

融合循环展开的向量VLIW体系结构软件流水研究

汤卓悠^{1,2}, 胡勇华^{1,2}, 程奥博^{1,2}, 刘 澎^{1,2}

(1. 湖南科技大学 计算机科学与工程学院;

2. 湖南科技大学 服务计算与新型软件技术湖南省重点实验室, 湖南 湘潭 411201)

摘要: 基于超长指令字(VLIW)体系结构的处理器具有丰富的寄存器与计算资源, 充分利用这些资源可以极大地提升程序运行速度。软件流水是位于编译器后端的一项优化技术, 可以提高硬件资源利用率。目前的软件流水算法对原循环次数的考虑往往不够周全, 得到的新循环体对寄存器资源的利用率有待提高。因此, 提出一种融合循环展开的软件流水优化方法。该方法首先将循环展开, 然后进行软件流水处理。一方面改进软件流水序言、主体与尾声排列方式, 另一方面分析在不同启动间距下经过软件流水优化后循环主体内执行包的数量, 确定最佳的软件流水启动间距。使用该方法优化后的循环体具有更高的指令调度自由度, 有利于得到运行速度更快的目标代码。经过实验验证, 该方法优化后的代码与仅进行循环展开优化的代码相比, 平均速度提升了5%。

关键词: VLIW 体系结构; 高性能处理器; 编译优化; 软件流水; 指令级并行优化

DOI: 10.11907/rjdk.241244

开放科学(资源服务)标识码(OSID):

中图分类号: TP311

文献标识码: A

文章编号: 1672-7800(2025)006-0095-07



Research on Software Pipelining in VLIW Architecture with Loop Unrolling

TANG Zhuoyou^{1,2}, HU Yonghua^{1,2}, CHENG Aobo^{1,2}, LIU Peng^{1,2}

(1. School of Computer Science and Engineering, Hunan University of Science and Technology;

2. Hunan Key Laboratory for Service Computing and Novel Software Technology, Hunan University of Science and Technology, Xiangtan 411201, China)

Abstract: Processors based on Very Long Instruction Word (VLIW) architecture have rich register and computing resources. Making full use of these resources can bring great performance improvement to the program. The current software pipeline algorithms often do not consider the original factor of loops comprehensively, and the utilization of register resources in the resulting new loop body needs to be improved. This paper proposed a fusing loop unrolling software pipelining optimization method. This method optimizes software pipelining for code that has already been loop unrolling: Improve the arrangement of prologue, body and the epilogue of the software pipelining. Determine the optimal software pipeline initiation interval by comparing the number of execution packages within the loop after software pipelining optimization under different initiation interval. In this paper, software pipelining optimization fusing loop unrolling fully considers the use of register resources and computing units, making the optimized loop have higher instruction level parallelism, which is conducive to obtaining faster running target code. After experimental verification, compared with the code optimized by loop unrolling, the average performance of the basic digital signal processing algorithm has been improved by 5%.

Key Words: VLIW architecture; high performance processors; compilation optimization; software pipelining; instruction level parallel optimization

收稿日期: 2024-03-13

扫描二维码阅读全文:

基金项目: 湖南省自然科学基金项目(2023JJ50019, 2024JJ7172)

作者简介: 汤卓悠(1999-), 男, 湖南科技大学计算机科学与工程学院硕士研究生, 研究方向为DSP编译优化; 胡勇华(1981-), 男, 博士, 湖南科技大学计算机科学与工程学院教授、博士生导师, 研究方向为编译优化技术、嵌入式系统及软件等; 程奥博(1999-), 男, 湖南科技大学计算机科学与工程学院硕士研究生, 研究方向为DSP编译优化; 刘澎(1997-), 男, 湖南科技大学计算机科学与工程学院硕士研究生, 研究方向为DSP编译优化。本文通讯作者: 胡勇华。



