DOI:10.11918/j.issn.0367-6234.201508011

M5-EDGE 分布式取指模型设计

张 超,喻明艳

(哈尔滨工业大学 航天学院,哈尔滨 150001)

摘 要:为解决 M5-edge 模拟器的理想化集总式取指令结构对基于 EDGE 体系结构设计空间探索的限制问题,对原模拟器的取指令前段进行分布式设计,包括总体的功能、具体的取指单元及单元间的互连网络设计,并在取指令块头的方式上设计了固定方式和循环方式两种方案.通过对实现后的结构进行在不同分布单元数量条件下的仿真分析,得到从理想集总式取指结构到实际分布式结构的性能下降关系和不同取指令块头方式的优劣.通过进一步分析,得出通信延迟和缓存缺失率对处理器性能的影响.

关键词:EDGE 体系结构;分布式取指;通信延迟;缓存缺失率

中图分类号:TP302.7 **文献标志码**:A **文章编号**:0367-6234(2017)05-0016-06

The design and analysis of distributed fetch based on M5-edge

ZHANG Chao, YU Mingyan

(Department of Astronautics, Harbin Institute of Technology, Harbin 150001, China)

Abstract: A distributed fetch structure of M5-edge is designed for the purpose of expanding the design space of EDGE architecture. The structure includes the overall function, distributed fetch unit and the interconnection network between the units. Two kinds of fetching block head are realized, including fixed fashion and round robin one. The analyses, which are made in different distributed fetch unit counts, provide the leave of reduction of distributed fetch comparing with the ideal lumped fetch model, as well as the difference between the two fashions of fetching block head. Furthermore, the effect of the processor performance by the communication latency and the cache miss rate are shown.

Keywords: EDGE; distributed fetch; communication latency; cache miss rate

为提高处理器单线程性能而进行的体系结构上的创新,将会带来明显的功耗增加[1],而通过多核技术带来的处理器性能的提高在不远的未来可能无法满足性能每两年翻一番的期望[2-3].新的指令集体系结构作为可能的解决方案在过去的十余年间进行了多种尝试[4-7],国内外多所知名院校及研究机构推出了具有各自特色的用于学术研究的未来处理器模型.作为其中较为成熟的代表,EDGE[8](explicit data graph execution)体系结构在设计理念、结构创新、工具链完善程度、流片验证,及体系结构的后续发展等方面都具有一定优势.

EDGE 体系结构采用显示通信指令集和块原子性的执行方式,通过块内指令的直接通信实现类数据流结构,充分开发指令级并行性,加快程序的执行速度,在满足低功耗、高可扩展性、多种应用的适应性等要求的前提下,实现了出色的性能功耗比.以 EDGE 指令集为基础已经设计实现了多种先进的处理器结构,主要包括 TRIPS[9]、TFlex[10]、T3[11]、E2[12]等.而针对该指令集体系结构,现有的通用处理器仿真平台不具有对其指令集的描述和微结构仿真的支持.M5-edge[13]作为目前唯一基于广泛使用的开源模拟器[14]进行开发的 EDGE 体系结构仿真平台,相比于奥斯丁 TRIPS 小组所提供的仿真工具链具有明显的优势.该模拟器完成了很好的仿真精度、速度和可扩展性的平衡,为基于显式通信和 EDGE 体系结构为基础的新型处理器提供研究支持.

M5-edge 通过分析 EDGE 体系结构特点和其微结构实现的共性,对处理器工作过程进行合理分割,提出四级时序模型,分别为取指令块、映射、执行、递交.该划分准确反映 EDGE 体系结构特点,使得研究人员可以方便快捷地进行处理器设计空间的探索.但是,M5-edge 同时存在着一些不足,其中较突出的问题在于缺乏一些具体的处理器分布式微结构模型的实现,包括分布式的取指令系统和分布式的数据存储系统等.原模拟器中以理想的集总式的结构代替,随着处理器设计理念的发展,这些集总单元限制了研究者的拓展空间,在进行针对这些部件的具体研究中,该模拟器无能为力.

收稿日期:2015-08-11

作者简介:张 超(1984—),男,博士研究生;
喻明艳(1962—),男,教授,博士生导师

通信作者:喻明艳,myyu@hit.edu.cn

针对该问题，本文对 M5-edge 模拟器的取指令块部分进行分布式研究，提出一种可以自由配置的分布式取指令单元结构，并对提出的方法和不同参数配置下的性能结果进行分析，为 EDGE 结构的分布式取指端实现提供一种有效的解决方案，该方法同样可以作为模拟器其他部件分布化的参考.

