EPIC微体系结构的存储级并行执行模型的研究

邓让钰 陈海燕 邢座程 谢伦国 曾献君
(国防科学技术大学计算机学院 长沙 410073)

摘 要 描述了一种可以有效提高存储级并行(Memory Level Parallelism,MLP)的指令优化锁步执行模型——OLSM(Optimized Lock-Step execution Model)执行模型,并建立了一种能体现OLSM模型思想的层次存储结构.OLSM允许显示并行指令计算(Explicit Parallel Instruction Computing,EPIC)微处理器实现一定程度的乱序执行,解决了传统超长指令字(Very Long Instruction Word,VLIW)锁步执行的缺陷,可以充分利用结构中的大量计算和存储资源,最大化隐藏存储延迟、提高MLP.

关键词 显示并行指令计算;单位延迟;非单位延迟;存储级并行;优化的锁步执行模型
中图法分类号 TP302

The Research on Memory-Level Parallelism Execution Model in EPIC Architecture

DENG Rang-Yu CHEN Hai-Yan XING Zuo-Cheng XIE Lun-Guo ZENG Xian-Jun
(*School of Computer Science, National University of Defence Technology, Changsha* 410073)

Abstract This paper presents an instruction Optimized Lock-Step execution Model(OLSM) and builds a memory hierarchy which can embody the essence of this model. In OLSM EPIC microprocessor, instruction can be executed out-of-order, so shortcoming of tradition VLIW lock-step execution is resolved. OLSM can make use of the abundance computation and memory resources to hide memory access latency and improve Memory Level Parallelism (MLP) at best.

Keywords EPIC; unit access latency; non-unit access latency; memory level parallelism; optimized lock-step execution model

1 引 言

EPIC体系结构的本质是VLIW,同时还吸取了超标量处理器的许多新思想.通过指令集显式表达并行性信息,如每个指令可以有多个操作(MultiOP),程序执行计划(Plan Of Execution, POE)通过指令集传递给硬件,这样既保持了VLIW由编译器静态构造POE的特点,同时又具有一定的动态特性,使处理器硬件更简单[1-2].

VLIW结构中,多操作指令直接利用操作的非原子性特点:操作的执行时间至少为1个时钟周期,同一发射组的各个操作需要同时写结果,因此,即使指令存在反相关也可以调度到同一个多操作(指令组)中.另外,同一指令的两个操作也可以存在反相关,例如,寄存器原子交换可以实现成两个复制操作、互相写.

EPIC结构中,某些操作可能需要多个时钟周期才能完成,编译器必须能够理解这些延迟,才能获得正确和高质量的代码调度.最佳情况是,编译器知道操作发射后读操作数和写结果的确切时刻,这样就可以将代码调度成:每个操作的目的寄存器的旧值在操作发射之前,到操作执行完之间的时间段仍可用(一般来说这是违背顺序语义的).只在操作结束

收稿日期:2005-06-16;修改稿收到日期:2006-05-04.本课题得到国家"八六三"高技术研究发展计划项目基金(2002AA110020)和国家自然科学基金(90207011)资助.**邓让钰**,男,1972年生,副教授,主要研究方向为高性能计算机、微处理器体系结构和集成电路设计.E-mail:rydeng@21cn.com.**陈海燕**,女,1967年生,副教授,主要研究方向为微处理器设计与实现.**邢座程**,男,1964年生,教授,博士生导师,主要研究领域为微处理器设计与实现.**谢伦国**,男,1947年生,教授,博士生导师,主要研究领域为高性能计算机、微处理器体系结构.**曾献君**,男,1966年生,教授,博士生导师,主要研究领域为微处理器设计与实现.

时保留这些操作的结果寄存器，这样可以减少临时寄存器的压力.

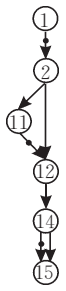
(1) UAL(Unit Assumed Latency)延迟语义

RISC 和超标量处理器都采用 UAL 延迟语义，这是一种顺序模型，不能把延迟暴露出来. 程序的虚调度时间(指令执行的时间单位)假定每个操作的延迟都为 1，即假定每条指令发射前，上一指令已经完成. 如果以前的指令延迟大于 1，处理器必须能够保证后续相关指令获得正确的值，通常办法是，停顿后续的相关指令.

图 1(a)是一段用 UAL 延迟语义调度的代码，其相关语义如图 1(b).

