计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.191000148

负载均衡的处理器运算资源分配方法

王国澎　杨剑新　尹　飞　蒋生健
上海高性能集成电路设计中心　上海 201204

摘　要　现代超标量处理器通常设置有多套计算部件支持指令并行执行,以提高程序的运行效率。运算资源分配策略在很大程度上决定了处理器能否充分利用计算部件并行加速计算,具有重要作用。就指令调度以及运算资源分配问题而言,当前的研究以软件编译优化方法居多。但编译优化是一种静态方法,优化排布的指令无法再次改变次序,且缺乏运行时信息,灵活性不够。为了降低因运算资源分配不当导致的指令阻塞,充分释放其运算能力,从硬件自动调度和实现的角度分析了这个问题,研究了对称情形和非对称情形下运算资源的细粒度自动分配方法,进而提出了负载均衡的贪心分配策略。实验结果表明,提出的运算资源分配方法可以有效减弱现代超标量处理器中指令发射不公平所带来的负面影响,能更好地利用片上缓冲资源和计算资源,且处理器中运算资源越多,发射缓冲深度越大,负载均衡方法获得的性能提升就越明显。

关键词:负载均衡;资源分配;指令调度;指令发射;流水线
中图法分类号　TP302

Computing Resources Allocation with Load Balance in Modern Processor

WANG Guo-peng, YANG Jian-xin, YIN Fei and JIANG Sheng-jian
Shanghai High Performance IC Design Center, Shanghai 201204, China

Abstract　To improve the efficiency of program, it is used to arrange multiple function units in modern superscalar processor, supporting to execute instructions in parallel. The allocation policy of computing resources plays an important role in taking full advantage of multiple function units. Although the policy of how to allocate computing resources and schedule instruction has been well studied in literature, the proposed solutions almost concentrate on optimization methods at compile time, which is mostly static, inflexible and inefficient because of lack of computing pipeline information at run time. To mitigate the negative impacts of improper computing resource allocation and maximize the power of multiple function units, this paper abstracts the mathematical model of resource allocation problem at run time and makes a study of hardware fine-grained automatic method based on symmetric and asymmetric configuration of function units, in order to make dynamic and wise computing resource allocating decision when instructions are issued in general situation. As a result, a load-balanced greedy resource allocation strategy is proposed and evaluated. The experimental results show that our policy is efficient to minimize blocking time caused by unfair allocation of computing resources. Furthermore, the more computing resources are provided, the better performance our policy can yield.

Keywords　Load balance, Resource allocation, Instruction scheduling, Instruction issue, Pipeline

现代超标量处理器采用深流水多发射机制挖掘程序指令级的并行性,相应地设置多个运算簇来处理发射的可以并行的指令。每个运算簇由一个或者多个计算部件构成,每个计算部件能够处理一类指令,多个运算簇聚合形成处理器的整体运算能力。处理器中运算簇的设置并不是简单地以量取胜,需要综合考虑性能需求、指令特点和硬件开销等因素,做出折中。

现代超标量处理器一般通过乱序发射来提升指令的并行性。乱序发射涉及两个重要任务:1)挖掘指令窗口中可以并行的指令;2)将这些指令均匀地分配到运算簇中进行并行计算。发射引擎可见的指令并行性与程序固有属性有关,也与指令窗口的大小相关,增大指令窗口有助于挖掘指令并行性,但也意味着需要设置更大的指令缓冲,会耗费更长的查找时间。另一方面,在给定指令窗口大小的情况下,指令的分配调度策略能决定了能否充分利用运算部件并行加速计算,好的发射策略能够降低因运算簇分配不当导致的指令阻塞,充分释放运算簇的计算能力。

就计算部件而言,分配的指令形成了其运算负载,要想提高处理器的吞吐率,就要保证指令尽可能快地完成处理。指令从发射到完成之间的时间 T 大体上可以用如下公式表示:

$$T=T_{\text{wait}}+T_{\text{exe}}$$

其中,T_{wait} 是从指令发射后到执行之前的等待时间,T_{exe} 是执行时间。一般而言,指令的执行时间是固定的,且在不同的计算部件实例中都是相同的;而等待时间则由指令间相关性、计算部件特点和分配策略决定,现代处理器中大多数计算部件

到稿日期:2019-10-23　返修日期:2020-05-17　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:"核高基"重大专项课题(2018ZX01029101)
This work was supported by the Natonal Science and Technology Major Project(2018ZX01029101).
通信作者:王国澎(wgp.sun@163.com)

都支持流水执行，指令间的相关性相对固定，分配策略需要保证指令不会因计算部件分配不当而延长等待时间。另外，各计算部件并行的指令序列中总有一条是关键路径，处理器的效率受限于运行时负载最多的那条运算流水线，因此负载均衡是运算流水线分配策略的主要目的，对提升处理器的执行效率有重要意义。

本文着眼于指令发射部分，主要研究运算流水线的细粒度自动分配方法。首先，将运算流水线分配问题抽象出来，建立数学模型并进行推导分析，得到对称情形下(即指令可以无差别分配到任意流水线上)一种流水线分配的贪心解法，进而推广到非对称情形(指令的流水线选择范围受限)进行求解。然后，根据具体的运算流水线模型设计了相应的流水线分配策略，进行模拟分析，并与传统的分配算法进行对比。

1 相关工作

计算部件属于处理器中的一种资源，为追求资源利用的最大化，设计过程中不可避免地面临着分配调度的问题。解决方法不外乎软件和硬件层面，实现方式又可分为静态分配和动态分配两种。

软件主要通过编译时优化指令排布来指导分配。文献[1]在代码生成过程中引入了针对表达式树的局部指令调度和局部寄存器分配的集成算法，减少了由于资源共享而引起的互锁和冲突问题。文献[2]同时进行指令调度与寄存器分配，编译器优化时能同时得到这两者信息，从而提升优化的效果。针对 VLIW 架构处理器，文献[3]通过编译器在程序中添加显示的指示并行的指令，硬件根据这些标识来发射指令，但是只能基于基本块内部的调度。软件流水[4]能够对循环结构实现细粒度的指令调度，模调度[5-6]就是一类常用的软件流水调度方法。文献[7]通过使用循环相关反依赖来消除模调度算法产生的变量活跃域重叠，并使用依赖约束和资源约束回溯模型消除节点冲突，提高模调度算法的有效性。编译优化是一种静态方法，排好的指令无法再次改变次序，且缺乏运行时信息，效果有限。