1　取指模型整体实现

对于 EDGE 体系结构，由于其本身的块原子性设计，使得取指令阶段区别于传统处理器的以指令为单位，而是以本身定义的指令块为单位完成取值工作. 该指令块由多个基本块构成，采用谓词化技术(predicate)消除基本块间的控制相关，以实现块内数据流执行方式，达到加速处理器执行能力的目的.

M5-edge 取指阶段大致由五部分组成，如图 1 所示. 其中指令缓冲的设计大部分继承原有 M5 模拟器的 O3(out of order)CPU 模型，增加了为构成指令块数据结构而必须的数据位和结构等. 块预测器通过当前完成的递交块地址结合历史信息对下一个块的地址信息进行预测. 虽然以块为单位的特点，一定程度上提高了指令缓冲和块预测器的命中率，但一旦出现投机的失败，所承担的恢复开销也更大，所以同传统的处理器一样，影响取值带宽的主要因素仍然集中在预测器的预测失败和一级指令缓存的未命中. 当发生指令缓存未命中的情况时，每个取指令单元应该独立处理属于自己部分的未命中事件. 取指单元负责处理取指令过程中的各部分状态变化和异常处理，当块中所有取指令过程完成时，还负责生成指令块数据结构. 至此，取指令部分工作完成，在译码过后，控制单元将完成指令到分布式的执行单元的映射工作.

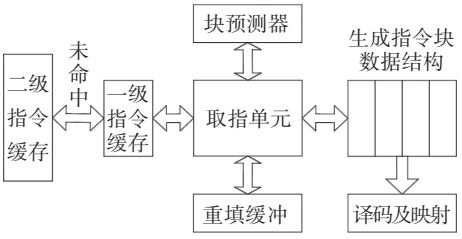

图 1　M5-edge 取指令块阶段过程示意

Fig.1　The schematic of fetch instruction stage

取指令过程的模拟器实现的各阶段状态转换如图 2 所示. 主要包括等待指令(i)、取指令块块头(f_h)、取指令块块体(f_b)、等待指令缓存(w_i) 和等待异常处理(w_f). 在 EDGE 指令集中将指令块分为块头和块体，分别存储不同类型的指令. 其中，块头中不仅有寄存器相关指令，还有其他一些和指令运行有关的重要信息，需要专门进行处理. 值得注意的是，异常处理需要等到指令块递交时才能进行，因此在递交之前出现异常需要等待，直到异常处理结束.

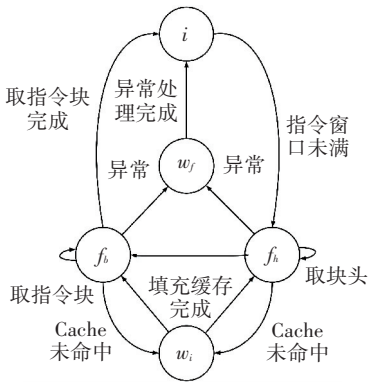

图 2　取指令过程有限状态机

Fig.2　The finite state machine of fetch

2　分布式取指单元实现

本文实现了一个可变数量的分布式取指令单元，可完成 2、4、8、16 及更多的分布式取指单元实现. 在具体设计过程中参考了 TFlex 取指令结构随着执行核重构而产生不同的取指令结构，完成 1 * 2、2 * 2、2 * 4 等取指令结构的设计. 之后开发的基于 EDGE 体系结构处理器，包括 E2、T3 等具有相同的取指设计方式. 由于实现原理相同，本文以八单元分布式取指前端为例(如图 3 所示)，为 8 取指单元 cpu 高层模型.

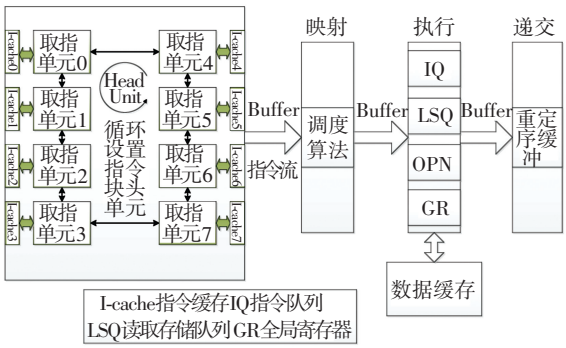

图 3　具有分布式取指令前端的模拟器时序模型

Fig.3　The four-stage model of M5-edge with distributed fetch units

考虑模拟器不同情况的仿真需求，分布式取指前端将以理想的无网络延时和具有实际消息传递网络两种模式实现. 图 3 中，小的分布式指令缓存与分布式取指单元紧密相连，完成甚块中各部分的取指任务. 两个取指单元之间的信息传递延时在理想情况下设为 0，这意味着产生于某个取指单元的信息可以在相同周期内被其他取指单元获得.