Cycle	Operation
1	$r1=load(r2)$
2	$r1=load(r3)$
3	
4	
5	
6	
7	
8	
9	
10	
11	$r4=fmul(r1, r5)$
12	$r4=fadd(r1, r6)$
13	
14	$r7=fmul(r4, r9)$
15	$r7=fadd(r7, r8)$

(a)



(b)

图 1　UAL 代码调度的相关语义

(2) NUAL 延迟语义

NUAL(Non-Unit Assumed Latency)延迟语义是指，结构中

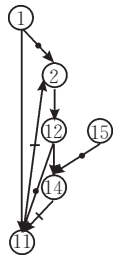
(a) 至少有一个操作的延迟为 L，$L>1$；

(b) 当前操作完成前，可发射后面的 $L-1$ 条指令，且程序语义正确.

用 NUAL 语义调度上述相同代码，其相关语义见图 2. T_1 和 T_2 连续发射两条指令，尽管第一条指令在 T_{10} 才产生出结果.

Cycle	Phase1 Operation	Phase2 Operation
1	$v1=load(r2)$	
2	$v2=load(r3)$	
3		
4		
5		
6		
7		
8		
9		
10		$r1=v1$
11	$v3=fmul(r1, r5)$	$r1=v2$
12	$v4=fadd(r1, r6)$	
13		$r4=v4$
14	$v5=fmul(r4, r9)$	$r4=v3$
15	$v6=fadd(r7, r8)$	
16		$r7=v6$
17		$r7=v5$

(a)



(b)

图 2　NUAL 代码调度的相关语义

EQ(equals)是最严格的 NUAL 形式，即读操作数的准确时间就在发射时刻，产生结果的时刻也在指定的延迟时间点上. 另一种 NUAL 是 LEQ (less or equals)，写结果事件的时刻范围在 1 个时钟周期到假定的延迟之间，可以简化精确中断的实现. 使用 LEQ 延迟语义调度的代码即使实际操作比假定时间快，程序也是正确的，可实现二进制兼容.

目前，VLIW 和 EPIC 都主要使用 NUAL 延迟语义，允许结构在性能、硬件简单性以及不同硬件实现间的兼容性方面进行折衷.

2　基本 NUAL 执行模型

2.1　延迟缓冲执行模型

由于每个时钟周期都可能发射新操作，对于 EQ 操作，为了不违背 EQ 语义，不能过早写结果，也不能因为晚写而阻塞功能流水线. 解决这个问题的一种方法是功能部件的输出使用延迟缓冲区，使写事件延迟一定的时间.

2.2　延迟停顿执行模型

NUAL 操作在指定时间内不能按时完成，最简单的控制办法是在产生结果前停止推进指令执行的虚时间，暂停指令发射，也不再读、写结构寄存器，这就是延迟停顿(latency stalling). 假设一个操作的调度延迟为 P，在功能部件中执行的实际延迟为 V，则最大需要停顿 $(V-P)$ 个时钟周期，这就保证了消费者肯定可以取得正确的操作数. 这种方法的优点是不需要检查源和目的操作数的使用情况，不需要记录寄存器使用踪迹，简化了硬件设计. 另外，这种方法可保证 NUAL 操作的完成时间不会比假定的短，这也是必须的，因为在任何调度虚时间点上，读或写事件相对于写事件进行了重新排序后，程序正确性就可能得不到保证. 调度得很好、很紧凑的代码总希望事件在与之相关的写事件后立即发生.

延迟停顿在实现上除了需要保证正确性外，还要使插入的停顿尽可能少. Rau 曾提出了一种修改延迟停顿的简单方法，该方法可以保证在虚时间上与编译器调度一致[3].

虚时间停顿后，处理其它可以按时完成的操作方法有两种：(1) 冻结功能流水线上的可按时完成操作，但不阻塞不可按时完成的 NUAL 操作. 由于不再发射新指令，中断也不能使用该冻结的功能部件，因此，这种方法可能存在死锁；(2) 许可按时完成的操作继续前进，而不考虑虚时间是否推进. 这种

方法的优点是,当前不能完成的操作可以充分利用停顿虚时间停顿周期,来减少它完成的时间,甚至可能完全消除其延迟,从而使该操作成为可按时完成操作.