硬件方法基于指令窗口进行调度。如果发射指令数目远大于提交指令数目，则发射队列中会存在许多等待的指令，造成浪费；如果发射指令数目少于提交指令数目，则会造成处理器后端“饥饿”问题，无法充分利用资源。文献[8]研究了早期超标量处理器资源设置方法以及发射策略，对当今处理器的设计仍具有指导意义。文献[9]通过观察发射队和重排序缓冲中的指令数量来决定是否关闭取指部件或者调整发射缓冲大小，提出了两级 shelving 的指令调度设计，把发射队列划分为两级，提高了指令发射队列的利用率。此方法虽然能降低能耗，但也降低了性能。应用程序在执行过程中常常会出现重复执行同一段代码段的情况，并且乱序执行这部分代码时所产生的执行调度信息也非常类似。文献[10]通过保存和重用这部分调度信息，把乱序执行和 VLIW 的两种优势结合起来，提高了性能功耗比。

相比编译器，处理器发射部件看到的只是运行时的代码片段，但对整个运算流水线的具体负载以及执行情况较为了解，因此可以在指令筛选和组合时进行细粒度的动态调整。但分配算法本身的开销不容忽视，复杂的分配算法效果较好但不易于实现，有时为了简化设计，发射部件也可以实现静态分配算法，例如简单轮转等。

2 流水线模型

研究过程中，我们模拟的处理器核心所包含的计算资源如表 1 所列。计算部件再互相组合成为运算簇，运算簇设置的原则如下：

(1)整数指令、浮点指令以及访存指令分开配置运算簇。

(2)按照指令运算特点将指令分类，每一类指令对应一种计算部件，每一类指令只能发送到对应的计算部件上执行。

(3)对于出现频率较高的指令，设置多个计算部件，较为冷僻的指令类所对应的计算部件相应减少。

(4)不同的计算部件按照执行时间长短互补的方式聚合成运算簇，运算簇所能接受的指令为所包含计算部件指令的并集。出现频率高的指令类所对应的计算部件数目也相对较多，并且分散在不同的运算簇中。

表 1 目标处理器核心所设置的计算资源

Table 1 Computing resource list of target processor

类型	计算部件	功能	发射延时	执行延时
整数	IntALU	基本的整数加减、比较、逻辑运算	1	1
	IntSHTBOP	整数移位、字节运算	1	1
	IntSELCNT	整数选择传送、位计数运算	1	1
	IntBR	执行整数分支指令	1	1
	IntMULT	整数乘法运算	1	4
访存	RdPort	发送访存读	1	1
	WrPort	发送访存写	1	1
浮点	FltMA	浮点加减、比较、乘法、符号拷贝以及乘加运算	1	6
		浮点条件选择运算	1	2
	FltCVT	浮点转换	1	4
	FltDS	浮点单精度除法运算	17	17
		浮点双精度除法运算	32	32
		浮点单精度平方根运算	17	17
		浮点双精度平方根运算	31	31
	FltBR	执行浮点分支指令	1	1

综上所述，本文运算簇模型的具体配置如表 2 所列。从结构上讲，发射延时分为运算簇的发射延迟和运算部件的发射延迟。运算簇虽然包含多个计算部件，但发射通道只有一条，每个时钟周期只能接受 1 条新指令，因此发射延时为 1 个时钟周期。就运算部件而言，除 FltDS 之外的运算部件都可以流水地执行指令，发射延时也为 1 个时钟周期。浮点除法和平方根运算无法流水处理，如果发射了一条浮点除法或平方根指令，则 FltDS 执行完已发射的指令后才能接受下一条指令，发射延时即指令执行延时。因此，指令发射到对应的运算部件上执行时，需要运算簇和运算部件同时可接受。

指令完成译码重命名后，先放置在等待缓冲中，每个运算簇对应一个发射缓冲；指令完成运算流水线分配后，放置在发射缓冲中。图 1 给出了运算流水线模型，每周期最多可同时发射 3 条整数指令、2 条浮点指令和 2 条访存指令。发射缓冲中的指令被发射执行的条件为该指令源操作数已经准备好，且对应的运算簇和计算部件均空闲。指令发射后，将对应的运算簇置为忙状态，下一拍恢复空闲状态；同时，封锁对应的计算部件，可以流水处理的计算部件一拍后解除封锁，FltDS 部件的封锁状态则要等正在计算的指令完成后才能解除。

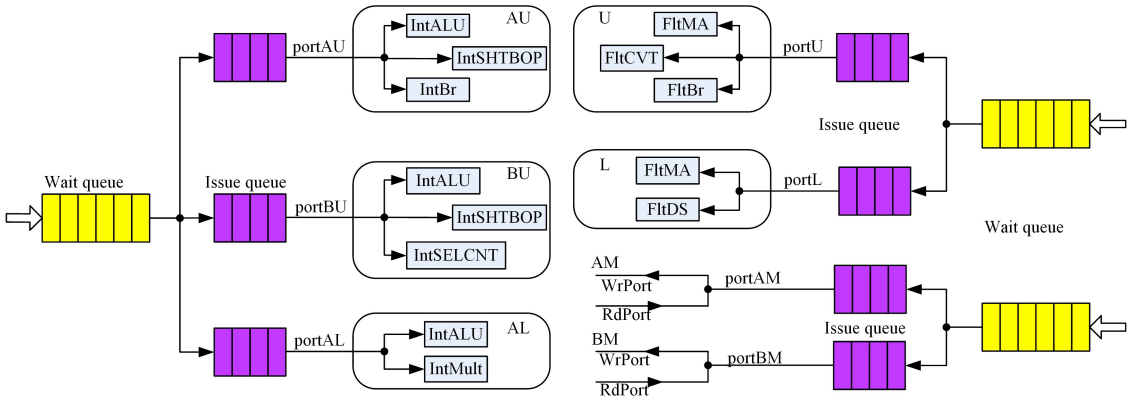

图 1 运算簇模型结构

Fig. 1 Computing resource model

运算簇中计算部件的组成决定了指令发射后可能的流向，如果某类指令对应的计算部件在多个运算簇中都有例化，那么这类指令可分配的运算簇就有多种选择。指令对应运算簇的发射延迟和选择范围是影响分配的首要因素，也是影响执行效率的重要因素，本文将重点对其进行研究。