由于甚块的特殊构造，取指过程被分为两个阶

段. 首先,甚块头必须被优先取出以获得指令块的信息,比如指令范围和 chunk 数量. 在取指令块头结束后,改变各相关部分状态信息,并开始取甚块的指令 chunk 部分.

在设计取指令块头时,考虑两个方案:首先,为了实现简单选着不同的取指单元来进行头 chunk 和指令 chunk 的取指. 选取取指单元 0 进行取指令头操作,而其他取指单元则只用于取指令 chunk. 这样的好处是除了单元 0 以外的其他取指单元可以减少指令块头信息的解释器,从而简化硬件实现. 但是,由于指令 chunk 必须在所在甚块头 chunk 取完后才能取其中指令,因此,在取指令 chunk 时,原有的进行头 chunk 取指的单元将会被闲置. 第二种方案中考虑使每个取指令单元完全一致,可以完成所有取指要求. 这种方式同时具有更好的硬件可扩展性.

在模拟器中,取指单元的运行由其状态位决定. 状态位每时钟周期进行更新,包括以下状态:运行(running)、空闲(idle)、插入(squashing)、(icache wait response)、指令缓存重取等待(icache wait retry)、阻塞(block). 当处于 running 状态时,取指单元开始运行. 该有限状态机如图 4 所示.

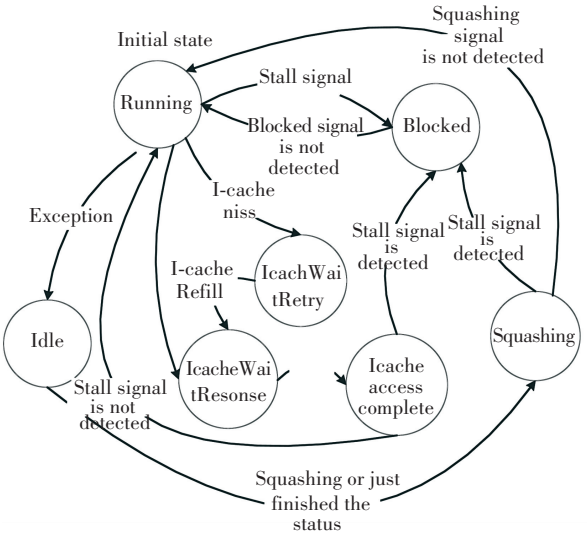

图 4 取指过程有限状态机

Fig.4 The finite state machine of fetch status

考虑实际分布式取指单元间通信时间问题,在取指单元间建立路由网络. 在取指过程中,取指单元间信息传递主要出现在两个阶段. 当取指令块头 chunk 完成时,该取指单元将会发送数据包给其他单元. 数据包中主要包括:取块头完成信号,目的取指单元,指令 chunk 地址,下一个指令块头取指单元等. 当各取指单元完成该指令块指令 chunk 取指任务后,将会向头取指单元发送反馈信息,并通过检查所有的返回信息判断该指令块取指是否完成.

在每个取指单元中增加路由形成网络,构成最终完整的分布式取指部件. 路由内部结构如图 5 所示,仲裁器通过对接收到的信息包进行译码来获知目地取指单元,并将信息包传进本地或 4 个方向的先进先出队列之一.