延迟停顿的缺点是,即使不存在相关性,指令的发射也可能停顿.不能按时完成操作的其它处理方法还有许多,这些方法性能很可能更高,但代价也更大.例如,某些锁步方法,只在必须的时候才停止发射指令,性能高了,代价也大了.

传统的顺序锁步(in-order interlock)模型,常用于顺序性 ISA 结构,对操作的虚拟完成时间没有任何控制,因而不能支持 NUAL,这是与延迟停顿方法的根本区别.

2.3 基本锁步执行模型

NUAL 锁步与延迟停顿的方法相似,区别在于需要记录不能按时完成操作的轨迹.延迟停顿需要停止发射指令(写目的的虚时间点),而 NUAL 锁步仅仅把目的寄存器标识为无效,然后继续发射新指令,操作完成时,寄存器再标识为有效.准备发射指令时,只在读到一个无效寄存器(如写后读互锁)或第二个操作准备写一个无效寄存器(写后写互锁)时,才停止指令发射和虚时间推进.一旦寄存器可用,立即恢复指令发射.NUAL 锁步确保了对寄存器的所有读、写事件的相对顺序,即程序正确性语义.

尽管 NUAL 锁步有许多优点,但延迟停顿对嵌入式处理器来说也是必不可少,毕竟简单性对这类处理器来说至关重要,并且重新编译程序也是可以接受的.原则上用 MultiOp 和 NUAL 语义可以实现乱序执行,但会破坏 EPIC 设计思想,需要避免[3].

对于原子操作,NUAL 锁步模型与顺序锁步模型相同,即操作的目的寄存器在指令发射时标识为无效,如果指令需要读或写一个无效寄存器,那么指令不再发射.

2.4 执行模型比较

NUAL EQ 语义提供给编译器的信息最确定,因而优化的机会最大[4].实际上,与短延迟操作存在反相关的长延迟 EQ 操作甚至可以在后者之前发射.相反,LEQ 语义中,如果操作之间存在反相关,则不能在反相关解决之前发射.另外,EQ 功能流水部件需要延迟缓冲区和一个用于选择流水线或延迟缓冲区的开关,而 LEQ 操作产生结果的时刻比预计时间早,于是结果不需保存到延迟缓冲区.如果应用本身对延迟增长不十分敏感,那么使用 EQ 比较有利,但会增加寄存器的使用压力.

如果实际延迟与假定延迟之间的比值较大,延迟停顿模型的性能会很差.如果调度存在某些难以克服的问题,如事件之间存在相关且事件的调度顺序不能改变,那么 NUAL 锁步模型比延迟停顿模型性能上要好得多.锁步模型中,程序中复杂的控制流可能导致实际处理过程不流畅,这是因为编译代码调度不太可能同时使所有路径都最优,而只能保证某些可能性最大的路径比可能性小的路径要优,实际上,很多情况在编译时是很难确定的.这样,编译器能开发的理论 ILP 往往比硬件实际执行时的 ILP 要高得多.

3 基本 EPIC 执行模型

基本 EPIC 结构采用锁步模型,假定所有非存储操作的延迟为 1 个时钟周期,通过延迟缓冲区和停止存在相关的操作来维护程序正确性.具体执行模式为:任意指令 I_j 能够发射的前提条件是指令 I_j 之前的所有指令都已经或者能够与指令 I_j 同时发射.为了能够尽可能提高性能,如果指令不能写结果则需要进入缓冲区,等待写结果的时机.因此,基本的 EPIC 模型实际上是带延迟缓冲的锁步模型.

能够同时发射的指令一定属于同一个指令组,我们称这些并行发射的指令为一个发射组或执行组(executing group),即一个 MultiOp.编译器保证同一指令组中各指令之间不存在写后读相关(Read After Write,RAW)和写后写相关(Write After Write,WAW),同一发射组内部不需要检查 RAW 和 WAW 寄存器相关性,简化了冲突检查逻辑.

假定某 EPIC 微处理器共有 11 条流水线(4 条存储流水线,2 条整数流水线,2 条浮点流水线和 3 条分支流水线).主流水线采用 8 站,分别为 FETCH、PREDICT、ISSUE、RENAME、REG、EXE、COMMIT 和 WRB.整个流程上可以分为前端和后端,前端包括 FETCH 和 PREDICT,完成取指令流任务;后端包括 ISSUE、RENAME、REG、EXE、COMMIT 和 WRB,分别实现指令发射、寄存器换名、读操作数、执行、提交和写回.