表 2 运算簇的配置

Table 2 Configuration of computing pipeline

运算簇	发射端口	包含的计算部件	发射延时	类型
AU	PortAU	IntALU，IntSHTBOP，IntBR	1	整数
BU	PortBU	IntALU，IntSHTBOP，IntSELCNT	1	
AL	PortAL	IntALU，IntMULT	1	
U	PortU	FltMA，FltCVT，FltBR	1	浮点
L	PortL	FltMA，FltDS	1	
AM	PortAM	RdPort，WrPort	1	访存
BM	PortBM	RdPort，WrPort	1	

3 负载均衡的运算流水线分配方法

3.1 问题抽象

首先需要定义运算流水线负载。由于大多数计算部件都支持流水，且指令发射到计算部件后开始执行，因此一个运算簇中可能有多条指令并行计算，指令执行时间并不是分配时所参考的运算流水线负载。如前所述，一个运算簇一般只有一个发射端口，运算簇接到一条发射的指令后需要一定的时间（指令发射延时）做进一步的处理，在这段时间内，运算簇无法接受新的指令。指令发射延时由指令本身和运算簇特点共同决定，已完成分配但尚未执行的指令的发射延时之和形成了运算流水线的负载。

假设有 m 条流水线，每一轮需要将 n 条指令分配到这 m 条流水线上，每条指令的执行时间已知，指令 i 分配到流水线 j 上的发射等待延时为 $w_{i,j}$，如果指令 i 无法分配到流水线 j 上，则 $w_{i,j}=Max\_Lat$（最大等待延时），并且假设流水线 j 上已有指令还需要 Lat_j 的时间才能全部完成发射。分配时先不考虑指令间的相关性所带来的延时，指令相关所带来的发射延时由编译器和发射策略利用，不在本文的考虑范围之内。

用以下函数表示指令 i 是否被分配到流水线 j 上：

$$I(i,j)=\begin{cases}1, & \text{指令 } i \text{ 分配到流水线 } j \text{ 上}\\ 0, & \text{否则}\end{cases}$$

可知 $\sum_{j=0}^{m-1} I(i,j)=1$，即指令 i 不可能同时分配到多条流水线上。经过这一轮分配后，流水线 j 上的指令需要 $L_j=Lat_j+\sum_{i=1}^{n} I(i,j)w_{i,j}$ 的时间才能发射完毕。

为了衡量 m 条流水线的负载均衡效果，采用方差作为评价函数。m 条流水线将指令执行完的平均时间为：

$$\overline{L}=\frac{1}{m}\sum_{j=0}^{m-1} L_j$$

其方差为：

$$Var=\frac{1}{m}\left[\sum_{j=0}^{m-1}(L_j-\overline{L})^2\right]$$

运算流水线负载均衡的目标是使分配完 n 条指令后的 Var 最小，并求解 $I(i,j)$。当方差最小时，各条流水线上的负载较为均衡，消除了指令并行执行的短板效应。

由上述分析可以看到，该分配问题并非常规线性规划问题，无法直接求解。解决该问题最简单的方法就是遍历每种可能的分配组合，使得各流水线负载方差 Var 最小，但这种方法的开销也最大，无法通过硬件来实现。

通过流水线分配问题的抽象模型来看，由于处理器看到的指令有限，且每一轮只有完成预处理的 n 条指令需要进行分配，直观的想法为每次分配时仅立足于当前的状态，每一轮都做出最优的选择，那么经过一系列的决策后可以保证应用程序更快地被执行。这一解决方法符合贪心策略的思路，下面研究如何用贪心法解决运算流水线分配的问题。

3.2 对称情形

贪心解法通过每一步寻找当前的最佳选择，期望通过所得到的局部最优解构造出一个全局最优解。但贪心法并不总能产生最优解，下面寻找使用贪心法可以获得指令对称分配情况下流水线分配问题最优解的条件和方法。对称情况下，每个运算簇的配置都是一样的，指令可被无差别地分配到任意运算流水线。

运算流水线的负载用多元组 $L_i=\{L_{i,0},L_{i,1},\cdots,L_{i,j},\cdots,L_{i,m-1}\}(0\leqslant i\leqslant n)$ 表示，其中 $L_{i,j}$ 表示分配完第 i 条指令后第 j 条运算流水线的负载，初始时 $L_0=\{Lat_0,Lat_1,\cdots,Lat_{m-1}\}$。在对称贪心解法中，每一步为一条指令分配运算流水线，为第 i 条指令分配流水线时，选择 $k_i=\arg(\min_{j=0}^{m-1} Var_{i,j})$（$Var_{i,j}$ 表示把指令 i 分配到流水线 j 上时各流水线的方差）对应的流水线作为分配对象，如果满足条件的 k_i 有多个，则按照一定次

序进行轮转分配。指令 i 分配结束后，得到当前所有流水线的负载：

$$L_i=\{L_{i-1,0},L_{i-1,1},\cdots,L_{i-1,k_i}+w_{i,k_i},\cdots,L_{i-1,m-1}\}$$

令 Var_i 表示分配指令 i 后各流水线负载的方差，$\overline{L_i}$ 表示分配指令 i 后各流水线负载的均值，假设对于指令序列{1,2,…,i,…,n}所分配的流水线编号序列为{$k_1,k_2,\cdots,k_i,\cdots,k_n$}，则：

$$\overline{L_i}=\frac{w_{i,k_i}}{m}+\frac{1}{m}\sum_{j=0}^{m-1}L_{i-1,j}=\frac{w_{i,k_i}}{m}+\overline{L_{i-1}}$$

进而可以计算方差 Var_i：

$$\begin{aligned}Var_i&=\frac{1}{m}[\sum_{j=0}^{m-1}(L_{i,j}-\overline{L_i})^2]\\&=\frac{1}{m}[(L_{i-1,0}-\overline{L_{i-1}}-\frac{w_{i,k_i}}{m})^2+\cdots+(L_{i-1,k_i}-\overline{L_{i-1}}+w_{i,k_i}-\frac{w_{i,k_i}}{m})^2+\cdots+(L_{i-1,m-1}-\overline{L_{i-1}}-\frac{w_{i,k_i}}{m})^2]\\&=\frac{1}{m}[mVar_{i-1}+\frac{2w_{i,k_i}}{m}\sum_{j=0}^{m-1}(L_{i-1,j}-\overline{L_{i-1}})+2w_{i,k_i}(L_{i-1,k_i}-\overline{L_{i-1}})+m(\frac{w_{i,k_i}}{m})^2+w_{i,k_i}^2-\frac{2w_{i,k_i}^2}{m}]\\&=Var_{i-1}+\frac{1}{m}[2w_{i,k_i}(L_{i-1,k_i}-\overline{L_{i-1}})+w_{i,k_i}^2-\frac{w_{i,k_i}^2}{m}]\end{aligned}$$