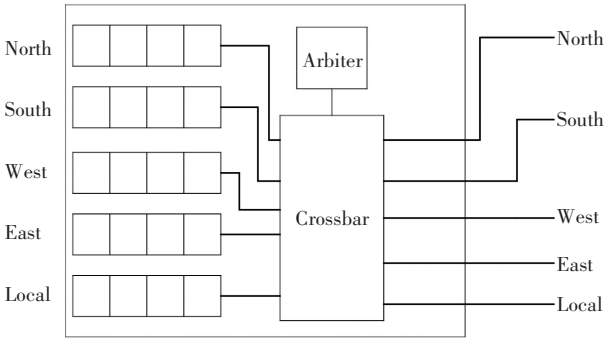

图 5 取指单元间路由结构

Fig.5 The structure of router

图 6 展示了完整的带路由网络的取指单元的运行模式,以伪代码方式呈现. 函数 promoteRouter( ) 从 fifo 中得到信息,并在路由中进行处理. 与之相反,promoteNetwork( )将路由器中处理好的信息传入输出端 fifo 中并发送数据包.

```
if ((UnitID == HeadUnitID)&&! headComplete )
fetch();
else if (headComplete)
fetch();
checkFault();
if (fault)
dealFault;
if (fetchComplete) {
if (fetchStatus == Head)
formHeadPacket(destID, UnitAddress, headComplete, blockID);
else if (fetchStatus == Body)
formBodyPacket(destID,bodyComplete,blockID);
}
promoteRouter();
promoteNetwork();
```

图 6 增加路由的取指单元伪代码

Fig.6 The pseudo-code of fetch unit with router

3 实验评估

3.1 仿真环境

本文以 M5-edge 作为基础,在其基础上进行了分布式的取指令系统和分布式一级缓存系统的设计与实现. 为了对所实现模型进行分析,基本参数配置如表 1 所示. 为了减少执行后端对性能的影响部分参数被设置成完美. 在这种情况下,每周期执行的指令数(IPC)将会很好地反映取指令能力的变化

对处理器性能的影响.

为了研究不同数量分布式取指令单元的性能影响,对 2、4、8 分布取指令单元进行了仿真. 一级缓存总大小保持不变,固定为 512 K,相应的 2、4、8 分布式的一级缓存大小为 256 K、128 K、64 K. 为了观察更明显趋势,部分仿真结果扩展到 16 分布取指单元. 为验证模拟器性能变化,仿真平台采用经过扩展分布式取指令前段的 M5-edge,分布式缓存系统仍为原有集总式实现方式. 测试程序为 SPEC CPU2000 中 TRIPS 工具链可编译的 18 个整形和浮点程序,具体见表 2.

表 1　M5-edge 模拟器仿真参数设置

Tab.1　The parameter configuration of M5-edge

参数	配置
取指	分布式可变数量取指单元,每单元 1 cacheline/cycle,每个 cache 行最多 16 条指令
映射	1 cycle 延时,依据静态调度算法映射
分派	1 cycle 延时
执行	16 个分布式执行单元,每个单元单发射,单旁路宽度,功能单元延时周期:整型(乘 3、除 20),浮点(乘 4、除 12、平方 24),其他 1
递交	1 cycle 延时
指令队列	分布式,每个执行单元中 128 个 entry
操作数网络	2D-mesh 1hop/cycle
分支预测器	完美
断言	完美
一级 cache	32KB,命中延时 1 周期
二级 cache	2MB,命中延时 12 周期
调度方法	SPS 方法

表 2　测试程序集列表

Tab.2　The test benchmark details

SPEC2000 测试集	配置
整型程序	164. gzip, 175. vpr, 176. gcc, 181. mcf, 186. crafty, 197. parser, 253. perlbmk, 255. vortex, 256.bzip2,300.twolf
浮点程序	168.wupwise,171.swim,172.mgrid,173.applu,177.mesa,179.art,183.equake,188.ammp

3.2　分布式取指性能评估

采用固定单元取指令块头方式,以某一分布取指令模块作为唯一拥有取指令块块头能力的模块而专用,其他模块用作指令块其余部分取指使用,这种方式带来的好处是节省了其他取指单元的硬件资源,简化了设计难度. 而不采用固定单元取块头的方式,将每个取指单元都增加相应硬件,以具备取指令块块头的功能,并在取指令块过程中循环设置,则可能会获得性能的提高. 下面通过扩展后的 M5-edge 平台完成对两种方式的性能评估.

首先,在不考虑分布单元间通信的情况下,考虑两种实现方式的性能比较及相较于理想的集中式方式的变化. 如图 7 所示,在相同的较为理想的执行后端条件下,固定取指令块块头的方式在性能上较集中式方式在低数量分布单元情况下,性能相差较大,而当分布单元数量超过 8 时,性能基本持平. 这是由于一个单元被专用于取指令块块头,而使得在取指单元较少情况下,进行取指令端口变少(所有分布式情况均为 8 端口)而引起,后面会通过同时在线指令数及 cache miss 率进行具体分析.

