

分簇 VLIW DSP 上支持单双字模式选择的 SIMD 编译优化

黄胜兵^{1,2*}, 郑启龙^{1,2}, 郭连伟^{1,2}

(1. 中国科学技术大学 计算机科学与技术学院, 合肥 230027; 2. 安徽省高性能计算重点实验室(中国科学技术大学), 合肥 230027)

(*通信作者电子邮箱 huangsb@mail.ustc.edu.cn)

摘要: BWDSP100 是一款采用超长指令字(VLIW)和单指令多数据流(SIMD)架构的针对高性能计算领域而设计的 32 位静态标量数字信号处理器, 其指令级并行(ILP)主要是通过其特殊的分簇体系结构和 SIMD 指令来实现, 然而现有的编译框架无法对这些特殊的 SIMD 指令提供支持。由于 BWDSP100 拥有丰富的 SIMD 向量化资源, 且其所运用的雷达数字信号处理领域对程序的性能要求极高, 因此针对 BWDSP100 结构的特点, 在传统 Open64 编译器中 SIMD 编译优化框架的基础上提出并实现了一种支持单双字模式选择的 SIMD 编译优化算法, 通过该算法可以显著提高一些在 DSP 上有着广泛运用计算密集型程序的性能。实验结果表明, 与优化前相比, 该算法方案在 BWDSP 编译器上的实现能够平均取得 5.66 的加速比。

关键词: 编译优化; 指令级并行; 分簇体系数字信号处理器; 超长指令字; 单指令多数据流; Open64 编译器

中图分类号: TP311.54 **文献标志码:** A

SIMD compiler optimization by selecting single or double word mode for clustered VLIW DSP

HUANG Shengbing^{1,2*}, ZHENG Qilong^{1,2}, GUO Lianwei^{1,2}

(1. School of Computer Science and Technology, University of Science and Technology of China, Hefei Anhui 230027, China;
2. Anhui Province Key Laboratory of High Performance Computing (University of Science and Technology of China), Hefei Anhui 230027, China)

Abstract: BWDSP100 is a 32-bit static scalar Digital Signal Processor (DSP) with Very Long Instruction Word (VLIW) and Single Instruction Multiple Data (SIMD) features, which is designed for high-performance computing. Its Instruction Level Parallelism (ILP) is acquired through clustering and special SIMD instructions. However, the existing compiler framework can not provide support for these SIMD instructions. Since BWDSP100 has much SIMD vectorization resources and there are very high requirements in radar digital signal processing for the program performance, an SIMD optimization which supported the selection of single or double word mode was put forward based on the traditional Open64 compiler according to the characteristics of BWDSP100 structure, and it can significantly improve the performance of some compute-intensive programs which are widely used in DSP field. The experimental results show that this algorithm can achieve speedup of 5.66 on average compared with before optimization.

Key words: compiler optimization; Instruction Level Parallelism (ILP); multi-cluster Digital Signal Processor (DSP); Very Long Instruction Word (VLIW); Single Instruction Multiple Data (SIMD); Open64 compiler

0 引言

BWDSP100 是一款针对高性能计算领域而设计的 32 位静态标量数字信号处理器(Digital Signal Processor, DSP), 采用超长指令字(Very Long Instruction Word, VLIW)、单指令多数据流(Single Instruction Multiple Data, SIMD)架构, 其内核由 4 个构造完全一致的基本执行簇组成, 分别为簇 x、簇 y、簇 z 和簇 t, 结构如图 1 所示。此外, 该处理器内部还有三个结构相同的地址产生器 u、v、w, 用来执行地址运算指令和访存指令^[1]。

BWDSP100 的 SIMD 架构并不是通过长向量化部件, 而是通过提供许多 SIMD 指令来支持 SIMD 向量化, 例如: $\{x, y, z, t\} Rs = Rm + Rn$ 表示在多个簇上同时对同一组数据进行加法操作, 其中 $\{x, y, z, t\}$ 可以是 xyzt 的任意组合, 表示执行该指令

的簇; $\{x, y, z, t\} Rs = [Un + = Um, Uk]$ 和 $\{x, y, z, t\} Rs + 1; s = [Un + = Um, Uk]$ 分别表示单字和双字访存指令, 其中表示地址的操作数有三个, 前两个分别是基地址和偏移地址, 第三个操作数表示簇间地址偏移^[2]。