由此得到 Var_n 的一个通项公式：

$$Var_n=Var_0+\frac{1}{m}\sum_{i=1}^{n}w_{i,k_i}[2(L_{i-1,k_i}-\overline{L_{i-1}})+\frac{m-1}{m}w_{i,k_i}] \tag{1}$$

因为对称情形下同一条指令在每条流水线上发射的等待时间都是一样的，即：

$$w_{i,0}=w_{i,1}=\cdots=w_{i,m-1}=w_i$$

所以式(1)可以表示为：

$$\begin{aligned}Var_n&=Var_0+\frac{1}{m}\sum_{i=1}^{n}w_i[2(L_{i-1,k_i}-\overline{L_{i-1}})+\frac{m-1}{m}w_i]\\&=Var_0+\frac{2}{m}\sum_{i=1}^{n}w_iL_{i-1,k_i}+\frac{m-1}{m^2}\sum_{i=1}^{n}w_i^2-\frac{2}{m^2}\sum_{i=1}^{n}\sum_{j=0}^{m-1}w_iLat_j-\frac{2}{m^2}\sum_{i=1}^{n}\sum_{j=1}^{i-1}w_iw_j\end{aligned}$$

其中，$\frac{m-1}{m^2}\sum_{i=1}^{n}w_i^2$，$\frac{2}{m^2}\sum_{i=1}^{n}\sum_{j=0}^{m-1}w_iLat_j$ 和 $\frac{2}{m^2}\sum_{i=1}^{n}\sum_{j=1}^{i-1}w_iw_j$ 均为常数，因此：

$$Var_n=\frac{2}{m}\sum_{i=1}^{n}w_iL_{i-1,k_i}+C \tag{2}$$

引理 1(选择引理)　分配指令 i 时，选择 $k=\arg(\min_{j=0}^{m-1}L_{i-1,j})$($1\leqslant i\leqslant n$)的流水线对应的方差最小。

证明过程详见首页 OSID 码。

定理 1(分配定理)　将 n 条指令分配到 m 条对称的运算流水线上时，将 n 条指令按照发射等待时间 w_i 从大到小排序，然后依次采用选择引理为指令分配流水线，每次分配一条指令，则可以得到本轮分配的一个全局最优解。

证明：对于给定的流水线 j，每次分配一条指令后，其负载序列{$L_{0,j},L_{1,j},\cdots,L_{n,j}$}是单调递增的。根据选择引理，每次选择当前负载最小的流水线作为分配目标，则执行整个选择过程获得的负载序列{$L_{1,k_1},L_{2,k_2},\cdots L_{i,k_i},\cdots,L_{n,k_n}$}也是单调递增的。

由于 Var_n 是一个非 0 值，要使 Var_n 最小，就要使其中的因子 $\sum_{i=1}^{n}w_iL_{i-1,k_i}$ 最小。为了达到这个目标，将指令按照执行时间 w_i 从大到小排序后逐步采用引理 1 进行分配即可实现，证明方法与选择引理的证明过程类似。

以下举例说明：假设有 3 条指令运算流水线 AU，BU 和 AL，每条流水线的初始负载分别为 3，2 和 1，当前有 4 条指令(I0，I1，I2 和 I3)需要分配，每条指令都可无差别地发射到 AU，BU 和 AL 上执行，指令发射等待延时分别为 6，4，3 和 2。表 3 列出了分配这 4 条指令的 3 种不同方案，方案 1 为按照分配定理处理后的结果，即先将待分配指令按照发射延时从大到小排序后，每次再使用选择引理为指令分配流水线。从表 3 中可以看到，使用分配定理处理后的各流水线负载是最均衡的。方案 3 和方案 1 的区别在于将指令按照发射延时升序排序后再分配，此时各流水线负载方差最大，最快的流水线在 5 个时钟周期即可完成指令发射，而最慢的流水线则需要 9 个时钟周期才能发射完毕。

通过以上证明和分析过程，可以得到对称情形下的流水线分配算法，即将 n 条指令流水线的分配过程划分为一系列的子问题，每次为 1 条指令完成流水线分配后都能保证流水线负载最为均衡，进而得到全局最优解。在实现中，由于多数指令的执行(除平方根和除法指令外)都是可以流水的，相当于每条指令的发射等待时间都为 1 个周期(即指令的 IssueWaitTime 为 1)，因此运算流水线的负载退化为分配到该流水线上的指令的数量。

表 3　对称情形下不同流水线的分配方案

Table 3　Examples under symmetric situation

Pipe	Plan1					Plan2					Plan3				
	分配 I0	分配 I1	分配 I2	分配 I3	Var	分配 I2	分配 I1	分配 I0	分配 I3	Var	分配 I3	分配 I2	分配 I1	分配 I0	Var
AU	3	3	6	6		3	3	9	9		3	3	7	7	
BU	2	6	6	8	2/3	2	6	6	6	2	2	5	5	5	8/3
AL	7	7	7	7		4	4	4	6		3	3	3	9	

3.3　非对称情形

由于计算部件的功能不同且资源有限，对于程序中经常出现的指令，往往设置多个运算部件以加速计算，而对于部分使用不是很广泛或者执行实现开销较大的指令，其运算部件通常设置较少，因此不同指令可分配执行的流水线可能不同，部分指令可能只能分配到其中一条或几条流水线上。如何找到满足负载均衡要求的运算流水线分配方案，就是非对称情形下所要研究的问题。

非对称情形下仍可使用贪心解法求解，每条指令只在其可以选择的范围中寻找负载最小的流水线作为发射的目标。

但在这种情况下,由于流水线分配候选对象受限,按照上述方法分配后各条流水线的负载可能不是最优的,下文将举例进行说明。

假设 3 条指令运算流水线 AU,BU 和 AL 的初始负载分别为 3,2 和 1,当前有 4 条指令(I0,I1,I2 和 I3)需要分配到这 3 条流水线上。除 I0 和 I2 选择不受限外,I1 只能分配到 AL 上,I3 只能分配到 BU 和 AL 上。表 4 列出了这 4 条指令按照对称情形下的贪心方法分配后各条流水线的负载和均衡程度。

