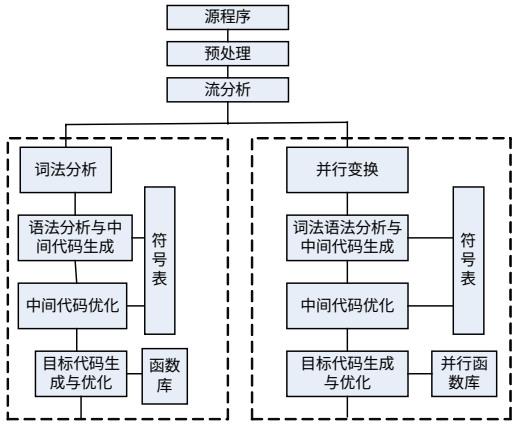

图 2 编译器结构

2.2 词法和语法分析

编译器的词法和语法分析器采用lex和yacc工具构造[4]。词法分析器需要识别的单词(token)主要有：关键字，标识符，数字，字符串和操作符等。词法分析器将M语言源程序解释成一个个独立的记号，然后将记号的类型以及记号所对应的值返回给语法分析器。如在关键字表中设置关键字moveeast, movewest, movesouth, movenorth, simd等，用它们将串行语句与数据并行语句区分开。

语法分析实现的主要任务是设计语法规则以及相应的语义分析子程序。编译器工作时，使用哪个产生式进行规约，就执行该产生式的属性动作，目标码的生成是通过产生式对应的属性动作来完成，其文法采用自顶向下逐步求精的方法实现。

2.3 中间表示

中间表示(IR)采用树的形式，把语句和树模板相对应并且将其做成匹配规则，代码选择等同于寻找 1 个用树模板对该树的匹配集合。树模板匹配方法实现了匹配规则的独立性(规则只与中间表示及机器指令相关)。对于 1 棵给定的中间表示树，可能存在多种匹配序列。这些匹配有优劣之分，于是引入匹配代价来判断匹配的优劣。匹配代价的大小根据所生成的代码执行速度来考虑。生成的代码执行速度越快，则代价越低。代价的计算是叠加的，匹配整棵树的代价等于其每棵子树所使用的匹配规则代价之和。因此，根据这种定义，最优的匹配集合就是整体匹配代价最小的匹配序列。

2.4 代码生成器

代码生成通过标记过程和规约过程来实现。代码生成器主要包括标记过程、规约过程和匹配过程。标记过程是自底向上的用所有匹配规则的树模板去试图匹配中间表示树。如果发现它的某一棵子树符合某匹配规则，则用该规则标记树的根节点，并把该子树规约成其根节点。该过程一直到根节点为止。通过匹配代价的计算和比较，得到最优的匹配序列。

编译 M 语言数据部分的变量和常数时，对变量的标识符和值都写入汇编文件的数据段内为其分配空间，并增加相应的符号表项；对常数先分配标识符，再以变量的方式处理。

在编译函数时，如果函数定义后面没有函数体，则认为这只是一个函数的定义，只是为它生成符号表项，记录它相应的信息。如果函数定义的后面有函数体，则先为它加入函数头部，根据函数体的代码，为它生成相应的汇编代码、返回动作 return Ar4,0(Ar4 定义为函数返回地址寄存器)和结尾，写入汇编文件的代码段。代码优化在生成汇编语言之前由优化器完成。

3 编译器优化方法

编译器的功能并不仅仅只是生成正确的目标程序，除正确性之外，还须重点考虑编译得到的机器代码的有效性。处理器性能的发挥很大程度上依赖于编译系统能否产生效率高的代码。编译优化技术就是要消除简单语言翻译中可能引入的低效率，改进目标程序的性能。本文 SIMD 计算机编译器设计中充分结合现代编译技术所采用的优化方法和 SIMD 计算机硬件特性，提出了以下几种编译器优化技术。

3.1 寄存器分配

由于在 SIMD 阵列计算机中含有用于寻址操作的辅助寄存器，且汇编代码基本上以间接寻址模式为主。数据从阵列存储器送到寄存器需要执行的时间远远大于阵列处理元寄存器之间传送数据的时间，如：在 LS SIMD 数据并行阵列处理器中，把数据从寄存器送到数据区需要 1 040 个时钟周期，同样将数据从数据区送到寄存器中也需要 1 040 个时钟周期，利用 1 次结果暂存寄存器可以节省 1 040×2 个时钟周期。暂存结果寄存器设置技术，大大节省了程序执行时间，这也导致暂存结果寄存器的频繁使用，需要一个好的替换策略来提高存取效率。现代编译器普遍采用的“最少使用优先替换”策略是合适的，它计算变量引用次数并在每次装载新的地址和中间结果时将最少引用的寄存器替换掉，而高利用率的变量仍然保存在寄存器中，这也是时间局限性原理要求的。所分配的寄存器是伪寄存器，与目标机器有多少可用寄存器无关，只有在生成目标代码时，才由各个伪寄存器变量的生存周期决定将伪寄存器与实际寄存器多对一地对应起来。