0 引言

随着计算机技术的发展,产生了大量的数据存储、传输及交互需求。在物联网、神经网络等领域,数据的计算规模不断增大。因此,基于超长指令字(Very Long Instruction Word, VLIW)架构的具有强大计算功能的数字信号处理器(Digital Single Processor, DSP)应用也越来越广^[5-8]。为了充分利用高性能处理器的向量部件,学者们基于高性能处理器提出了许多适应性算法,以不断增强处理器平台的软件支持能力^[9]。开发高性能处理器相应的函数库对充分发挥处理器的特殊硬件架构优势具有重大意义^[10]。在一般程序中,循环占据了绝大部分执行时间。因此,缩短循环程序的执行时间非常重要。编译优化一直是循环程序优化中的研究热点,在编译层面对算法进行优化往往能够事半功倍^[11-12]。

软件流水是一种用于挖掘循环程序中指令级并行性的方法,也是编译优化中的一项重要技术^[13]。其基本思想是:在资源限制、数据相关、控制相关和周期性条件等的保证下,一个循环可以等价变形,使循环的一次执行可以在前一次执行结束前启动,从而使多个循环体按照流水线方式并行执行。软件流水是一项常用的循环优化技术^[14]。目前,国内外常见的软件流水算法有模调度算法^[15]、Lam算法^[16]、分解式软件流水算法^[17]、线性规划法^[18]等。但是目前大部分高性能编译器中还没有给出有效的向量软件流水方案,并且主流的软件流水优化方法经常会为了找到核心循环而牺牲循环展开次数,对寄存器资源的利用率有待提高。

本文主要设计一种融合循环展开的软件流水优化方法,经过该方法优化后的代码能充分利用硬件的计算功能单元与寄存器资源,得到运行速度更快的目标代码。

1 常规软件流水原理与局限性

软件流水是一种可提高循环程序指令级并行度的方法,其能通过重叠循环之间的部分指令达到优化的效果^[19]。软件流水优化过程中,通过提前启动部分指令,可实现不同迭代之间的指令并行执行。通过重叠不同迭代之间指令形成的流水,可以提高硬件资源利用率,实现同一时间多个功能单元同时工作。软件流水的一般过程如图1所示。其中,图1(a)为优化前的原始循环,图1(b)为优化后循环的流水图。图1(b)多次迭代中同时包含每次迭代且刚好构成一次完整迭代的部分将作为软件流水的核心循环,其中每次迭代在核心循环之上的部分称为软件流水的序言,循环主体内的指令称为软件流水的主体,循环体外尾部的指令称为软件流水的尾声。相邻两次迭代之间的间隔为启动间距。

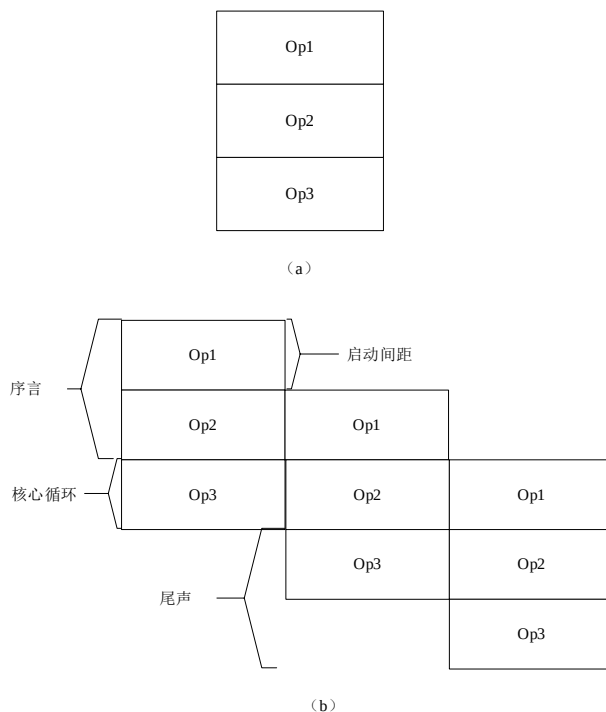


Fig. 1 Basic principle of software pipelining

图1 软件流水基本原理

图1(b)所示的优化方法可等效为在软件流水之前进行循环展开3次,再对3次迭代进行软件流水优化。如图2所示,如果提升循环展开次数,再尝试采用更多次循环进行软件流水,则无法找到同时包含多次迭代的核心循环。在软件流水排布过程中,经常会为了找到核心循环而牺牲循环展开次数。在实际优化中,以较少展开次数进行的软件流水优化结果,可能不如充分进行循环展开而不进行软件流水优化的结果。