表 4 非对称情形下原贪心解法的分配方案

Table 4 Result of symmetric greedy allocation under asymmetric situation

Pipe	Init.	分配 I0	分配 I1	分配 I2	分配 I3
AU	3	3	3	3	3
BU	2	2	2	5	7
AL	1	7	11	11	11
Var	2/3	14/3	146/9	341/27	32/3

但以上解法并不是最优的,存在一种方差比 32/3 更小的分配方法,如表 5 所列。

表 5 非对称情形下一种更优的分配方案

Table 5 Example of better allocation under asymmetric situation

Pipe	Init.	分配 I1	分配 I0	分配 I3	分配 I2
AU	3	3	3	3	6
BU	2	2	8	8	8
AL	1	5	5	7	7
Var	2/3	14/9	38/9	14/3	2/3

当分配一条选择受限的指令时,如果当前负载最小的流水线恰好在其选择范围内,则仍能保证分配后的方差最小;否则就可能加大流水线负载不均衡的程度。一种可能的解决方法就是优先确定选择受限指令的分配,在此基础上使用对称指令来降低负载的不均衡性。这种优先确定的方式就是根据已经按照执行时间排好序的指令的流水线的选择范围对其进行调整,交换指令次序,从而达到流水线负载更均匀的目的。下文对指令位置交换条件进行讨论。

按照发射等待时间排序后的指令序列为$\{1,\cdots,i,\cdots,j,\cdots,n\}$,对于任意 $i<j$ 满足 $w_i\geqslant w_j$,由式(2)得到分配完 n 条指令后 m 条运算流水线的负载方差:

$$Var_n=\frac{2}{m}\sum_{i=1}^{n}w_iL_{i-1,k_i}+C$$

不妨假设指令 i 可供选择的流水线集合为 S_i,且 $|S_j|<|S_i|$,即指令 j 可供分配的流水线数更少。交换指令 i 和指令 j 后,n 条指令的负载为 $w_t'(w_t'=\underset{1\leqslant t\leqslant n,t\neq i,t\neq j}{=}w_t$,且 $w_i'=w_j$,$w_j'=w_i$),仍采用选择引理进行分配。j 之后的指令的分配都会受影响,假设对应的流水线分配序列为$\{k_1,\cdots,k_i',\cdots,k_j'\cdots,k_n'\}$,流水线的负载方差为:

$$Var_n'=\frac{2}{m}\sum_{i=1}^{n}w_i'L_{i-1,k_i'}+C$$

交换指令 i 和 j 后,从 1 到 $i-1$ 之间的指令的分配方式不会发生变化,因此$\sum_{t=1}^{i}w_t'L_{t-1,k_t'}=\sum_{t=1}^{i}w_tL_{t-1,k_t}$,则:

$$Var_n'=\frac{2}{m}\sum_{t=i}^{n}w_t'L_{t-1,k_t'}+C$$

$$Var_n'-Var_n=\frac{2}{m}\left(\sum_{t=i}^{n}w_t'L_{t-1,k_t'}-\sum_{t=i}^{n}w_tL_{t-1,k_t}\right)$$

若要使交换指令 i 和 j 后的流水线负载方差降低,则:

$$R=\sum_{t=i}^{n}w_t'L_{t-1,k_t'}-\sum_{t=i}^{n}w_tL_{t-1,k_t}<0$$

R 为评价交换指令次序获得更均匀负载的指标,如果 $R<0$,则说明交换后流水线负载的方差变小,即当前交换是有益的。

例如,对于表 4 所列的分配方式,如果交换 I1 和 I2,则交换前后的流水线序列分别为{AL,AL,BU,BU}和{AL,BU,AU,AL},对应的负载序列分别为:

$$\{L_{0,AL},L_{1,AL},L_{2,BU},L_{3,BU}\}=\{1,7,2,5\}$$

$$\{L_{0,AL},L_{1,BU},L_{2,AU},L_{3,AL}\}=\{1,2,3,5\}$$

$$R=35-50=-15<0$$

其满足交换条件。因此,交换 I1 和 I2 后再按照贪心算法进行分配,如表 6 所列,最终各流水线负载的方差更小,可以得到负载更均衡的流水线分配方案。

若要使 $R<0$,须降低$\sum_{t=i}^{n}w_t'L_{t-1,k_t'}$,该因子与指令的发射等待时间以及分配的流水线负载相关。如果只考虑指令的发射延时,对于选择受限的指令,即使其发射延时短,分配的优先级也较低。另外,对于每条流水线而言,其负载序列是递增的。延后分配选择受限的指令,可能会将其分配到一个负载已经很大的流水线上,得到一个较大的 $w_t'L_{t-1,k_t'}$,从而影响流水线上的最终负载分配情况,增加指令发射等待时间,对处理器性能带来了不利的影响。

表 6 非对称情形下不同分配方案的对比

Table 6 Comparison under asymmetric situation

Pipe	Plan1					Plan2				
	Init	分配 I1	分配 I2	分配 I3	分配 I4	Init	分配 I2	分配 I1	分配 I3	分配 I4
AU	3	3	3	3	3	3	3	3	6	6
BU	2	2	2	5	7	2	2	8	8	8
AL	1	7	11	11	11	1	5	5	5	7
Var	2/3	14/3	146/9	341/27	32/3	2/3	14/9	38/9	14/3	2/3

以上分析说明,在指令发射等待时间的基础上考虑指令的分配范围可以获得更好的分配效果。因此,在确定指令分配优先级时,将指令的选择范围也纳入考虑范围,允许选择范围受限但执行时间短的指令优先分配,在这种情况下,后续选择范围较广且执行时间长的指令也可能弥补某些负载较低的流水线短板,从而实现全局的分配优化,均衡各条流水线上的负载。在分配的过程中可以按照发射等待时间先将指令排好序,在此基础上,对分配受限的指令 i 和其前面的分配范围更广的指令$\{j||S_i|<|S_j|\}$按照交换指标 R 进行调整,则最终可以使流水线负载更加均匀,但这种方法需要每次计算 R,开