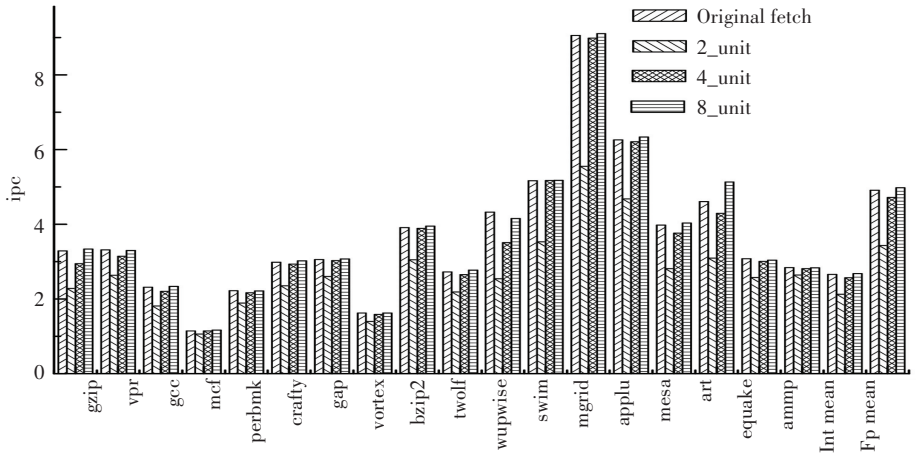

图 7　固定取指令块头方式与理想集总式取指单元性能比较

Fig.7　Performance comparison between fixed fashion and original fetch

图 8 给出了两种取指令块头方式的性能比较. 在所有情况下,循环设置方式具有更高达的性能,但性能差距随着分布单元数量的增加而减小,当数量增加到 8 时性能差距对于整型和浮点测试程序分别为 0.66% 和 1.28%,循环设置方式所带来的好处已经并不明显.

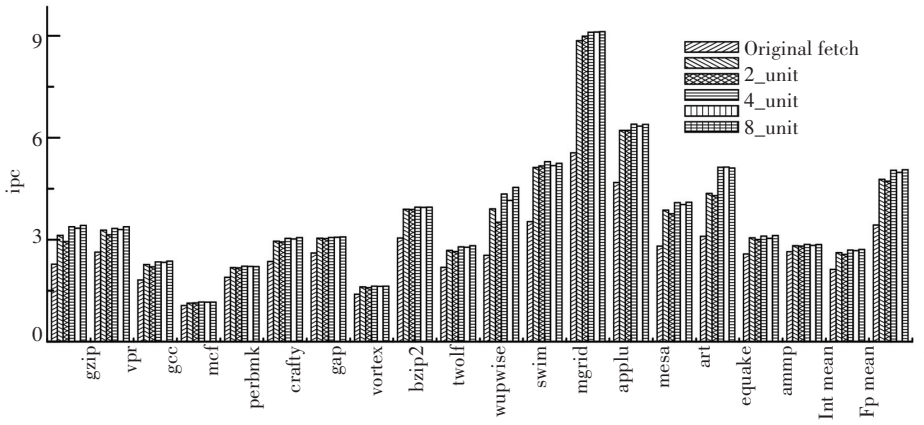

图 8　固定取指令块头方式与循环设置方式性能比较

Fig.8　Performance comparison between fixed and round robin fashion

考虑 cache miss 的影响对于处理器分布式取指单元的设计来说是重要的,合适的 cache 大小将对处理器性能产生决定性的影响. 图 9 和图 10 分别给出了本文环境中处理器 cache miss 率随着分布式的二级 cache 大小的减少而产生的变化,其中 512 K 代表的是集总式的取指令结构,256 K 则代表着两个分布取指单元,每个 cache 大小为 256 K,以此类推,32 K 代表着 16 个分布的取指令单元.

整型程序 cache miss 率受 cache 尺寸变小影响更大. 由图中纵坐标可以看出,两条曲线拐点出现的位置也并不相同,整型曲线出现在 256 K 和 128 K 间,也就是分布式单元超过 4,cache miss 率急剧上升,而浮点曲线则显示超过 8 时才会急剧上升. 对于 8 分布的取指单元对比原有集总式结构,cache miss 率在整型程序和浮点程序上分别上升了 10.8 倍和 46.3 倍,指数级增长的缺失率暗示了 cache 单元的尺寸不能过小,否则 cache miss 带来的性能损失将无法承受. 在本文实验环境中,128 K 的分布式二级 cache 大小可能更适合于分布的 cache 粒度.