指令后端流水线在执行(EXE)阶段之前(不包括 EXE)都是顺序的,即顺序发射.从发射组的角度看,顺序流出发射组,多条流水线中的同一个发射组中的多条指令在 EXE 阶段之前都是同步前进的,而且不同发射组的指令在 EXE 阶段之前没有交叉,保持严格的先后顺序.但是,发射组进入 EXE 阶段之

后，由于存储操作的延迟不确定，同一发射组的指令到达 COMMIT 站的时机也不确定，因此 EXE 站需要设计采用乱序执行(out-of-order executing)的策略. 为了支持精确中断，同一发射组需要实现指令顺序提交[4](in-order committing).

因此，基本 EPIC 执行模型的特点是：

(1) 顺序发射，乱序执行，顺序提交，顺序写回. 即各主流水站严格锁步，同一发射组的指令必须按序提交，以支持精确中断.

(2) 各功能流水线的 COMMIT 站为等深度的 FIFO 缓冲区，数据旁路和数据相关性判定必须针对每项有效缓冲区.

3.1　基本 EPIC 执行模型的存储层次

各指令流水线中，只有存储操作的延迟不确定. Cache 命中时，可以在 1 个时钟周期内提供数据，与整数运算的操作延迟相匹配；Cache 不命中时，存储访问处理时间可能比较长. 为了使系统性能最大化，各流水线附带一个延迟写结果缓冲区，一旦结果有效，将结果写入延迟缓冲区. 只有同一发射组的所有指令的结果都生产出来后，指令组才在同一时刻提交，因此各功能流水线的 COMMIT 站实际上采用等深度的缓冲队列.

存储层次及各流水线的基本结构见图 3. 存储操作在 EXE 站完成对 $L1$ 数据 Cache($L1D$)的访问，失效时还需要访问 $L2$ Cache，甚至主存，取得结果后，填充 $L1D$，然后进入 COMMIT. $L1D$ 与 $L2$ Cache 保持严格的低、高级关系. 图 3 中 SBI (System Bus Interface)为总线接口部件.

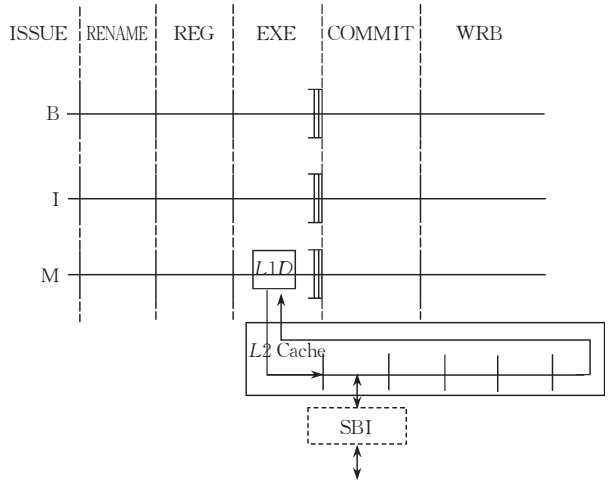


图 3　基本 EPIC 执行模型

3.2　基本 EPIC 执行模型的 *MLP* 分析

通过 EXE 站乱序执行来尽可能地隐藏访问延迟，要求有足够的存储指令发送到存储部件. 因此，数据 Cache 系统的实际 MLP 水平极大程度上受限于延迟缓冲区的深度(d)和存储流水线数目(w)，即 $MLP^{\text{Base}}_{D\text{-Cache}}=w\times(d+1)$.

最大 MLP 用 $MLP^{\text{Base}}_{\max}$ 表示，则

$$MLP^{\text{Base}}_{\max}=MLP_{I\text{-Cache}}+MLP^{\text{Base}}_{D\text{-Cache}}=MLP_{I\text{-Cache}}+w\times(d+1) \tag{1}$$

(1) 单发射结构中，$w=1$，且基本 EPIC 执行模型就是延迟停顿执行模型，则

$$MLP^{\text{Base}}_{\max}=MLP_{I\text{-Cache}}+d+1 \tag{2}$$

(2) 当 $w=4$，

$$MLP^{\text{Base}}_{\max}=MLP_{I\text{-Cache}}+4\times(d+1) \tag{3}$$

$MLP_{I\text{-Cache}}$ 与指令 Cache 的设计有关，在 EPIC 结构中主要强调预取指令流. 尽管预取本身并不是开发 MLP 的途径，但从 $L2$ Cache 处理角度看，指令预取将产生大量的未完成访存操作，因此对 MLP 有一定作用.