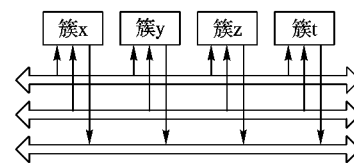


图1 BWDSP100 主要结构

BWDSP 编译器采用 Open64^[3] 编译器基础设施作为编译器研究框架, 其主要工作在于后端重定向和针对特定体系结构的优化。Open64 完善的前端处理功能能够将源程序代码转化成相应的层次化的中间表示 (Winning Hierarchical

收稿日期: 2015-03-23; 修回日期: 2015-06-02。 基金项目: 国家“核高基”重大专项(2012ZX01034-001-001)。

作者简介: 黄胜兵(1990-), 男, 安徽安庆人, 硕士研究生, 主要研究方向: 并行编译; 郑启龙(1969-), 男, 四川成都人, 副教授, 硕士, 主要研究方向: 并行编译; 郭连伟(1990-), 男, 安徽阜阳人, 硕士研究生, 主要研究方向: 并行编译。

Intermediate Representation Language, WHIRL),支持全程序的分析优化,功能强大;同时还具有很好的移植性,后端支持多种体系结构,包括 MIPS、Itanium、ARM、AMD64、X86-64、IA32、NVIDIA 等。

Open64 的前端将源程序转化为中间表示 WHIRL^[4],后端读入 WHIRL,经过翻译生成代码生成阶段(Code Generation, CG)的中间表示 CGIR(Code Generation Intermediate Representation),再经过一系列优化,最终 CGIR 经过代码输出生成汇编程序。Open64 编译器的架构如图 2 所示。

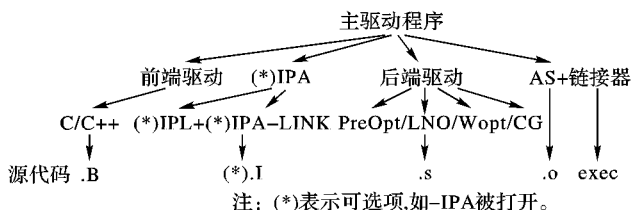


图2 Open64 编译器架构

然而 Open64 编译器框架支持的众多处理器中没有簇的概念,同时对 SIMD 的优化仅保守支持 X8664 后端^[5]。由于 BWDSP 拥有丰富的向量化资源,同时其应用对性能要求极高,因此有必要对 BWDSP100 提供 SIMD 向量化支持。本文在 Open64 原有的循环嵌套优化(Loop Nested Optimization, LNO)中 SIMD 模块的基础上,提出一种支持单双字模式选择的 SIMD 编译优化,能有效解决 BWDSP100 的 SIMD 向量化问题,提高程序的性能。

1 支持单双字模式选择的 SIMD 编译优化

SIMD 编译优化是指编译器将应用程序源代码尽可能地转换为优化过的 SIMD 指令序列,又称 SIMD 向量化^[6](simdization)。目前实现 SIMD 向量化的方法主要有两种:1)基于传统向量化的 SIMD 编译方法^[7-8],主要是通过分析语句循环层的依赖关系来发掘最内层循环中语句的并行性;2)超字并行(Superword Level Parallelism, SLP)向量化方法^[9],主要通过循环展开来将不同循环迭代的语句合并到同一基本块中来实现迭代间和迭代内的数据的 SIMD 向量化。

传统向量化方法不仅出现最早且最为成熟,为许多工业界编译器所采用(如 Open64),因此本文采用 Open64 编译器中原有的 SIMD 向量化框架,充分考虑 BWDSP100 的特点,在其基础上进行了移植和改进,实现了一种支持单双字模式选择的 SIMD 编译优化。

Open64 编译器框架中原有的 SIMD 向量化位于后端的循环嵌套优化中^[10],如图 3 所示,仅仅支持 X8664 处理器后端,由于 BWDSP100 独特的分簇体系结构,同时 SIMD 向量化支持的指令集跟目标处理器有很大的依赖性,因此在原有基础上进行了移植和改进。

