文章编号:1001 - 2486(2011)04 - 0102 - 05

一种基于 VLIW DSP 架构的高性能取指流水线*

杨 惠,陈书明,万江华
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:以超长指令字(VLIW)数字信号处理器(DSP)作为平台,针对现有提高单线程取指流水线效率的方法中存在的弊端,提出了一种高性能的取指流水架构。该架构支持无效取指的检测与作废,从而降低不必要的 cache 访问,减少取指流水停顿周期,该结构还引入专用硬件支持编译调度的循环软流水,有效提高指令并行性,降低代码存储空间,由此释放出的单线程取指流水线的空闲周期约达 46.34%。实验结果表明,相比优化前的取指流水而言,代码空间压缩约 11.93%,执行周期缩短约 8.67%,cache 访问次数下降约 12.84%,指令 cache 暂停周期缩短约 7.86%,处理器单线程的指令吞吐率平均提高约 11.7%。

关键词:数字信号处理器;无效取指;软件流水;循环缓冲
中图分类号:TP368.1 **文献标识码**:A

A High-performance Fetch Pipeline Based On the VLIW DSP Architecture

YANG Hui, *CHEN Shu-ming*, *WAN Jiang-hua*
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: For the drawbacks existent in single-thread fetch pipeline to improve the efficiency, a high-performance fetch pipeline structure is proposed in this paper based on the platform of the VLIW digital signal processor (DSP). It can support the detection and void for the invalid fetch, bypass for the missing fetch, which reduces the unnecessary cache access and fetch pipeline stall. The structure also inducts dedicated hardware which supports the software pipeline of scheduled compilation to improve the parallelism of instruction. It reduces the code memory space, and the idle cycles of released single-threaded pipeline is reached up to about 46.34%. Compared with the fetch pipeline before optimized, experiment results show that the code storage space is reduced about 11.93%, the average execution cycle is shortened about 8.67%, the cache access times is decreased about 12.84%, the suspension period of instruction cache is shortened about 7.86%, and the single-threaded instruction throughput of processor is increased by 11.7%.

Key words: digital signal processor; invalid instruction fetch; software pipeline; loop buffer

超长指令字 VLIW 结构,将不同的操作打包成长指令字,通过编译器调度,分派到多个功能单元,获取指令并行性,广泛应用于高性能 DSP 中。随着应用需求对 DSP 的性能和集成度要求的急剧提升,目前已提出支持进程级并行(SCMP)[1]、线程级并行(SMT)[2]、细粒度多流出(EPIC、VLIW)[3]、标向量结合[4]等一系列通过开发多种形式并行,以提升系统吞吐量的体系结构技术。从运算资源的角度,采用上述各类技术使得更多的指令并行执行,提高了运算资源利用率和系统吞吐率,但从指令存储角度,更多的指令派发使得指令存储的访问更加频繁,竞争更加激烈,导致更多的指令缓冲扑空带来的停顿与延迟[5]。其次,整个系统的吞吐量是整体性能的重要指标,但实时系统更加关注单任务的执行能力[6]。因此,优化单线程的取指流水,能够有效提升 DSP 整体性能。为增强取指效率,现有研究从体系结构、编译等[10-14]各种软硬件层面上,提出了诸多优化方法,如延迟分支技术[7],指令字压缩技术[8],跨指令字边界派发技术[9],循环的软硬件调度技术等。针对现有技术存在的弊端,本文提出了一种高性能取指流水架构,该结构引入无效取指的动态监测和作废机制,减少无效取指的 cache 访问。引入专用的硬件循环缓冲,装载循环的一次迭代,配合编译器调度,大大降低代码空间和 cache 的访问次数,实现了硬件支持循环的软流水,增强指令

* **收稿日期**:2010 - 11 - 29
基金项目:国家科技重大专门资助项目(2009ZX01034 - 001 - 006)
作者简介:杨惠(1987—),女,博士生。

并行性。

为了验证并评估所提出的取指流水架构的性能,基于 TI 公司的 TMS320C64xx 处理器内核原型,设计并实现了具有改进型取指流水线架构的 VLIW DSP。实验结果证明,本文提出的取指流水线架构,能够有效提升数字信号处理器性能。