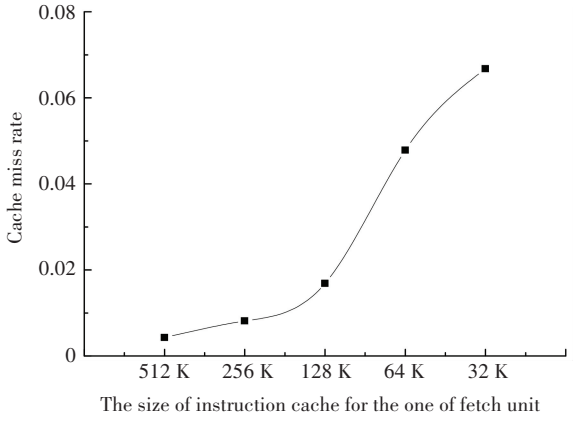

图 9　分布式 cache 大小与 miss 率关系(整型)

Fig. 9　The relationship between cache size and miss rate (integer)

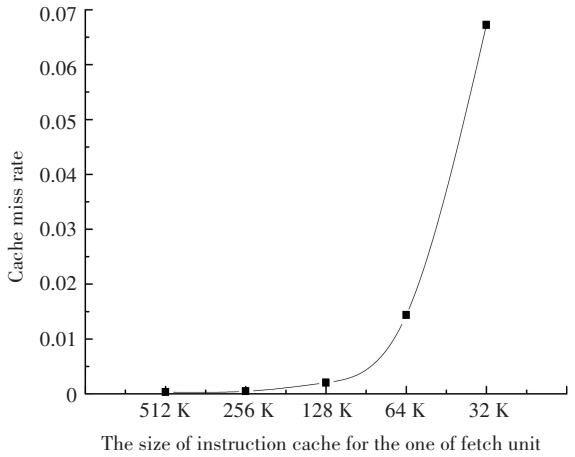

图 10　分布式 cache 大小与 miss 率关系(浮点)

Fig.10　The relationship between cache size and miss rate (float point)

考虑实际通信情况,增加路由网络,带来通信延迟,造成处理器性能下降,具体如图 11 所示. 随着取指单元数目增加,性能损失增加. 对于整型 SPEC2000 测试程序来说,性能在 2、4、8 情况下分别下降 11%、18%和 38%,而对于浮点程序来说,性能分别下降了 15%、17%和 38%. 相比于理想的集总式的取指令结构,增加通信路由后带来的性能下降是符合预期的. 但分布式的取指令单元除了解决过大的集总结构无法实现的问题外,还具有潜在的性能提高的方式. 将分布式的取指令单元与执行单元集成到一起,虽然取指时间相较集总式结构增加了,但是节省了指令分派时间,从而获得性能上的改善. 但是,这种方式在以指令为粒度进行取指的处理器中很难实现,这是因为为了追求最优的性能,指令会被编译器安排到合理的处理单元,但是,每次取指令不是逐条进行,而是每次取出一个 cache 行,二次调度会浪费更多的资源和时间. 目前,分布式取指单元与处理单元紧耦合结构基本上以线程或指令块为基本取指单位. 而以指令为目标的分布式取指单元和执行单元则是分离的结构,比如 TRIPS 处理器.

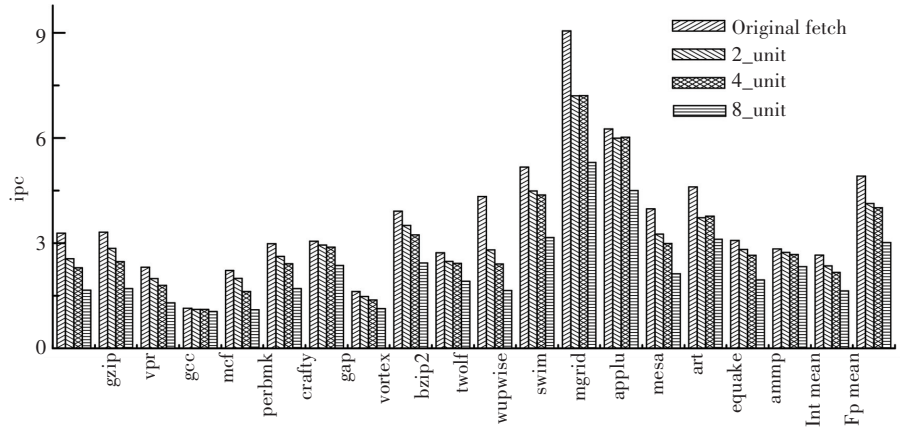

图 11　循环取指增加路由网络后性能比较

Fig.11　Performance comparison between original fetch and round robin fashion (router)