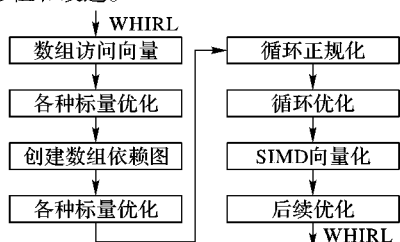


图3 SIMD 在 LNO 中位置

循环嵌套优化阶段使用 High WHIRL 作为中间表示,主要是对程序的循环部分进行优化,通过相应的循环变换和循环优化后,接着进行 SIMD 向量化,向量化后再通过标量优化来提高代码的性能和效率。

针对 BWDSP100 的结构特点,改进后的 SIMD 编译优化算法流程如图 4 所示。

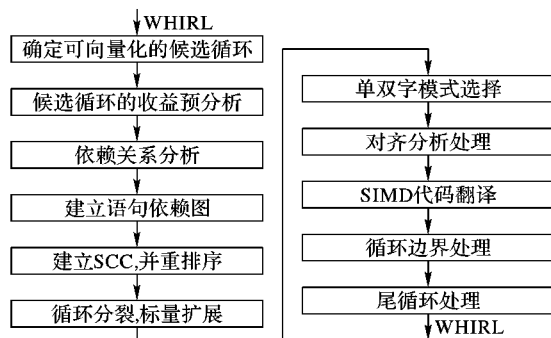


图4 SIMD 编译优化流程

与传统 Open64 中仅支持 X8664 后端部分指令的单字 SIMD 向量化算法相比,BWDSP 编译器采用更为独特的单双字模式选择,通过对候选循环进行相关的分析,根据 BWDSP 指令的特点选择优化性能更高、收益更大的指令模式进行 SIMD 向量化,其他模块也根据目标处理器结构特点进行了相应的移植和改进,具体功能叙述如下,其中确定可向量化的候选循环和预优化合称为预分析。

确定可向量化的候选循环主要用来判断 SIMD 向量化的合法性和合理性,筛选出 BWDSP100 支持的循环结构,排除不易被 SIMD 向量化处理的循环,确定候选循环的条件:

- 1) 循环的迭代次数在进入循环前已知;
- 2) 循环中不能出现函数调用、嵌入式汇编、OpenMP 函数调用、return 或 exit 等退出函数体的语句、goto 或 if 等控制流语句;
- 3) 循环体中不存在不完整的 du 链;
- 4) 循环迭代的次数不能太少;
- 5) 循环体中的运算为 BWDSP100 所支持的 SIMD 指令等。

预优化主要是为后面的 SIMD 向量化代码翻译作准备,包括以下几个阶段:

- 1) 候选循环的收益预分析:分析循环是否取得有效的收益。
- 2) 依赖关系分析:通过一系列的依赖测试(如 delta 测试等)来判断相同基址的数组中是否存在依赖关系,对候选循环进行筛选和程序变换。
- 3) 创建语句依赖图:通过依赖关系分析的结果来创建循环体中数组的层次依赖图。
- 4) 创建强连通分量(Strong Connect Component, SCC),并拓扑排序:在语句依赖图的基础上,将循环体中的语句划分为 SCC,然后对这些分量进行拓扑排序。
- 5) 循环分裂和标量扩展:按照排好序的 SCC 将原循环体分裂成多个小的循环体,同时进行标量扩展来消除标量在循环迭代间的依赖。

6) 单双字模式选择:通过对循环体中数组运算进行判断,根据 BWDSP100 中的单字或双字模式的 SIMD 指令特点,选择优化性能更高的模式。通过分析,最终优化后的结论大

致为:若循环中的增量为1,则选择双字 SIMD 指令进行向量化指令合成;若循环体中的增量为 $i + = C (C > 1)$,则选择单字 SIMD 指令进行向量化指令合成。