衡量程序性能的重要参数是平均 MLP，用 $MLP^{\text{Base}}_{\text{mean}}$ 表示. 由于 EPIC 锁步的本性，要使流水线因为存储数据供应不上导致停顿的前提是：首次访问未完成存储操作的目的寄存器. 该首次访问的时刻即为 Load-to-Use 时刻. Load-to-Use 的平均时间间距可以反映微处理器结构、代码优化及程序行为特性，用 $Dist$ 表示. 影响 $MLP^{\text{Base}}_{\text{mean}}$ 的因素有 $Dist$、存储资源的使用率 η 等.

$$MLP^{\text{Base}}_{\text{mean}}=f:Dist\times\eta\times w=\begin{cases}MLP_{I\text{-Cache}}+\eta\times w\times Dist, & Dist<d+1\\ MLP_{I\text{-Cache}}+\eta\times w\times(d+1), & Dist\geq d+1\end{cases} \tag{4}$$

4　OLSM 执行模型

上节我们已经提到，基本 EPIC 锁步模型中 EXE 站乱序执行策略要求有足够的存储指令发送到存储部件，才有可能提高 MLP. 由于 RENAME、REG 以及进入 COMMIT 站都是锁步的，要使每条存储流水线都有一条以上的存储指令，实际上既受延迟缓冲区深度的影响，也与程序本身的特点密切相关.

基本 EPIC 模型的另一个问题是：数据相关性控制和数据旁路逻辑极复杂. 在基本 EPIC 模型中，COMMIT 站采用 3 深度缓冲区，意味着缓冲区每一项都有可能成为结果数据的“寄存坞”，取源操作数的旁路逻辑需要 1200 个比较器[5]，代价极大，电路时延很难控制. 相关性控制与旁路逻辑息息相关，旁

路所在的流水站肯定是相关性控制所覆盖的流水站. 3 深度延迟缓冲区旁路相当于有 3 站 COMMIT,无论如何控制 WAW 和 RAW,记分牌逻辑都是复杂的,且极不规整.

实际上,各流水线有如下几个基本特点:

(1)整数操作可以在 EXE 站完成运算.

(2)大部分存储操作可以在 EXE 站完成运算.

(3)少数存储操作尽管不能在 EXE 站完成所有运算,但可以在 COMMIT 站判定是否异常(数据 Cache 或 TLB 不命中、指令非法或数据错).

根据上述特点,作如下修正:REG 站登记记分牌,在 COMMIT 站,只检查指令是否产生异常,只有产生异常或结果已经生产出来的指令才可以清记分牌,不保证 COMMIT 指令的数据可以立即通过 WRB 站写回结构寄存器. 此模型去掉了基本 EPIC 模型中的延迟缓冲区,同时可确保指令后端各流水站都是锁步执行的,这就是基于 NUAL 语义的优化的锁步执行模型(Optimize Lock Step Execution Model,OLSM),其硬件(缓冲区、记分牌及其它硬件)代价很少,控制非常简单.

在 OLSM 模型中,存储层次及各流水线结构如图 4 所示. 如果 *L*1*D* 命中,*L*1*D* 可以直接提供数据,如果 *L*1*D* 不命中,*L*1*D* 并不直接访问 *L*2 Cache,而是在指令提交后,由控制流水线控制是否访问 *L*2 Cache,此时 *L*2 Cache 作为数据的提供者,直接写回数据(寄存器),同时填充 *L*1*D*.

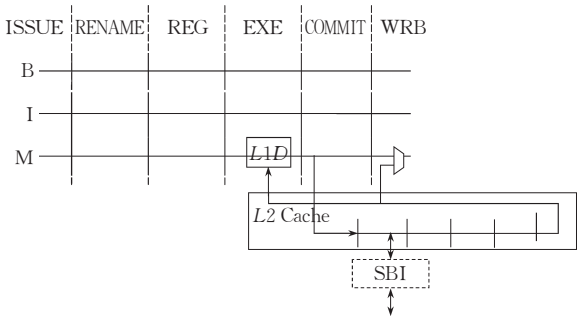


图 4 OLSM 锁步执行模型

基本的 NUAL 锁步执行模型与延迟缓冲区模型相比,前者逻辑更简单,但后者性能占优,因为后者可开发的 *MLP* 更高. NUAL OLSM 模型既保持了基本的 NUAL 锁步执行模型硬件的简单性,同时克服了基本 EPIC 模型中不能充分开发 *MLP* 的问题,其思想基础是:存储操作的某些异常可以异步处理,COMMIT 异常处理与数据写回在时间点上是可以分切的,指令提交不应该赋予其它更多含义,如数据返回.