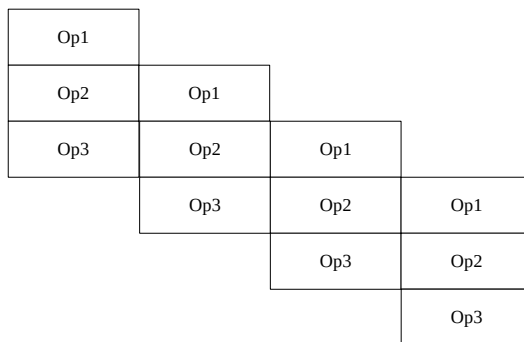


Fig. 2 Pipelining figure after increasing the unrolling factor

图2 提升展开因子后的流水图

2 融合循环展开的软件流水优化方法

2.1 基本流程

本文为了解决循环展开次数与软件流水优化之间存在的问题,设计了一种融合循环展开的软件流水优化方法。先将循环展开至最佳结果,再对展开后的循环进行软

为4、启动间距为1时进行常规软件流水优化的核心循环。按照常规软件流水的优化方法加上任意次数的迭代都无法找到同时包含每次迭代,且正好构成一次完整迭代的核心循环。因此,如果进行常规软件流水处理,循环的展开次数只能为4。本文为了充分利用寄存器资源,尝试对多次迭代进行软件流水优化。如图4所示,针对6次迭代进行软件流水处理,将这6次迭代视为一个大循环。常规软件流水处理的展开次数为4,则认为前4次迭代为一个小循环。逻辑上对不在小循环中的最后两次迭代按照小循环中前两次迭代的方式进行处理,以此形成6次迭代的流水。在小循环的每次迭代中,斜体指令之上的指令构成软件流水的序言。对于不在小循环中的剩余两次迭代,采用与小循环前两次迭代相同的处理方式,其拥有同样的软件流水序言,即将图4最后两次迭代中,在斜体表示指令之上的部分作为软件流水的序言。

本文采用如下方案实现核心循环体的变化:在已识别出循环主体所有向量代码的基础上,根据循环展开次数对向量代码进行复制。针对展开6次的示例1循环构建一个如图5所示的二维指令矩阵 Mtr1。矩阵中的每个元素代表一条指令,矩阵的行数为向量指令条数,列数为循环展开次数,每列的所有指令对应一次完整的循环。

LDW ₁	LDW ₂	LDW ₃	LDW ₄	LDW ₅	LDW ₆
LDW ₁	LDW ₂	LDW ₃	LDW ₄	LDW ₅	LDW ₆
MUL ₁	MUL ₂	MUL ₃	MUL ₄	MUL ₅	MUL ₆
STW ₁	STW ₂	STW ₃	STW ₄	STW ₅	STW ₆

Fig. 5 Instruction matrix Mtr1

图5 指令矩阵 Mtr1

之后,构建一个与 Mtr1 同等大小的辅助矩阵 Mtr2(见图6),里面的值仅为a或b。矩阵 Mtr2 中a所在的位置,对应矩阵 Mtr1 中融合循环展开的软件流水序言部分值的位置。矩阵 Mtr2 中的值是有一定规律的,记列长为x,启动间距为y,小循环迭代次数为n。则在第z列中,a的位置为该列的前 $x-(y*z)\%n$ 项(\%表示取余),该列剩余位置全为b。

a	a	a	b	a	a
a	a	b	b	a	a
a	b	b	b	a	b
b	b	b	b	b	b

Fig. 6 Auxiliary matrix Mtr2

图6 辅助矩阵 Mtr2

生成辅助矩阵 Mtr2 的算法如算法2所示。

算法2 生成辅助矩阵 Mtr2

Input: x, y, m, n
Output: Mtr2
 1. CreateMtx.Mtr2(x, m);
 2. i=0, j=0, index=0;
 3. while(i<m)
 4. while(j<x)
 5. case= (x-y*(index +1)-j);

```

6. if(case>0)
7. MtxInit.Mtr2(i, j, a);
8. else
9. MtxInit.Mtr2(i, j, b);
10. endif
11. j=j+1;
12. index=index+1;
13. if(index==n)
14. index=0;
15. endif
16. end while
17. i=i+1;
18. end while
End function