7) 对齐分析处理:对循环体中数组引用的连续性等进行分析和处理。

SIMD 代码生成主要是根据所提出的 SIMD 向量化算法来实现具体的代码变换,包括 SIMD 代码翻译、循环边界处理和尾循环处理。

1) SIMD 代码翻译:将可量化的中间表示 WHIRL 转换为带有 SIMD 扩展指令的中间表示 WHIRL。

2) 循环边界处理:修改循环体中索引变量的增量值、开始值和结束值。

3) 尾循环处理:根据循环体中索引变量的结束值和循环体内的向量操作类型确定尾循环,补充那些最后剩余的串行迭代,保证循环变换的完整性和正确性等。

2 SIMD 向量化相关模块的改进

由于 BWDSP100 结构和指令的特殊性, SIMD 向量化之后对其他模块都有较大的影响,因此需要进行相关的改进。其 SIMD 向量化优化涉及的相关模块包括机器描述、指令注释、指令分簇、寄存器分配和汇编指令生成等阶段。各模块主要以下列简单加法为例进行更为形象地叙述:

```
for(i=0; i<N; i+=C) { //N=16, C表示循环增量的间隔
    c[i] = a[i] + b[i];
}
```

2.1 机器描述文件

Open64 编译器框架采用了二次编译的方式设计机器描述文件的架构,对 SIMD 向量化的支持通过填写相应的机器描述模型文件,完成 BWDSP 相应的寄存器模型后,在文件 v11-itanium-extra.knb 和 v12-itanium-extra.knb 中添加需要支持的 SIMD 指令描述,然后编译机器描述文件,生成动态链接库供编译器在运行时使用。

2.2 指令注释

经过 SIMD 向量化优化和全局标量优化(Global Scalar Optimization, WOPT)后,代码生成阶段会调用指令注释将 Very Low WHIRL 扩展成与机器指令一一对应的后端中间表示 CGIR^[11],由于指令注释依赖于目标处理器的机器指令,因此需要对 SIMD 指令注释进行特殊的处理。

1) 单字寻址模式的指令,如 $\{x, y, z, t\} Rs = [Un + = Um, Uk]$ 和 $[Un + = Um, Uk] = \{x, y, z, t\} Rs$,在指令注释时需要通过判断循环中增量的间隔来设置簇间偏移值 Uk 。

例如:当 C 为 2 时,加法操作生成的伪代码如下:

```
xyzRm = [Um + = 8, 2] //Um 为 a 数组基地址
xyzRn = [Un + = 8, 2] //Un 为 b 数组基地址
xyzRm = Rm + Rn
[Uk + = 8, 2] = xyzRm //Uk 为 c 数组基地址
```

2) 双字寻址模式的指令,如 $\{x, y, z, t\} Rs + 1; s = [Un + = Um, Uk]$ 和 $[Un + = Um, Uk] = \{x, y, z, t\} Rs + 1; s$,由于其目的地址有两个,因此在指令注释阶段要通过使用寄存器对(又称 TN 对^[12])来进行处理。

对于双字寻址 load 和 store 指令,通过寄存器对来保存和获取高位寄存器中的值。主要通过哈希表这种特殊的数据结构来支持。

例如:当 C 为 1 时,加法操作生成的伪代码如下:

```
xyzRm + 1; m = [Um + = 8, 1] //Um 为 a 数组基地址
```

```
xyzRn + 1; n = [Un + = 8, 1] //Un 为 b 数组基地址
```

```
xyzRm = Rm + Rn
```

```
|| xyzRm + 1 = Rm + 1 + Rn + 1
```

//“||”表示两条指令在同一时间执行

```
[Uk + = 8, 1] = xyzRm + 1; m
```

//Uk 为 c 数组基地址

3) 对于采用双字模式的 SIMD 运算指令,需要通过识别 SIMD WHIRL 节点,对两个寄存器对中的操作数分别进行运算,将运算的结果再组成寄存器对,以便后续的操作。

4) 对于归约处理和卷积运算,指令注释时首先需要从符号表中取出累加的基数,循环中将运算分布在 4 个簇上同时进行,运算结束后在将结果传输到同一个簇上进行输出,此时簇间的传输需要通过簇间传输指令完成。

2.3 指令分簇

SIMD 指令分簇是把每条 SIMD CGIR 指令指定到 x, y, z, t 四个簇上同时执行,利用四个簇上的资源进行同时计算。具体的指令分簇算法请参考文献[13]。