销很大，实用性不强。

综上所述，可以直接将指令发射等待时间以及流水线选择范围都纳入排序关键字的构造中，这里采用 $w_i(m+1-|S_i|)$（m 为总运算流水线数量）作为关键字对指令进行大小排序。如果同一关键字对应多条指令，则 $|S_i|$ 小的指令优先分配，如果 $|S_i|$ 也相同，则遵守程序固有顺序进行分配。这种排序方法仅参考了候选流水线的个数，虽然没有按照交换指标 R 查找真正符合要求的指令，但仍反映了指令分配的优先次序，而且省去了交换查找的过程。

按照这种排序方式，可得到 3.3 节举例中指令的排序关键字和分配方法，分别如表 7 和表 8 所列。可以看到，改进后的运算流水线分配方法可以在非对称情形下获得更均衡的分配效果。

表 7 非对称情形下指令的发射属性和排序结果

Table 7 Example of instructions under symmetric situation

指令	候选流水线	发射延时	$w_i(4-s_i)$	排序结果
I0	AU/BU/AL	6	6	2
I1	AL	4	12	1
I2	AU/BU/AL	3	3	4
I3	BU/AL	2	4	3

表 8 非对称情形下负载均衡的流水线分配方案

Table 8 Resource allocation under asymmetric situation

Pipe	Init.	分配 I1	分配 I0	分配 I3	分配 I2
AU	3	3	3	3	6
BU	2	2	8	8	8
AL	1	5	5	7	7
Var	2/3	14/9	38/9	14/3	2/3

4 实验设置

实验采用 Simplescalor 模拟器中的 sim-outorder 乱序执行模型对流水线分配方法进行评估。模拟器配置参数的选用贴近当前主流处理器，为 4 译码 7 发射的超标量架构（每拍可发射 3 条整数指令、2 条浮点指令和 2 条访存指令），具体参数如表 9 所列。

表 9 模拟器配置参数

Table 9 Microarchitectural parameters

Parameter	Value
RUU size	128 instructions
L/S queue size	64 instructions
Fetch queue size	16 instructions
Fetch width	4 instructions/cycle
Issue width	3 int instructions, 2 flt instructions, 2 mem instructions/cycle
Issue-wait queue	Int-wait queue:12, Flt-wait queue: 12, Mem-wait queue:12
Decode width	4 instructions/cycle
Commit width	4 instructions/cycle
Functional units	Refer to figure 1
Branch predictor	Gshare:4k-entry, 12-bit history BTB:128 sets, 4-way, LRU 5 cycles for extra branch misprediction
L1 ICache	32 kB, 2-way, 32B block, LRU, 1-cycle latency
L1 DCache	32 kB, 4-way, 32B block, LRU, 4-cycle latency
L2 Cache	512 kB, 8-way, 64B block, LRU, 11-cycle latency
Memory	50 cycles for addressing, 4 cycles/chunk for transporting. 2 memory ports, bus width is 8 B

测试程序选用 SPEC2000 测试集，包括所有整型测试程序和 9 段浮点测试程序。所有的测试程序使用 compaq dec 编译器编译，采用 ref 数据集作为测试输入，模拟时依据文献[11]的方法跳过了每段测试程序的初始化部分，统计后续 10 亿条指令的运行结果。

为了验证本文提出的流水线分配贪心方法的效果，将简单轮转分配（Round-Robin，RR）、改进的轮转分配（Optimazed-Round-Robin，ORR）、贪心分配（Load-Balance，LB）和非对称贪心分配（Load-Balance-Sort，LBS）作为对比。

（1）简单轮转分配方法

整数流水线、浮点流水线和访存流水线分别记录上次分配的流水线编号，当前指令根据上次分配的结果进行轮转，是一种静态分配方法。

（2）改进的轮转分配方法

浮点和访存指令的流水线分配与 RR 方法相同，整数流水线则优先参考当拍待分配指令的组合情况。算法中记录了当拍被唯一映射的指令独占的运算流水线编号，然后在指令可选范围中扣除已经独占的运算流水线，并在此基础上轮转。如果扣除独占的运算流水线后指令没有选择的余地，则直接轮转。

（3）贪心分配方法

按照 3.2 节所述方法，对于当拍已经准备好发射的指令，按照发射延时从大到小排序后使用选择引理进行分配，属于动态分配方法。

（4）非对称贪心分配方法

在模拟中按照 3.3 节所述方法构造指令排序关键字，排序后按照选择引理进行分配，属于动态分配方法。由于硬件构造关键字并排序的实现开销较大，这里仅用于实验对比分析。

5 效果评估

在图 1 所示的运算簇模型中，指令完成寄存器重命名后进入等待缓冲等待发射。指令源操作数都准备就绪后满足发射条件，完成运算流水线分配后进入对应运算流水线的发射缓冲等待发射执行。发射缓冲的深度是影响流水线分配算法效果的一个重要因素。发射缓冲深度过小不利于负载均衡流水线分配算法的发挥，过大不利于实现，因此首先在不同发射缓冲深度下对各种流水线分配算法进行评估，以测试程序 IPC 为主要评价指标。

5.1 发射缓冲深度为 4

首先横向对比 4 种运算流水线分配方法对程序 IPC 的影响。初始时，各条运算流水线发射缓冲的深度均为 4，实验测得 SPEC CPU 2000 测试集的程序 IPC 如表 10 所列。就测试结果而言，LB 分配方法获得的 IPC 最高，ORR 方法次之，RR 最低。其中，LBS 算法在流水线分配时根据关键字对指令进行排序，这会打破原始指令顺序，可能导致分配权重大而年轻的指令先分配，从一定程度上加剧了指令相关带来的影响，抵消了均衡分配带来的好处，因此在一些测试题中效果反而不如 LB 方法。而对于指令相互依赖情形较少以及依赖影响较低的测试题来说，LBS 方法无疑是最优的。

表 10 发射缓冲深度为 4 时的测试结果

Table 10 Experimental results of different resource allocation policies when depth of issue queues is 4