OLSM 模型的基本特点是:

(1)顺序发射,顺序执行,顺序提交,乱序写回(寄存器).

(2)各主流水站严格锁步,COMMIT 为异常检测站,同一发射组的指令按序提交,以支持精确中断,另外,相关性和旁路简化了.

(3)更新结构寄存器的时机不确定,访存数据乱序返回.

(4)消费者报错,异步处理 MCA(Machine Check Abort).

(5)*L*1*D* 与 *L*2 Cache 不保持纯粹意义上的低层和高层次关系.

4.1 OLSM 执行模型的存储层次

在 OLSM 模型中,存储系统的最大特点是:对大部分消费和生产数据的功能部件,如对整数功能部件而言,*L*1*D* 是第一级 Cache,*L*2 Cache 不是完整意义上的第二级 Cache. 另外,对大量应用的研究表明浮点应用对存储访问的敏感性远不如整数应用[2],主要原因是浮点应用对数据的处理和加工趋于批量化,比较适合流水突发处理. 同时,DEC Alpha21264 提出整数部件与浮点部件分簇的结构,有利于提高微处理器的主频,因此,对于浮点功能部件而言,*L*2 Cache 更适合作为浮点数据的第一级数据 Cache.

从数据流角度看,传统意义上的 Cache 层次如图 5(a),而 OLSM 模型下的 Cache 层次如图 5(b).

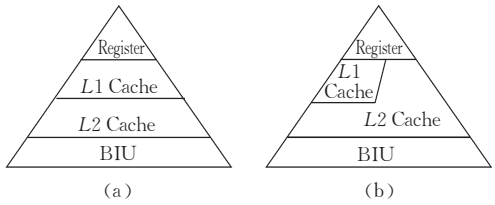


图 5 Cache 存储层次

4.2 OLSM 执行模型的 *MLP* 分析

OLSM 模型中,*MLP* 可以得到极大限度的开发,限制因素减低至很小,这些限制因素有:

(1)应用程序的特点;

(2)*L*2 Cache 微流水线数 w' 和深度 L;

(3)*L*2 Cache 与系统总线接口部件 SBI 之间的请求队列深度 p.

最大 *MLP* 用 $MLP_{\max}^{\mathrm{OLSM}}$ 表示,则

$$MLP_{\max}^{\mathrm{OLSM}}=MLP_{I\text{-}\mathrm{Cache}}+MLP_{D\text{-}\mathrm{Cache}}^{\mathrm{OLSM}}$$
$$=MLP_{I\text{-}\mathrm{Cache}}+w'\times(L+2)+p \quad (5)$$

平均 *MLP* 用 $MLP_{\mathrm{mean}}^{\mathrm{OLSM}}$ 表示,则

$$MLP_{\text{meam}}^{\text{OLSM}}=f:Dist\times\eta\times w'=\begin{cases}MLP_{I\text{-Cache}}+\eta\times w'\times Dist,\\ \qquad w'\times Dist<w'\times(L+2)+p\\ MLP_{I\text{-Cache}}+\eta\times(w'\times(L+2)+p),\\ \qquad w'\times Dist\geq w'\times(L+2)+p\end{cases}\tag{6}$$

5　OLSM 执行模型与基本 EPIC 执行模型的性能比较

5.1　实验环境

YHFT64-1 是一款 64 位通用微处理器，采用 EPIC 结构，在设计中使用了 OLSM 执行模型，其中 $w=w'=4$，$L=5$，$P=16$，$MLP_{I\text{-Cache}}=8$，则

$$MLP_{\max}^{\text{OLSM}}=MLP_{I\text{-Cache}}+MLP_{D\text{-Cache}}^{\text{OLSM}}=52\tag{7}$$

YHFT64-1 基本功能模拟器，采用基本 EPIC 模型，其中 $w=4$，$d=3$.