1　高性能取指流水架构

1.1　无效取指的作废机制

本文采用的内核原型的基本流水线架构如图 1 所示。引入无效取指作废率(Invalid Instruction Cancel Rate,IICR)对无效指令的作废进行量化,计算方式如式(1)所示,式中 $T_{invalid}$ 表示在程序执行过程中作废无效指令所节约的周期数,T_{total} 表示总的执行周期数。

$$IICR = \frac{T_{invalid}}{T_{total}} \tag{1}$$

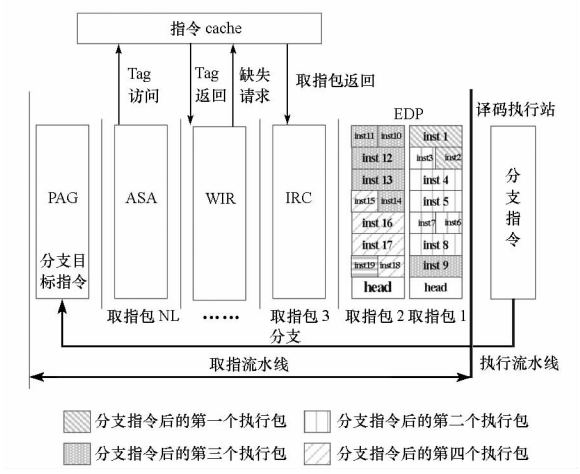

图 1　内核原型的基本流水线架构
Fig.1　Basic pipeline of the core prototype

经分析,从分支目标进入取指流水线 PAG 站开始,到分支目标指令进入执行站执行,这期间的取指流水线均受到指令字缺失的影响。因此无效取指应分三种情况进行检测和消除。

第一种情况,作废无效指令的访问请求。发生在程序地址产生站 PAG 和发送指令字地址访存站 ASA,指令字缺失尚未发生,分支目标可直接作废访存地址,从而减少无效访问。

第二种情况,对于已经发送无效指令访问请求的指令字,若发生指令 cache 读缺失,则作废缺失访问请求。当分支目标指令到达 WIR 站时,若 WIR 站对应指令字发生缺失,而 IRC 站中的指令字未发生缺失或缺失已返回。那么,cache 返回的第一个缺失指令字,即 WIR 站对应的指令字应作废。

当分支目标到达 WIR 站,WIR 站的指令字没有发生缺失,那么只需覆盖掉当前站对应的指令字地址,并继续向 IRC 站流动。若此时 IRC 站的无效指令字发生缺失,取指流水线应向指令 cache 发出作废缺失请求,并覆盖 IRC 站的地址,假如此时 IRC 站的无效指令字已经缺失返回,那么分支目标指令将直接作废指令字。

当分支目标地址进入 WIR 站时,WIR 和 IRC 站对应的无效指令字均已发生缺失。取指流水线首先发出作废第二个缺失请求的信号,并覆盖 WIR 站的指令地址。当分支目标指令继续流动,进入 IRC 站时,分两种情况讨论,若 IRC 的缺失指令字已返回,作废掉已经返回的指令字,并将发送的作废第二个缺失请求的信号,转换成作废第一个缺失请求的信号。若 IRC 站对应的指令字仍然处于缺失状态,那么取指流水线在 IRC 站也发送作废缺失请求的信号。

第三种情况,避免相同缺失指令的缺失请求重复发送。当连续两个地址读同一个指令字时,若连续发出两次读缺失请求,会造成更多的 cache 访问,因此应作出缺失变命中的判断,阻止第二个指令字缺失请求的发送。

综上所述,在取指流水线中引入无效取指的检测作废控制逻辑,该控制逻辑如图 2 中的状态机所示,共包含 7 种状态,各个状态的具体含义如表 1 所示。无效取指的检测作废机制可显著地提高无效取指作废率。

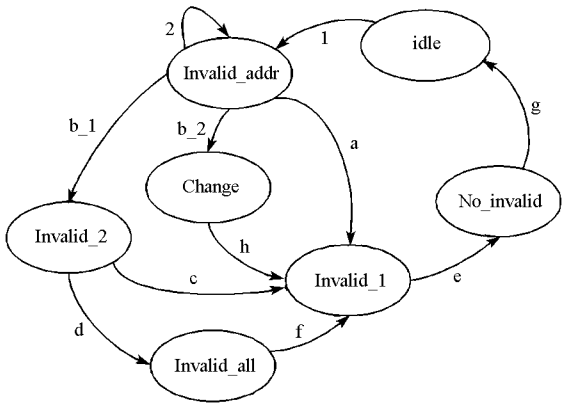

图 2　无效取指作废状态机
Fig.2　Void state machine of the invalid fetch

各个状态间的转换条件如下:

1 表示分支目标进入流水线;2 表示流水线不阻塞,且取指流水线没有缺失请求;a 表示取指流

表 1 作废状态机的各状态

Tab.1 Status of the void state machine

状态名称	状态含义
idle	不发生无效指令字的作废
Invalid _ addr	早期作废无效指令字的地址
Change	连续两个无效指令字的缺失便命中处理
No _ Invalid	缺失返回的指令字无需作废
Invalid _ 1	作废第一个缺失指令字
Invalid _ 2	作废第二个缺失指令字
Invalid _ all	作废连续两个缺失指令字

水线有一个缺失请求,并且分支目标到达 WIR 站或到达 IRC 站;b _ 1 表示取指流水线有两个缺失请求,并且分支目标到达 WIR 站,两个缺失请求地址不同;b _ 2 表示取指流水线有两个缺失请求,并且分支目标到达 WIR 站,两个缺失请求地址相同;c 表示取指流水线中已经有一个缺失请求被响应,并返回了缺失指令字,此时分支目标指令到达 IRC 站;d 表示取指流水线中有两个缺失请求,并且分支目标到达了 IRC 站;e 表示有一个缺失请求被响应;f 表示有一个缺失请求被响应;g 表示分支目标流出取指流水线;h 表示流水线不阻塞。

1.2 硬件实现循环的软流水机制

现有研究中引入的循环分离缓冲,是一个拥有特定映射方式的指令缓存,用于存储部分或全部的循环指令[11-13],从而有效降低代码量和访存次数,减小分支引发的开销。但这种循环缓冲并不能使得不同循环迭代、不同层次循环之间的指令并行执行,而仅仅实现了单次循环迭代间并行和不同迭代间的串行执行。图 3 指出了本文提出的硬件循环缓冲在流水线中所处的环节位置。它位于取指流水与执行流水之间,可动态缓存循环指令。

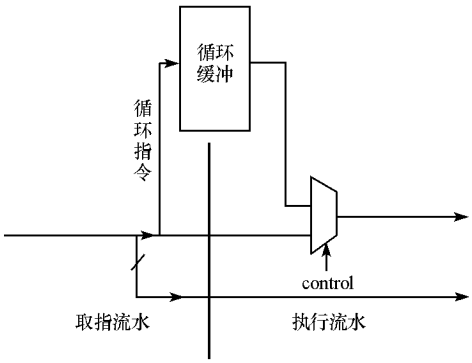

图 3 循环缓冲所处的流水线位置

Fig.3 Pipeline position of the loop buffer

在指令执行过程中,循环缓冲动态获取取指流向执行流派发的循环指令。循环指令派发往执行流的同时,存入循环缓冲。当循环体流入执行流一遍时,循环缓冲也已经存储了整个循环体的一次迭代。而上一个周期存入循环缓冲的指令,会在下一周期与取指流派发的指令一同流入执行流水线。当完成了循环体一次迭代的载入后,关闭取指流水线,流入执行站的所有指令,可完全由循环缓冲派发,直至循环核结束,打开取指流,使其流出的指令与循环缓冲流出的指令一同流入执行站。循环载入完成后,指令可全部由循环缓冲提供,因此,若处理器采用多线程技术,则循环体内的指令可用于提供某一固定线程的执行,其他辅线程的指令可由取指流水线单独调度派发,从而更大程度地提高指令并行性。