2.4 寄存器分配

寄存器分配是为每个虚拟寄存器分配其所在簇上的实际寄存器,包括全局寄存器分配和局部寄存器分配。由于 SIMD 指令的局部性, SIMD 指令寄存器分配主要在局部寄存器分配阶段。局部寄存器分配采用的是一种线性分配策略^[14],通过扫描局部基本块中的 SIMD 指令,记录其实际所需分配的寄存器数目,预留出四个簇上编号相同且未使用的寄存器,为后面的 SIMD 指令进行寄存器的分配作好准备。然后通过调用寄存器分配模块进行寄存器分配,如果失败则需要调用相应的溢出处理并重复上述过程,直至分配成功。

2.5 汇编指令生成

对每条 SIMD 指令按照 BWDSP100 的指令格式进行汇编代码输出。通过判断 SIMD 指令中的操作数序号和簇信息,加上相应的簇标识。

3 实验结果和分析

本文在 BWDSP 编译器上实现了该 SIMD 向量化编译优化算法,加入 SIMD 优化选项(-O3)之后,编译器能够自动识别并生成 SIMD 汇编指令,并选取在数字信号处理应用的 DSPstone^[15]测试集中具有典型应用的循环结构程序来验证方案的有效性。

最后在 BWDSP 编译器的开发平台环境 Red Hat Enterprise Linux 5 中对这些测试程序是否进行 SIMD 优化进行测试,采用的性能指标为加速比,即优化前和优化后指令的时钟周期数比值,加速比越大表示优化的效果越好,运用 BWDSP 的调试器 ECS 来测量优化前后生成的汇编指令代码的时钟周期数,结果如表 1 所示。

表 1 优化前和优化后的时钟周期数

运算(N=2048)	优化前/周期数	优化后/周期数	加速比
卷积	12 290	1 798	6.83
双字向量乘	12 288	2 048	8.00
单字向量乘	12 288	3 072	4.00
规约求和	8 194	21 286	6.37
fir	1 668	397	4.20
fft_radix4	297 430	50 417	5.90
lms	2 460	457	5.38
matrix(100×100)	7 989 541	1 736 856	4.60
平均			5.66

从表1可以看出,对于所测试的几种循环结构的双字运算,其加速比均未达到实际的理论值8。主要原因在于双字load后的操作数仍然需要两次运算才能完成,另外簇间传输指令也消耗了其中的部分性能。下面以卷积运算为例进行更加详细的分析。

测试用例中的卷积计算的主要程序代码如下:

```
int sum=0;
for(i=0; i<N; i++) //N=2048
    sum+=a[i]*b[i];
```

未优化前生成的汇编代码如下:

```
_L0_2050:
xrl0=[u5+=1,0] //u5为a数组地址
||xrl7=1
||xrl6=2047
xrl1=[u6+=1,0] //u6为b数组地址
||xrl5=r15+r17
xrl0=r10*r11
xrl4=r14+r10
.code_align 16if xrl5!=r16 B _L0_2050
```

```
_BB3_main:
u6=__sum
[u6+0,0]=xrl4 //u6为sum地址
```

SIMD优化后生成的汇编代码如下:

```
_L0_3330:
xyztr19:18=[u5+=8,1] //u5为a数组地址
||xrl7=8
||xrl6=2047
xyztr21:20=[u6+=8,1] //u6为b数组地址
||xrl5=r15+r17
xyztr11=r19*r21
xyztr10=r18*r20
xyztr12=r10+r11
.code_align 16
If xrl6>=r15 B _L0_3330
_BB3_main:
u7=__sum
xr20=sigma xyztr12
[u7+0,0]=xr20 //u7为sum地址
```

从上面的部分汇编代码中可以看出,对于卷积运算未进行SIMD编译优化前循环中的指令周期为 $5 \times 2048 = 10240$,加上后面的2条指令,总周期大约为10242,表中周期略高,因为还有其他的一些参数设置等指令周期在内;进行SIMD编译优化后其循环内的指令周期为 $6 \times 2048/8 = 1536$,加上后面的几条指令,总的周期大约为1540,通过计算其加速比大约在6.6~6.8,跟实验验证的结果基本一致。

结合上面的分析和表中的实验结果,对于测试用例中几种循环结构的运算,使用本文提出的支持单双字模式选择的SIMD编译优化策略,相对于优化前平均能够取得5.66倍的加速比。