```

在输入中,x为原始循环体中的向量指令条数,决定矩阵 Mtr2 的列长度;y为启动间距;m为循环展开次数,即大循环的迭代次数,决定 Mtr2 的行长度;n为小循环的迭代次数。在算法开始时调用函数 CreateMtx.Mtr2(x, m),生成一个大小为 x*m 的矩阵 Mtr2,之后用一个二层循环遍历矩阵 Mtr2。外层循环遍历矩阵的行,内层循环遍历矩阵的列。遍历到矩阵某个元素时,判断 case 值是否大于0。若大于0,将a值赋给该元素,否则将b值赋给该元素。算法中 index 值的作用是以小循环为一个周期,以保证不在小循环内的迭代按照指定方式进行赋值。

在生成优化后循环的指令链表时,按照以下步骤可得到优化形式的代码。具体步骤为:①生成软件流水的序言部分:序言由矩阵 Mtr2 中位置 a 对应 Mtr1 中的指令组成;②生成软件流水的主体:主体由矩阵 Mtr2 中位置 b 对应 Mtr1 中的指令、标量偏移指令和矩阵 Mtr2 中位置 a 对应 Mtr1 中的指令组成;③生成软件流水的尾声:尾声由 Mtr1 中对应于矩阵 Mtr2 中位置 b 的指令组成。

生成软件流水序言部分的算法如算法3所示。

算法3 生成软件流水的序言

Input: m, n, headinsList, exinsList
Output: headinsList
 1. while(i<col)
 2. while(j<row)
 3. case=Mtx.Mtr2(irow, icol);
 4. if(case==a)
 5. insStr = Mtxstr.Mtr1(irow, icol);
 6. HeadListAddTail(insStr);
 7. endif
 8. j=j+1;
 9. end while
 10. i=i+1;
 11. j=0;
 12. end while
 End function

在循环开始前插入矩阵 Mtr2 中a元素对应的指令。

该算法为两层循环结构,外层循环遍历矩阵 Mtr1 的行,内层循环遍历矩阵 Mtr1 的列。遍历到某个元素时,假设该元素处在 irow 行、icol 列,则坐标为 (irow, icol)。因为矩阵 Mtr1 与 Mtr2 大小相同,所以遍历到矩阵 Mtr1 的 (irow, icol) 元素时,可以等同于遍历到 Mtr2 的 (irow, icol) 元素。Mtx.Mtr2(irow, icol) 函数的意义为取出矩阵 Mtr2 中 irow 行、icol 列元素的值。若为 a,则调用 Mtxstr.Mtr1(irow, icol),取出矩阵 Mtr1 中 irow 行、icol 列对应的指令,再调用 HeadListAddTail(insStr),将该指令插入软件流水序言部分指令链表的尾部。

同理,生成融合循环展开的软件流水尾声部分的算法与算法 3 类似,仅将插入指令的判断条件从矩阵 Mtr2 中取出的 case 值为 a,改为从矩阵 Mtr2 中取出的 case 值为 b,以及指令插入位置改为软件流水尾声部分指令链表的尾部。

生成融合循环展开的软件流水主体部分的算法如算法 4 所示。

算法 4 生成融合循环展开的软件流水主体

Input: m, n, bodyinsList, exinsList

Output: bodyinsinsList

```

1. while(i<col)
2. while(j<row)
3. case=Mtx.Mtr2(irow, icol);
4. if(case==b)
5. insStr = Mtxstr.Mtr1(irow, icol);
6. BodyListAddTail(insStr);
7. endif
8. j=j+1;
9. j=0;
10. end while
11. i=i+1;
12. end while
13. BodyListAddTail();
14. while(i<col)
15. while(j<row)
16. case=Mtx.Mtr2(irow, icol);
17. if(case==a)
18. insStr = Mtxstr.Mtr1(irow, icol);
19. BodyListAddTail(insStr);
20. end if
21. j=j+1;
22. end while
23. i=i+1;
24. j=0;
25. end while
End function