测试集	RR	ORR	LBS	LB
164.gzip	1.3699	1.3794	1.4211	1.4211
175.vpr	1.0588	1.0592	1.0877	1.0874
176.gcc	0.8368	0.8392	0.8506	0.8508
181.mcf	1.2745	1.3154	1.3470	1.3624
186.crafty	0.7386	0.7396	0.7500	0.7487
197.parser	1.0214	1.0223	1.0473	1.0489
252.eon	1.0075	1.0122	1.0167	1.0149
253.perlbmk	0.3377	0.3360	0.3462	0.3487
254.gap	1.3237	1.3089	1.3476	1.3609
255.vortex	0.9459	0.9521	0.9783	0.9778
256.bzip2	1.3755	1.3783	1.3975	1.3983
300.twolf	0.722	0.7072	0.7100	0.7307
average	1.0010	1.0042	1.0250	1.0292
168.wupwise	1.2126	1.2266	1.2261	1.2240
171.swim	1.4247	1.4234	1.4014	1.4005
172.mgrid	1.3199	1.3196	1.3178	1.3191
173.applu	1.2512	1.2520	1.2502	1.2499
177.mesa	1.0255	1.0762	1.0934	1.0874
178.galgel	1.246	1.2447	1.2731	1.2734
179.art	0.8609	0.8603	0.8723	0.8725
183.equake	0.8335	0.8545	0.8983	0.8501
189.lucas	1.0404	1.1292	1.0723	1.0767
average	1.1350	1.1541	1.1561	1.1504

对于浮点测试程序，在平均效果方面 LBS 方法是 4 种方法中最好的，这主要是因为浮点指令的执行时间普遍较长且部分浮点指令的发射等待延时也较长，例如浮点除法和平方根指令往往需要数十个周期才能完成，优先分配执行这种长延时指令带来的增益较为明显。从表 10 还可以看到，LB 方法对整数测试课题带来的好处比浮点测试程序明显。这是由两方面因素决定的：

(1) 整数流水线的数目多，可选择的范围大，利于负载均衡方法发挥其优势；

(2) 指令相关性影响不容忽视，浮点指令的执行时间普遍较长，因此带来的相关性阻塞情况也较多。

5.2 浮点发射缓冲深度为 6

由于硬件排序开销较大，LBS 只作为理论分析的一个参考对象，现只对比 RR，ORR 和 LB 这 3 种分配方法的性能。因为大多数浮点指令(例如浮点加法、乘法等)的执行时间为 6，为了减弱浮点指令相关性所带来的负面影响，将浮点发射缓冲的深度也增大到 6，其他发射缓冲深度保持不变，再进行横向比较实验，结果如表 11 所列。

表 11 浮点发射缓冲深度为 6 时的测试结果

Table 11 Results when depth of float issue queue is 6

测试集	RR	ORR	LB	测试集	RR	ORR	LB
164.gzip	1.370	1.374	**1.421**	168.wupwise	1.216	1.208	**1.231**
175.vpr	1.059	1.065	**1.087**	171.swim	1.470	1.465	**1.470**
176.gcc	0.837	0.841	**0.851**	172.mgrid	1.401	1.399	**1.483**
181.mcf	1.275	1.288	**1.362**	173.applu	1.286	1.286	**1.336**
186.crafty	0.739	0.741	**0.749**	177.mesa	1.027	1.057	**1.098**
197.parser	1.022	1.034	**1.049**	178.galgel	1.207	1.282	**1.292**
252.eon	1.008	1.010	**1.016**	179.art	0.864	0.859	**0.881**
253.perlbmk	0.336	0.342	**0.352**	183.equake	0.842	**0.856**	0.850
254.gap	1.323	1.328	**1.361**	189.lucas	1.069	**1.130**	1.124
255.vortex	0.946	0.952	**0.978**				
256.bzip2	1.376	1.376	**1.398**				
300.twolf	0.722	0.722	**0.740**				
average	1.001	1.006	**1.030**	average	1.154	1.171	1.196

各种流水线分配算法程序 IPC 更直观的比较如图 2 所示。从图中可以看到，对于浮点程序，LB 算法发挥出了优势，是 3 种方法中最好的。就整数测试程序而言，LB 算法相对于 RR 性能平均提升了 2.92%，相对于 ORR 性能平均提升了 2.40%；就浮点测试程序而言，LB 算法相对于 RR 性能平均提升了 3.68%，相对于 ORR 性能平均提升了 2.10%。

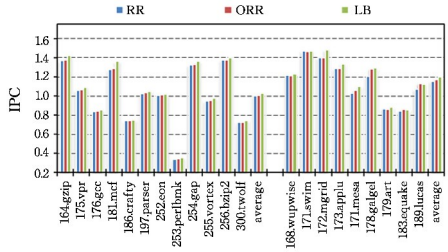

图 2 浮点发射缓冲深度为 6 时 3 种流水线分配算法的对比

Fig. 2 Comparison when depth of float issue queue is 6

纵向来看，当浮点发射缓冲增大到 6 后，相比 5.1 节中 4 条目的浮点发射缓冲，3 种流水线分配方法对每个测试程序的 IPC 都有不同程度的提升。其中，浮点程序 IPC 提升幅度较为明显，且 LB 方法获得的性能增益最为突出；部分整数测试程序中带有一些浮点指令，IPC 也能获得一些提升，但浮点指令比例较小，增益有限。

5.3 发射缓冲深度无限制

从上面的实验可以看出，增加发射缓冲深度对提升程序 IPC 是有帮助的，较大的发射缓冲可允许指令较早地进入发射缓冲中，因此可以参考更大的指令发射窗口做出运算流水线分配决策，以更好地保证各条运算流水线的负载均衡。

在给定 ROB 以及各缓冲的深度后，将各运算流水线的发射缓冲深度设置为与 ROB 深度一致，即发射缓冲深度无限制，测得的实验结果如图 3 所示。

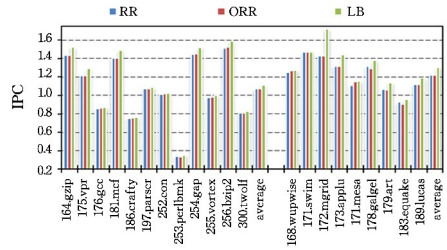

图 3 发射缓冲深度无限制时 3 种流水线分配算法的对比

Fig. 3 Comparison with limitless issue queue

从图 3 中可以看到，不限制发射缓冲深度时，相比其他两种分配方法，LB 方法更能利用这种特点，程序 IPC 获得了较大的提升。就整数测试程序而言，LB 相对于 ORR 性能提升幅度最明显的是 175.vpr，IPC 提升了 6.45%，所有测试程序平均提升了 3.44%；对于浮点程序，LB 相比 ORR 性能提升幅度最明显的是 172.ngrid，IPC 提升了 20.55%，所有程序平均提升了 6.2%。