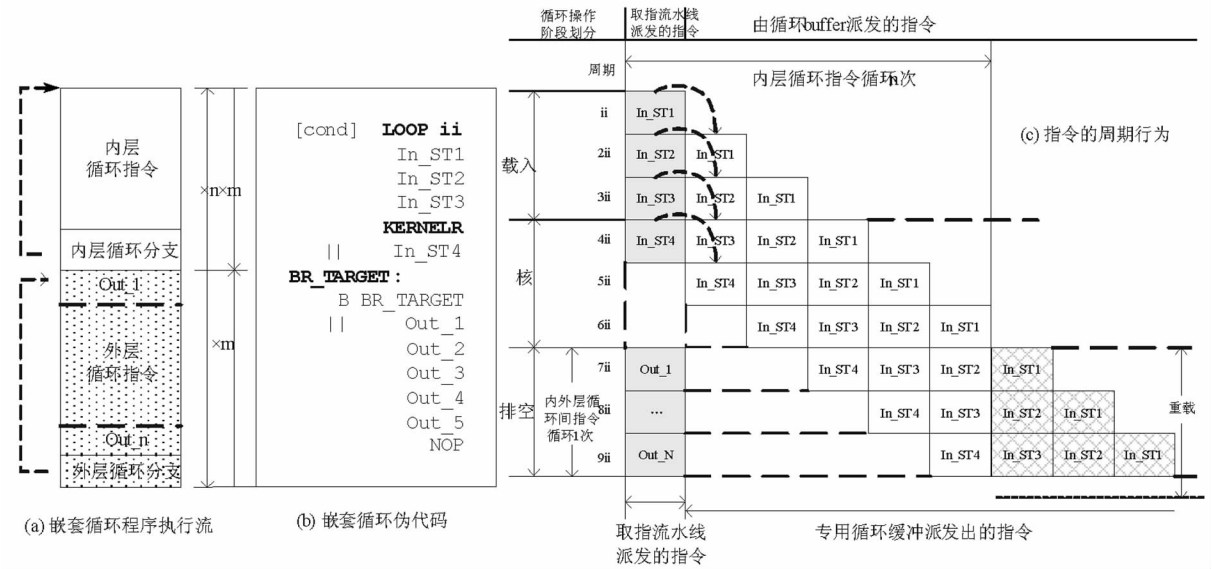

图 4 嵌套循环程序软件流水执行示意图

Fig.4 Diagram of the software pipeline for the nested loop program

图 4 给出了嵌套循环指令的执行流和其伪代码表示,并描述了嵌套循环指令在流水线中的周期行为。其中图 4(b)描述了一段嵌套循环的伪代码。内层循环遍数为 n 次,循环体启动间距为 ii(initiation interval),内层循环包括 4 个执行包。外层嵌套循环执行遍数为 m 次,外层循环的执行包放入外层分支跳转指令的延迟槽内。由伪代码看出当 cond 条件为真时,嵌套循环会重复执行 m 次。

此伪代码在流水线中的周期行为如图 4(c)所示。图 4(c)中的每一行是当前周期派发到执行流水的指令。专用硬件循环缓冲,用于装载嵌套循环内层的循环指令,外层循环与内层循环间的指令,将全部调度在内层循环指令与外层分支指令之间,如图 4(a)所示。外层循环的向后分支的分支目标为 Out _ 1,作为外层循环的首条指令地址。内层循环指令通过硬件实现的软件流水进行派发执行,循环核结束后(6ii 周期后),进入到排空阶段,如图 4(c)所示。在周期 7ii 时,将由取指流水线同时参与派发指令,取指流水线派发的指令为外层循环指令。此时若判断出嵌套循环的条件满足,则触发再一次的内层 n 次循环。于是周期 7ii 时,循环缓冲同时进行排空和重载操作,内层循环指令不必再重新从取指流水线取指,只需利用一个重载过程就可以恢复执行,将已经存入循环缓冲的指令重新置为有效,与外层循环指令并行派发执行。到达周期 9ii,即外层循环派发完毕,外层循环的分支目标地址为 Out _ 1,此时的外层循环分支目标指令将在取指流水线中等待,直到内层循环操作完成后再次派发,重复图 4(c)所示的过程,直至嵌套循环条件不满足,执行第 m 次后结束。由此可以看出,周期 ii 至 6ii,循环缓冲硬件实现了循环不同迭代间的并行执行,周期 7ii 至周期 9ii,实现了循环不同层次间和循环不同迭代间的并行执行。

2　实验结果及分析

2.1　评估实验平台

为了有效评估本文所提出的流水线架构的性能及硬件代价,将 TI 公司的 c64 系列处理器内核的取指流水做为基本模型,在 RTL 级描述并实现了具有改进型取指流水线架构的 VLIW DSP。在 130nm 工艺下,采用 Synopsys 公司的 Design Compiler 工具进行综合。特性参数由综合结果提取。设计目标频率为 500MHz。