```

该算法由两个两层循环构成,在第一个两层循环的结束位置调用函数 BodyListAddTail(exinsList)插入标量偏移指令,这两个两层循环均与算法 3 相似。第一个两层循环的作用为插入 Mtr2 矩阵中 b 元素位置对应 Mtr1 矩阵中该

位置的指令,第二个两层循环的作用则是插入 a 元素对应的指令。

2.4 优化后循环内的执行包数量比较

在确定软件流水的序言、主体与尾声后,需要对循环体进行指令调度。指令调度是对本文方法处理后输出代码的进一步处理,不受本文方法的约束。下面对调度后的指令代码进行讨论。2.3 节优化后的代码经过常规的指令调度后,可以得到如图 7 所示的汇编代码片段。其中,第 1~4 条指令为第一个执行包;5~7 条指令为第二个执行包;8~10 条为第三个执行包;10、11 条为第四个执行包。每个执行包视为一个执行周期,处于一个执行包内的指令在同一时间并行执行。需要计算出在当前启动间距下循环体内执行包的总数,例如图 7 中的示例代码片段共有 4 个执行包。

```

SADDA.M1 R59:R58, OR4, OR4
| SADD.M2 R34, R36, R45
| VADD.M2 VR40, VR42, VR55
| VSTW VR43, *+AR0[OR2]
SADDA.M2 R59:R58, OR2, OR2
| SADD R34, R45, R45
| VSTW VR55, *+AR0[OR3]
SADDA.M1 R59:R58, OR3, OR3
| SADD R34, R45, R45
| VADD.M1 VR41, VR54, VR60
SADD R34, R45, R36 ; <1
| VSTW VR60, *+AR0[OR0]

```

Fig. 7 Assembly code snippet

图7 汇编代码片段

在不同的启动间距下,循环体内部执行的迭代次数相同。因此,执行包数量更少的循环具有更高的运行效率。比较了不同启动间距下生成汇编代码的执行包总数后,将获得最佳效果,即将执行包总数最少的优化结果作为融合循环展开的软件流水优化最终结果。

3 实验结果与分析

本文在 FT-M7002 处理器上,对基础循环、仅进行循环展开优化的循环和融合循环展开的软件流水优化循环性能进行比较。

3.1 测试平台体系结构与参数

本文测试所用体系结构基于飞腾系列芯片 FT-M7002 平台,该系列处理器可被广泛运用于雷达、电子对抗、精确制导、图像处理、通信保障等信号处理领域^[20]。FT-M7002 处理器是一款基于 VLIW 体系结构的高性能数字信号处理器(DSP),其采用 40 nm 工艺、三级存储结构,每个 DSP 核心有着 32 KB 的标量存储空间(Scalar Memory, SM)和 512 KB 的向量存储空间,全部计算共享 2 MB 的全局 cache,核外还有 32 GB 的大容量 DDR(Double Data Rate)存储空间。

FT-M7002 的单核部分结构如图 8 所示。内部 DSP 核

基于 VLIW 结构,拥有向量处理单元(Vector Processing Units, VPU)和标量处理单元(Scalar Processing Units, SPU),且 SPU 可以和 VPU 通过标向量共享寄存器(Scalar Vector Shared Register, SVR)交换数据^[21]。测试函数使用的向量计算指令都是在 VPU 中计算的。