3.2 单值合并

SIMD 数据并行协处理器在计算单值变量和数组变量时所耗费的时间和能耗一样多，这是因为处理元阵列执行操作指令时，所有处理元同时执行同一操作指令，这种硬件工作方式对处理图像数据(数组变量)非常有利，而在处理单值变量计算时，却会造成大量的能耗浪费。为解决单值变量计算能耗浪费的情况，本文提出基于静态检测的单值合并技术。算法的伪代码表示如下：

```
if(GetVarSymbolArray(Var1)&& GetVarSymbolArray(Var2))
{
StatementVar();   //执行数值合并计算
  }
else
  {
```

```
StatementArray();   //作相应处理
}
```

通过 *GetVarSymbolArray*()函数来判断当前参加计算的变量是不是数组变量，只有当参与计算的 2 个变量不是数组时，才能进行合并计算产生临时变量，由函数 *StatementVar*()来完成相应动作。假如参与运算的有 1 个或 2 个都是数组变量则由函数 *StatementArray*()来完成非单值变量的相应动作。

3.3 代码压缩

由于 SIMD 数据并行计算机内存容量大小有限制，因此应用软件必须做到代码紧凑。但是有一些代码优化方法却使得最终目标代码增加，如循环展开等代码变换，有必要在执行效率和空间占用上作一些折中，压缩代码长度，使程序代码紧凑。本文采用基于字典的压缩方法，它的基本原理是寻找公共代码并将其合并为宏定义的策略，从代码流中找出程序中相同的代码，用一份代码和一些指针做成字典，在原来公用块出现的地方使用访问字典项的指令代替，这样就可大大减少主程序的长度。

根据文献[5]的算法描述，需要如下数据结构：(1)宏代码缓冲区，用来存放公共的宏代码。(2)字典索引表，用来查找宏，并可以反向查找主程序中该宏所出现的位置，其中字典索引表的每一项指针指向宏代码缓冲区中的一段宏代码，并且还指向主程序体中的该代码所处的位置。(3)主程序，由代码和指向宏代码的指针组成。代码的压缩过程如图 3 所示。

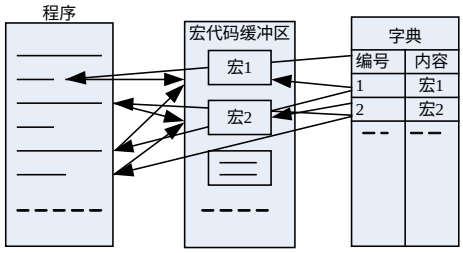

图 3　代码的压缩

进行代码压缩时，找出可以合并的重复代码序列(公共程序块)设置为宏，根据这些公用块的长度、出现的次数和位置，决定将那些重复代码合并，同时解决重复代码相互覆盖的问题，最终确定的公共程序块不允许相交。提取重复代码直接复制到字典中，加上子程序调用返回语句。将原公共程序块替换为子程序调用，完成代码合并过程。主要的工作是尽可能寻找最长的公共程序块集。基于字典的压缩方法压缩比大致为 73%~93%，运行时间增长并不显著。

4　编译实例分析

局部运算通常用于图像的平滑运算或正好相反的边缘检测运算。以 3×3 的邻域为例，计算每个像素需要用 9 个数据值。由于每个像素通常存储在不同的处理元中，因此局部运算需要一定量的并行数据交换。该程序先计算局部和，然后除以像素的个数。一个选定 3×3 像素的领域需要将每一个像素(该像素自身的值和 8 个相邻的像素的值)的 9 个灰度值相加，然后除以 9 所得结果为均值。这里定义图像的像素值存放在数组 *img* 中，数据并行协处理器是二维的 32×32 网格互连阵列。

局部求和与均值程序如下：

```
simd sum_3×3(img){
r1=img;
r2=(moveeast)r1;
r3=(movewest)r1;
r2=r2+r3;
r3=(movenorth)r1;
r2=r2+r3; r3=(movesouth)r1;
r2=r2+r3;
img=r2;
}
…
sum_3×3    proc
ldn Ar0,img ;
ldary Ar0,0 ;
movdst r1 ;
movw r1 ;
movrdst r2 ;
move r1 ;
movrdst r3 ;
addi r2,r2,r3 ;
movs r1 ;
movrdst r3 ;
addi r2,r2,r3 ;
movn r1 ;
movrdst r3;
addi r2,r2,r3 ;
movsrc r2 ;
ldn Ar0,img ;
stary Ar0,0 ;
return Ar4,0 ;
sum_3×3 endp
…
simd mean_3×3(img){
simd sum_3×3( img);
r1=img;
r1=r1/9;
img=r1;
}
…
mean_3×3 proc
blink Ar4,sum_3×3;
ldn Ar0,img;
ldary Ar0,0;
movdst r1;
ldn Ar0,var_1;
ldcell Ar0,0;
movdst r0;
divi r1,r1,r0;
movsrc r1 ;
ldn Ar0,img ;
stary Ar0,0;
return Ar4,0;
mean_3×3 endp
```

实验结果表明，所有的 M 语言程序执行结果和汇编程序执行结果一致。代码大小和运行周期的统计及对比结果如表 1 所示。

表 1　目标代码质量统计对比

	手写汇编	编译产生汇编	优化率
代码大小	1	1.326	1.076
运行周期	1	1.423	1.068

（下转第 206 页）