特别地,为获取高性能流水线架构的性能统计,本文在 Cadence 公司的 NC _ Verilog 模拟环境下,搭建用于统计性能的时钟精确的处理器仿真模型,仿真模型按照处理器真实的运行环境来设计,并将监控程序加入其中。仿真器配置成两种不同的架构模型,包括:(1)基本的单线程取指流水架构模型;(2)上文所描述的优化后单线程取指流水架构模型。

最后,评估程序采用的是 DSP/IMG 库,即在 TI 公司 CCS3.3 集成的编译器上编译,并使用最优化的选项。TI 的编译器在处理 DSP 程序时,可实现很高的并行度,在一定程度上降低了分支执行过程中指令作废的比例。

2.2　实验结果及分析

综合结果表明,优化所增加的取指流水线面积约为 72 939μm^2,占基准取指流水线硬件资源的 22.5%,其中循环缓冲的面积约占 19.7%。优化所增加的取指流水线面积,约占内核总面积的 4.4%。

由图 5 对无效指令作废率进行了数值计算和统计,可以看出,采用无效取指的作废机制,使得无效指令作废率 IICR 达到 2.5% ~ 11.4%。

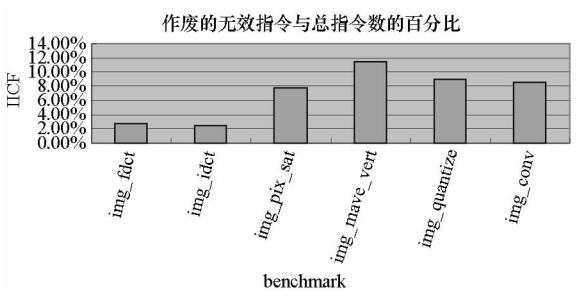

图 5　无效指令作废率统计

Fig.5　Void rate of invalid instruction

图 6 描述了取指流水与循环缓冲的指令派发百分比。可以看出,由循环缓冲派发的指令周期数占据了相当大部分的程序执行时间。对于 DSP/IMG 库中典型的程序 FDCT、IDCT、量化等,指令由循环缓冲派发的周期数可占总执行周期的

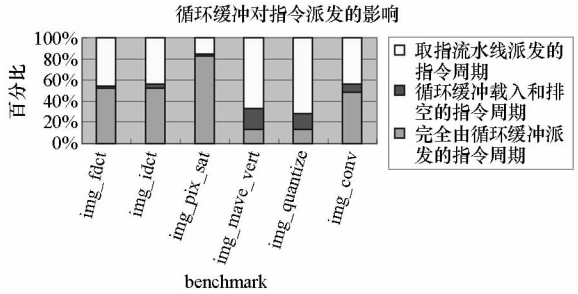

图 6　取指流水与循环缓冲的指令派发百分比

Fig.6　Instruction dispatch of the fetch pipeline and loop buffer

33.1% ~ 86.8%,从而有效减少了指令 cache 访问。取指令流水线可被释放平均约为 46.34% 的空闲周期,有利于开发线程级的并行。

图 7 显示对程序存储空间、程序总执行时间、cache 访问次数和 cache 停顿周期这四种性能的统计结果,整体性能大幅提升。

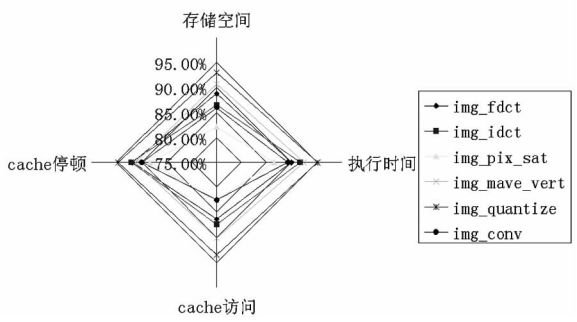

图 7 高性能取指流水架构提升性能的统计图

Fig.7 Performance improvement statistics for the high-performance fetch pipeline

最后,为了对本文所提出的高性能流水线架构和现有的基本流水线架构的吞吐率进行比较,论文采用每周期执行指令数(Instruction Per Cycle, IPC)作为评估指标。可以看出,与基本流水线架构相比,由图 8 可知,改进的流水线架构使 IPC 百分比平均提升约 11.7%。