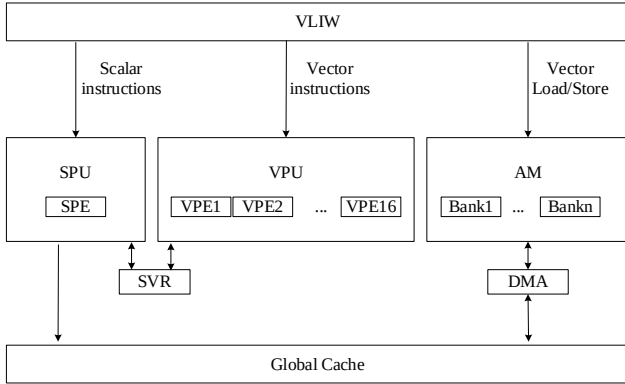


Fig. 8 Partial structure of FT-M7002

图8 FT-M7002部分结构

在飞腾平台的 FT-M7002 开发板上实现本文的优化方法,调用定时器函数测试优化后程序的执行时间,并使用多组数据量进行比较。实验参数设置如表 1 所示。

Table 1 Experimental platform parameter settings

表1 实验平台参数设置

参数	FT-M7002
主频/GHZ	1.00
L1D/(KB/core)	32
AM/KB	512
编译器	M7002 编译器

3.2 测试结果

以数值比较算法 vsip_vleq_d 和 vsip_vlgt_d 的关键循环部分为例,选取不同数据量来运行算法。在不同的启动间距(II)下,两个算法循环主体内的执行包数量不同。两个算法启动间距与执行包数量(Num)的对应关系如表 2、表 3 所示。其中,II=0 的情况为仅进行循环展开优化得到的结果。由表可知,当启动间距为 2 时进行软件流水优化,vsip_vleq_d 可以得到最佳结果;当启动间距为 1 时进行软件流水优化,vsip_vlgt_d 可以得到最佳结果。

Table 2 Execution packets number of vsip_vleq_d under different initiation interval

表2 vsip_vleq_d 不同启动间距下的执行包数量

II	II=0	II=1	II=2	II=3	II=4	II=5
Num	37	35	33	36	38	40

Table 3 Execution packets number of vsip_vlgt_d under different initiation interval

表3 vsip_vlgt_d 不同启动间距下的执行包数量

II	II=0	II=1	II=2	II=3	II=4	II=5
Num	35	32	35	38	36	39

对 vsip_vleq_d 与 vsip_vlgt_d 分别以 2 和 1 为启动间距进行融合展开的软件流水优化后,在 FT-M7002 处理器上的运行结果(时钟周期数量)如表 4、表 5 所示,对应的柱状图如图 9、图 10 所示。其中,横坐标为数据量,按数据占 AM 空间大小均匀增长;纵坐标为时钟周期数量,1 个时钟周期等于 16 个标准节拍。

Table 4 Clock cycle counts of vsip_vleq_d

表4 vsip_vleq_d 时钟周期数量

数据量	原始循环	循环展开	软件流水
3 007	512	351	332
5 000	708	391	372
8 009	1 002	451	432
12 006	1 393	533	516
21 501	2 319	762	727
32 141	3 358	972	948
40 023	4 343	1 418	1 372
52 319	5 612	1 893	1 812

Table 5 Clock cycle counts of vsip_vlgt_d

表5 vsip_vlgt_d 时钟周期数量

数据量	原始循环	循环展开	软件流水
6 006	976	396	379
10 007	1 976	498	475
30 005	3 652	948	905
50 006	5 109	1 716	1 613
70 008	7 876	2 236	2 117
100 009	8 921	3 176	3 020
120 012	9 823	3 654	3 505