4　结　论

对基于 EDGE 体系结构研究而设计的模拟平台 M5-edge，进行了分布式取指令前段的设计，并对实现方式和重要参数进行分析. M5-edge 可以实现理想的集总式取指令前段，也可以实现多种方式的分布式取指端设计. 该模型既可以为基于 EDGE 的处理器后端微结构设计提供更精确和更灵活的取指令前段，也可以为基于前端的其他结构设计提供了基础.

通过对取指令端的整体功能、取指令单元和单元间互连网络的实现，完整介绍了分布式取指令前段的设计工作. 通过仿真结果分析得到取指令块头的不同设计方法的适应性，并确认了在从理想集总式结构进行分布式实现所带来的网络延时和 cache 缺失所带来的损失.

本文聚焦于分布式取指前端的实现和由集总式进行分布化所带来的性能损失，缺少对分布式取指单元与指令执行单元的紧耦合设计，该方式可以部分弥补由于分布化而带来的性能损失.

参考文献

[1] CZECHOWSKI K, LEET V, GROCHIWHKI E, et al. Improving the energy efficiency of big cores [C]// ACM/IEEE 41st International Symposium on Computer Architecture (ISCA). Minneapolis: IEEE, 2014, 493-504.

[2] BUGER D, KECKLER K, MCKMLEY K S, et al. Scaling to the end of silicon with edge architectures[J] IEEE Computer, 2004, 37(7): 44-55.

[3] ESMAEILZADEH H, BLEM, AMANT E. Dark silicon and the end of multicore scaling[C]//Proceeding of the 38th annual international symposium on Computer architecture. San Jose: IEEE, 2011: 122-134.

[4] TORRELLAS J, CEZE L, TUCK J, et al. The bulk multicore architecture for improved programmability[J]. Communications of the ACM, 2009, 52(12):58-65.

[5] SWANSON S, MICHESON K, SCHWERIN A, et al. Wavescalar [C]//Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture. San Jose: IEEE, 2003: 291-302.

[6] GEBHART M, MAHER B A, COONS K E, et al. An evaluation of the TRIPS computer system[J]. Acm Sigplan Notices, 2009, 44(3):1-12.

[7] WAINGOLD E, TAYLOR M, SARKAR V, et al. Baring it all to software: the raw machine[J]. IEEE Computer, 1990,30(9):86-93.

[8] SMITH J, GIBSON B, MACHER N, et al. Compiling for edge architectures[C]//The 4th International Symposium on Code Generation and Optimization. New York: IEEE, 2006:185-195.

[9] SANKARALINGARM K, NAGARAJAN R, MCDONAL R, et al. Distributed micro architectural protocols in the TRIPS prototype processor[C]//39th Annual IEEE/ACM International Symposium on Micro architecture. Orlando: IEEE, 2006:480-491.

[10] KIM C, SETHUMADHAVAN S, GOVINDAN M, et al. Composable light weight processors[C]//40th Annual IEEE/ACM International Symposium on Microarchitecture. Chicago: IEEE, 2007: 381-394.

[11] ROBATMILI B, LI D, ESMAEILZADEH H, et al. How to implement effective prediction and forwarding for fusable dynamic multicore architectures[C]//19th IEEE International Symposium on High Performance Computer Architecture (HPCA). Shenzhen: IEEE, 2013:23-27.

[12] DURIC M, PALOMAR O, STANIC A, et al. Dynamic-vector execution on a general purpose EDGE chip multiprocessor[C]//IEEE International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. Samos Island: IEEE, 2014:18-25.

[13] GOU P, LI Q, JIN Y, et al. M5 based edge architecture modeling [C]//IEEE International Conference on Computer Design (ICCD). Amsterdam: IEEE, 2010: 289-296.

[14] BINKERT N L, DRESLINSKI R G, HSU L R, et al. The M5 simulator: modeling networked systems[J]. IEEE Micro, 2006,26(4): 52-60.

（编辑　王小唯，苗秀芝）