5.4 发射等待时间

指令完成流水线分配后在缓冲队列中的等待时间是衡量分配方法效果的指标之一。如前所述，如果一种分配方法可以充分利用运算簇，保证指令的分配不会有偏向性，则每条运

算流水线的负载是比较均衡的，指令在发射队列中驻留的时间也会缩短。

实验测试了 3 种流水线分配方法下，所有测试程序中指令的平均发射等待时间。等待时间包括指令在等待缓冲和发射缓冲中驻留的时间总和，测试时发射队列的条目深度均为 6。实验结果如表 12 所列，单位为每条指令的平均驻留周期数。从表中可以直观地看到，LB 方法能更快地将指令送到运算部件，这与 LB 方法对程序 IPC 增益最优是吻合的。

指令发射等待时间还有一个比较重要的作用就是帮助确定缓冲深度，理想情况下，缓冲的深度应该不小于指令在其中的最大等待周期，但由于相关性阻塞的问题，这个值通常很大，没有实现意义。退一步，可以让缓冲的深度不小于指令驻留的平均周期数。

由于实验中统计的等待时间计入了指令在等待队列和发射队列中驻留的时间，因此等待缓冲和发射缓冲的深度总和应不小于指令平均等待延迟。实验结果中，当总的整数发射队列(包括等待缓冲和发射缓冲)深度为 16、总的浮点发射队列(包括等待缓冲和发射缓冲)深度为 18 时，已经能满足大多数情形的要求。因为整数指令中最长执行时间为 4 个周期，而浮点指令中多数执行时间为 6 个周期，去掉 12 条目的等待队列，将整数发射队列设置为 4 个条目、浮点发射队列设置为 6 个条目是较为合理的。

表 12 3 种流水线分配算法下指令发射的等待时间

Table 12 Instruction issuing delay under 3 different resource allocation policies

测试集	RR	ORR	LB	测试集	RR	ORR	LB
164. gzip	9.9	10.01	**9.69**	168. wupwise	15.75	15.96	**15.29**
175. vpr	15.77	15.77	**15.14**	171. swim	7.92	8.02	**7.79**
176. gcc	8.55	8.54	**8.28**	172. mgrid	12.24	12.24	**12.23**
181. mcf	12.21	12.09	**11.41**	173. applu	13.00	**12.97**	12.98
186. crafty	7.87	7.86	**7.61**	177. mesa	15.67	14.81	**14.62**
197. parser	13.62	13.52	**13.29**	178. galgel	18.22	**18.16**	18.28
252. eon	6.01	5.99	**5.89**	179. art	26.69	26.72	**26.22**
253. perlbmk	35.83	**35.34**	36.15	183. equake	18.69	18.75	**18.20**
254. gap	10.42	10.27	**10.19**	189. lucas	**17.12**	17.17	18.29
255. vortex	8.07	7.88	**7.44**				
256. bzip2	12.26	12.32	**12.05**				
300. twolf	**22.01**	22.07	22.11				
average	13.54	13.47	**13.27**	average	16.14	16.09	**15.99**

结束语 就指令调度和处理器资源分配问题而言，当前的研究以软件编译优化居多，本文则从硬件自动调度和实现的角度分析了这个问题。针对运算流水线分配的组合优化，首先进行了模型抽象，对其进行了理论分析，探讨了运算部件对称以及非对称情况下的负载均衡算法，然后在给定运算流水线模型的基础上测试了所提出的负载均衡分配方法的效果。

实验结果表明，负载均衡的运算流水线分配方法可以将指令更快地发射到执行部件，充分利用计算资源加速程序执行，而且处理器中的运算资源越多，发射缓冲深度越大，负载均衡方法获得的性能提升就越明显。另外，在具体实验中，本文还测试了指令发射等待时间，研究了发射缓冲深度对程序性能的影响，得到了评估确定发射缓冲深度的方法。

综上所述，负载均衡的流水线分配方法可以有效减缓现代超标量处理器中指令发射不公平所带来的负面影响，能更好地利用片上缓冲资源和计算资源，加速程序执行，提升处理器性能。

参 考 文 献

[1] DAI J, DAI G L, ZHANG S Q, et al. Integration of instruction scheduling and register allocation[J]. Journal of Tsinghua University(Sci & Tech), 2004, 44(1): 69-73.

[2] LIAN R Q, WU C Y, ZHANG Z Q. Integrating code optimization and instruction scheduling[J]. Chinese Journal of Computer, 2001, 24(7): 694-701.

[3] QU Q W, LIANG L P. Instruction parallel scheduling and implementation based on LLVM[J]. Microelectronics & Computer, 2013(11): 60-63.

[4] LAM M. Software pipelining: an effective scheduling technique for VLIW machines[J]. Acm Sigplan Notices, 1988, 23(7): 318-328.

[5] STOTZER E J, LEISS E L. Modulo scheduling without overlapped lifetimes[C]// ACM Sigplan/sigbed Conference on Languages, Compilers, and TOOLS for Embedded Systems. ACM, 2009: 1-10.

[6] LLOSA J. Swing Modulo Scheduling: A Lifetime-Sensitive Approach[C]// Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques, 1996. IEEE, 1996: 80-86.

[7] TAN M X, LIU X H, ZHANG J Y, et al. Non-overlapped modulo scheduling with optimized backtracking model[J]. Acta Electronica Sinica, 2012, 40(8): 1681-1686.

[8] UPTON M, HUFF T, MUDGE T, et al. Resource allocation in a high clock rate microprocessor[C]// International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 1994: 98-109.

[9] BUYUKTOSUNOGLU A, KARKHANIS T, ALBONESI D H, et al. Energy efficient co-adaptive instruction fetch and issue [C]// International Symposium on Computer Architecture, 2003. IEEE, 2003: 147-156.

[10] VILLAVIEJA C, JOAO J, MIFTAKHUTDINOV R, et al. A Hybrid Dynamic VLIW/OoO Processor: Technical Report: TR-HPS-2014-001[R]. 2014.

[11] PERELMAN E, HAMERLY G, VAN BIESBROUCK M, et al. Using SimPoint for accurate and efficient simulation[J]. ACM SIGMETRICS Performance Evaluation Review, 2003, 31(1): 318.

WANG Guo-peng, born in 1988, MSc, engineer. His main research interests include computer architecture, microprocessor design.