4 结语

本文针对BWDSP100分簇体系结构,提出了一种支持单双字模式选择SIMD编译优化策略,提供了在Open64编译器基础设施上的实现方法,并在BWDSP编译器上对该方案进行了测试和分析,得到的平均加速比为5.66。然而目前的优化算法只能优化最内层数组结构的循环体,暂时还不支持指针循环体和外层循环的SIMD向量化,所以在今后的工作中

还需要进一步研究,尽量对各种不存在依赖关系的循环体都提供支持。

参考文献:

- [1] CETC38. BWDSP100 hardware user manual [R]. Heifei: China Electronics Technology Group Corporation No. 38 Research Institute, 2011: 1-2. (CETC38. BWDSP100 硬件用户手册[R]. 合肥: 中国电子科技集团第三十八研究所, 2011: 1-2.)
- [2] CETC38. BWDSP100 software user manual [R]. Heifei: China Electronics Technology Group Corporation No. 38 Research Institute, 2011: 181-191. (CETC38. BWDSP100 软件用户手册[R]. 合肥: 中国电子科技集团第三十八研究所, 2011: 181-191.)
- [3] SUI Y. Open64 introduction [EB/OL]. [2015-03-17]. <http://www.cse.unsw.edu.au/~ysui/saber/open64.pdf>.
- [4] Open64. Open64 compiler WHIRL intermediate representation [EB/OL]. [2015-03-17]. <http://www.mcs.anl.gov/OpenAD/open64A.pdf>.
- [5] Open64. Using the x86 Open64 compiler suite [EB/OL]. [2015-03-17]. <http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/open64.pdf>.
- [6] SIMD [EB/OL]. [2015-03-17]. <http://en.wikipedia.org/wiki/SIMD>.
- [7] CHENG G, LAM M S. An optimizer for multimedia instruction sets [C/OL]// Proceedings of the 2nd SUHF Compiler Workshop. Stanford: Stanford University, 1997 [2015-01-16]. <http://www-suif.stanford.edu/suifconf/suifconf2/>.
- [8] KRALL A, LELAND S. Compilation techniques for multimedia processors [J]. International Journal of Parallel Programming, 2000, 28(4): 347-361.
- [9] LARSEN S, AMARASINGHE S. Exploiting superword level parallelism with multimedia instruction sets [J]. ACM SIGPLAN Notices, 2000, 35(5): 145-156.
- [10] LI Y. Research on SIMD vectorization and optimization of non-multimedia applications [D]. Heifei: University of Science and Technology of China, 2008: 42-44. (李玉祥. 面向非多媒体程序的SIMD向量化方法及优化技术研究[D]. 合肥: 中国科学技术大学, 2008: 42-44.)
- [11] ZHANG J. Register allocation for machines with connected and partitioned register banks [D]. Beijing: Chinese Academy of Sciences, Institute of Computing Technology, 2005: 92-94. (张军超. 相连多寄存器组体系结构上的寄存器分配技术[D]. 北京: 中国科学院计算技术研究所, 2005: 92-94.)
- [12] XUE L. Research of global register allocation based on Godson 1 [D]. Beijing: Chinese Academy of Sciences, Institute of Computing Technology, 2001: 21-24. (薛丽萍. 基于龙芯1的全局寄存器分配研究[D]. 北京: 中国科学院计算技术研究所, 2004: 21-24.)
- [13] WANG H, HUANG G, WANG X. Research and implementation of propagation cluster algorithm based on BWDSP100 [J]. China Integrated Circuit, 2014, 23(8): 24-28. (王昊, 黄光红, 王向前. 基于BWDSP100的传播簇算法研究与实现[J]. 中国集成电路, 2014, 23(8): 24-28.)
- [14] JIANG J, WANG C, WEI H. An optimisation strategy for local register allocation [J]. Computer Applications and Software, 2013, 30(12): 216-217. (姜军, 王超, 尉红梅. 一种局部寄存器分配的优化策略[J]. 计算机应用与软件, 2013, 30(12): 216-217.)
- [15] Integrated Systems for Signal Processing. DSPstone: a DSP-oriented benchmarking methodology [EB/OL]. [2015-03-12]. <https://www.ice.rwth-aachen.de/fileadmin/publications/Zivojnovic94icspat.pdf>