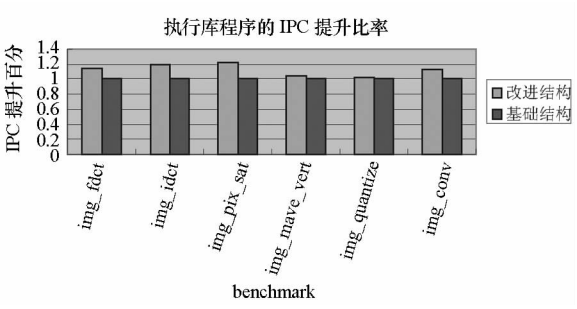

图 8 IPC 的提升率

Fig.8 IPC improvement rate

3 结束语

本文提出了一种高性能取指流水线架构,该架构引入了无效指令的检测作废机制,用于消除无效指令取指访问带来的开销,采用专用硬件循环缓冲,配合编译器调度,完成循环的软件流水,大大削减代码量,减少 cache 访问,大幅提高指令并行性,释放取指流水线。基于该架构,设计了一款具有增强型取指流水线架构的 VLIW DSP,采用 130nm 工艺库综合结果表明,最坏情况下,该 DSP 得取指流水线可工作于 500MHz,工作于峰值频率时,取指流水线的功耗约为 78mW,总面积约 321 518μm^2。在整个内核的硬件开销中,仅需要约 4.4%用于支持取指流水线的优化。实验结果表明,针对高性能数字信号处理器,本文提出的流水线优化策略可大幅度提高整体性能,并使得 IPC 平均提高约 11.7%。

参 考 文 献:

[1] Hu Y L, Wang Y M. Task Scheduling and Management in Single-Chip Multi-Mrocessor System [C]//International Conference on Electronic Packaging Technology and High Density Packaging, 2008(7): 1 - 4.

[2] Chishti Z, Vijaykumar T N. Optimal Power/Performance Pipeline Depth for SMT in Scaled Technologies[J]. IEEE Transactions on Computers, 2008,57(1): 69 - 81.

[3] Deng Q, Zhang M. A Parallel Infrastructure on Dynamic EPIC SMT[C]//Algorithms and Architectures for Parallel Processing, 2007,4494: 165 - 176.

[4] Rowen C, Nuth P. A DSP Architecture Optimized for Wireless Baseband[C]//Proceedings of the 11th international conference on System-on-chip, Tampere, Finland, IEEE Press, 2009: 151 - 156.

[5] Wan J H. A Method to Improve the Throughput of the Instruction Fetch Unit in SMT VLIW Processors[J]. Computer Engineering & Science, 2007, 29(6): 97 - 101.

[6] Shen Z. Architecture Design of Simultaneous Multithreading VLIW DSP[J]. Acta Electronica Sinica, 2010,38(2): 352 - 358.

[7] Jin T S, Dong K, Kim, et al. Static Branch Prediction Method And Code Execution Method For Pipeline Processor, And Code Compiling Method For Static Branch Prediction [P]. US: 20100205405, 2010.12.8.

[8] Shen Z, He H. Architecture Design of a Variable Length Instruction Set VLIW DSP[J]. Tsinghua Science & Technology, 2009,14(5): 561 - 569.

[9] Texas Instrument Incorporated. Very Long Instruction Word Microprocessor with Execution Packet Spanning Two or More Fetch Packets with Pre-dispatch Instruction Selection from Two Latches According to Instruction Bit[P]. US :7039790, 2000.10.31.

[10] Knijnenburg P M W. Branch Classification to Control Instruction Fetch in Simultaneous Multithreaded Architectures[C]//Proc of the Int'l Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, 2002: 67 - 76.

[11] Chiu J C. A Novel Instruction Stream Buffer for VLIW Architectures[J]. Computers & Electrical Engineering, 2010,36(1):190 - 198.

[12] Jayapala M. Clustered Loop Buffer Organization for Low Energy VLIW Embedded Processors[J]. IEEE Trans. on Computers, 2005,54(6):672 - 683.

[13] Rivers J A. Reducing Instruction Fetch Energy with Backwards Branch Control Information and Buffering[C]//Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003,8:25 - 27.

[14] Datta K, Murphy M, Volkov V. Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures[C]// High Performance Computing, Networking, Storage and Analysis, 2008:1 - 12.