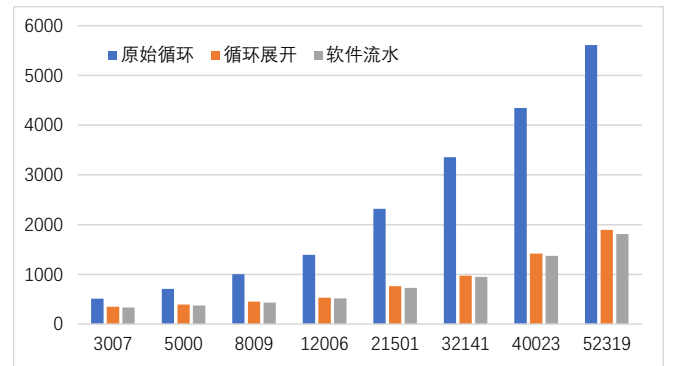


Fig. 9 Comparison of vsip_vleq_d clock cycle counts

图9 vsip_vleq_d 时钟周期数量比较

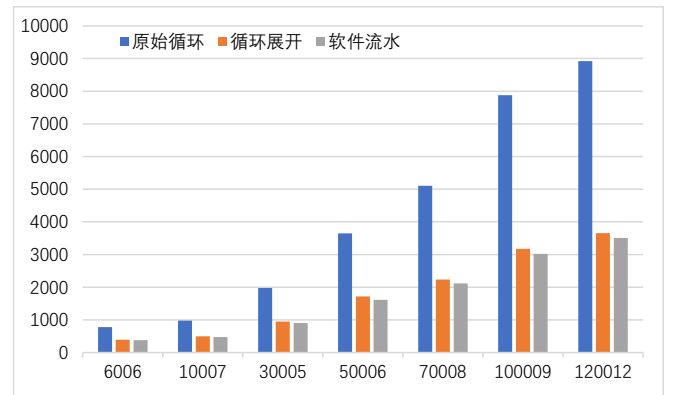


Fig. 10 Comparison of vsip_vlgt_d clock cycle counts

图10 vsip_vlgt_d 时钟周期数量比较

3.3 性能分析

从实验结果可以得出,仅进行循环展开优化后的代码,在原循环的基础上,vsip_vleq_d平均能达到2.57的加速比,vsip_vlgt_d平均能达到3.19的加速比;经过本文方法优化后,vsip_vleq_d的平均加速比为2.68,vsip_vlgt_d的平均加速比为3.34;与仅进行循环展开的代码相比,软件流水优化后代码vsip_vleq_d的平均加速比为1.04,vsip_vlgt_d的平均加速比为1.05。融合循环展开的软件流水在先循环展开的基础上,综合考虑了硬件功能单元的利用率与寄存器资源的使用情况,使优化后的代码具有更高的指令级并行度,运行速度更快。与原始循环相比,速度提升了200%以上;与仅进行循环展开的代码相比,速度提升了5%。

4 结语

本文提出一种融合循环展开的软件流水优化方法,对先循环展开的代码进行软件流水优化,提高了对寄存器资源与计算功能单元的使用率,增强了代码的指令级并行度。经过实验验证,与未经优化的原始循环相比,该方法的速度提升了200%以上;与仅进行循环展开优化的代码相比,平均速度提升了5%。后续将研究更先进的算法来计算软件流水的序言、主体与尾声,以期得到更好的优化结果。

参考文献:

- [1] LIANG W, YANG Y, YANG C, et al. PDPChain: a consortium block-chain-based privacy protection scheme for personal data[J]. IEEE Transactions on Reliability, 2023, 72(2): 586-589.
- [2] ZHANG S W, HU B, LIANG W, et al. A caching-based dual K-anonymous location privacy-preserving scheme for edge computing[J]. IEEE Internet of Things Journal, 2023, 10(11): 9768-9781.
- [3] KANG L, CHEN R S, XIONG N, et al. Selecting hyper-parameters of Gaussian process regression based on non-inertial particle swarm optimization in Internet of Things[J]. IEEE Access, 2019, 7: 59504-59513.
- [4] ZHANG J Q, LI B, ZHAO B. Blockchain-based agricultural IoT AI big data platform[J]. Internet of Things Technology, 2024, 14(2): 121-122, 126.
张继群,李斌,赵斌. 基于区块链的农业物联网AI大数据平台[J]. 物联网技术, 2024, 14(2): 121-122, 126.
- [5] PSIAKIS R, KRITIKAKOU A, SENTIEYS O. Dynamic fault-tolerant VLIW processor with heterogeneous function units[J]. Microprocessors and Microsystems, 2022, 93: 104564.
- [6] FERREIRA L, MALKOWSKY S, PERSSON P, et al. Design of an application-specific VLIW vector processor for ORB feature extraction[J]. Journal of Signal Processing Systems, 2023, 95(7): 863-875.
- [7] MUTO V, ANDREOZZI E, CAPPELLI C, et al. Real-time implementation of a frequency shifter for enhancement of heart sounds perception on VLIW DSP platform[J]. Electronics, 2023, 12(20): 4359.
- [8] SUN L. Design of multi-channel subband filter banks based on DSP[J]. Radio and Television Technology, 2023, 50(3): 90-94.
- [9] 孙亮. 基于DSP的多通道子带滤波器组设计[J]. 广播与电视技术, 2023, 50(3): 90-94.
- [9] WANG M Y, CHAI X N, CHEN Y, et al. Optimization implementation of image median filter algorithm for YHFT-M7002 platform[J]. Computer Applications and Software, 2023, 40(9): 205-210, 241.
王梦园,柴晓楠,陈云,等. 面向YHFT-M7002平台图像中值滤波算法的优化实现[J]. 计算机应用与软件, 2023, 40(9): 205-210, 241.
- [10] MO S F, ZHOU Z F, HU Y H. Transplantation and optimization of complex field row vector matrix multiplication based on FT-M7002[J]. Computer Science, 2023, 50(S2): 839-844.
莫尚丰,周振芬,胡勇华. 基于FT-M7002的复数域行向量矩阵乘法移植与优化[J]. 计算机科学, 2023, 50(S2): 839-844.
- [11] LI W, WANG R. Research on optimization methods for compilation of embedded communication software[J]. Application of Microcontrollers and Embedded Systems, 2023, 23(8): 83-86, 91.
李维,王瑞. 嵌入式通信软件编译优化方法研究[J]. 单片机与嵌入式系统应用, 2023, 23(8): 83-86, 91.
- [12] ZHOU Q W, LI J N, ZHAO R C, et al. Compilation optimization of DCU-oriented OpenMP thread scheduling[EB/OL]. <https://iopscience.iop.org/article/10.1088/1742-6596/2558/1/012003/meta>.
- [13] LIU Y Y, ZHANG Z W. Instruction level parallel architecture design based on SIMD architecture[J]. Electronic Design Engineering, 2017, 25(21): 152-156.
刘有耀,张仲伟. 基于SIMD体系结构的指令级并行结构设计[J]. 电子设计工程, 2017, 25(21): 152-156.
- [14] LU H S, HU Y H, WANG S Y, et al. Research on hybrid resource heuristic loop unrolling factor selection method for vector DSP[J]. Computer Science, 2022, 49(S1): 777-783.
陆浩松,胡勇华,王书盈,等. 向量DSP的混合资源启发式循环展开因子选择方法研究[J]. 计算机科学, 2022, 49(S1): 777-783.
- [15] WANG X Q. Compilation design and optimization of soul core DSP[D]. Hefei: Hefei University of Technology, 2015.
王向前. 魂芯DSP编译设计与优化[D]. 合肥: 合肥工业大学, 2015.
- [16] LAM M S. Software pipelining: an effective scheduling technique for VLIW machines[J]. ACM SIGPLAN Notices, 2004, 39(4): 244-245.
- [17] BENABID A, HANEN C. Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays[J]. Journal of Scheduling, 2011, 14(5): 511-522.
- [18] XUE H. Research on robustness validation of neural networks based on deep learning[D]. Shanghai: East China Normal University, 2024.
薛昊. 基于深度学习的神经网络鲁棒性验证研究[D]. 上海: 华东师范大学, 2024.
- [19] ZHANG R G. Research and implementation of soul core DSP software pipeline framework[D]. Hefei: University of Science and Technology of China, 2017.
张仁高. 魂芯DSP软件流水框架的研究与实现[D]. 合肥: 中国科学技术大学, 2017.
- [20] WANG Y H, CHEN L, CHANG L, et al. Advancing DSP into HPC, AI, and beyond: challenges, mechanisms, and future directions[J]. CCF Transactions on High Performance Computing, 2021, 3(1): 114-125.
- [21] HU Y H, ZHANG X, WANG S Y, et al. Research on global register allocation for code containing array-unit dual-usage register names[J]. Concurrency and Computation: Practice and Experience, 2023, 35(19): e7519.

(责任编辑:黄健)