5.2　两种模型的 $MLP_{\max}$ 分析比较

(1) 当 $w\times(d+1)>w'\times(L+2)+p$ 时，

$$MLP_{\max}^{\text{Base}}>MLP_{\max}^{\text{OLSM}}\tag{8}$$

(2) 当 $w\times(d+1)<w'\times(L+2)+p$ 时，

$$MLP_{\max}^{\text{Base}}<MLP_{\max}^{\text{OLSM}}\tag{9}$$

(3)假设存储流水线数目与 $L2$ Cache 微流水线数目相等，延迟缓冲区的深度与 $L2$ Cache 微流水线深度相等，即 $w=w'$，$d=L$，则

$$MLP_{\max}^{\text{Base}}<MLP_{\max}^{\text{OLSM}}\tag{10}$$

基本 EPIC 模型中，MLP 与 $L2$ Cache 的微流水线及请求队列深度无关，而与延迟缓冲区的深度相关. 从硬件设计角度看，延迟缓冲区对指令相关性判定、数据旁路的影响巨大，不利于实现，也不利于提高微处理器主频，d 的选择一般不超过 4；由于 $L2$ Cache 的微流水线深度在设计上与执行模型无关，基本 EPIC 模型和 OLSM 模型中基本一致，L 的选择一般为 5～7；$L2$ Cache 与 SBI 之间的请求队列可以采用 FIFO 结构，对主频的影响不大，p 可以达到 16 以上. 因此，理论上，在相同流水线数目和相同发射规则下，$MLP_{\max}^{\text{Base}}<MLP_{\max}^{\text{OLSM}}$，OLSM 模型能开发的 MLP 比基本 EPIC 模型要大得多.

$MLP_{\max}^{\text{Base}}(d=3)$ 和 $MLP_{\max}^{\text{OLSM}}$ 在 CPU SPEC2000 测试中的结果如图 6. 由于存储流水线可以进行基本的整数运算，即执行 A 类指令(包括 ALU 和 MMX 指令)，在基本 EPIC 模型中，A 类指令降低了存储指令对延迟缓冲区的使用率，$MLP_{\max}^{\text{Base}}$ 平均只有 5.34. 在 OLSM 模型中，A 类指令的延迟确定，除了影响 EXE 站和 COMMIT 两站外，不影响存储指令的其它资源的使用，$MLP_{\max}^{\text{OLSM}}$ 平均为 18.2.

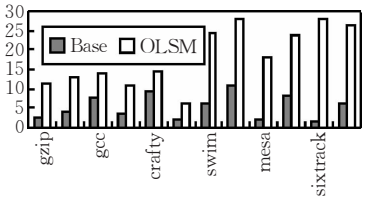


图 6　两种模型下的 $MLP_{\max}$

5.3　两种模型的 MLP_{mean} 分析比较

首先，平均使用距离对 MLP 的影响较大. 图 7 中，当 $w=w'$ 时，由于 $d<L+p+1$，OLSM 模型无论在 $Dist$ 较小还是较大情况下，$MLP_{\text{mean}}^{\text{OLSM}}>MLP_{\text{mean}}^{\text{Base}}$. 整数应用对访存延迟较为敏感，$Dist$ 较小，浮点应用对访存延迟敏感性差些，但对带宽敏感，$Dist$ 较大. 因此，图 7 说明了 OLSM 模型能够较好地适应各种不同类型的应用.

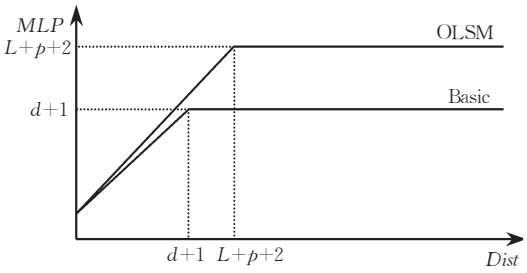


图 7　EPIC 基本锁步执行模型与 OLSM 执行模型中的平均 MLP

另外，OLSM 模型中新的存储层次更能体现编译器对数据局部性的判断，而且浮点 load 全部旁路 $L1D$，$L2$ Cache 作为浮点数据一级 Cache，总体上看，针对 OLSM 模型优化的代码有利于提高 MLP. 我们作了 CPU SPEC2000 测试，其结果如图 8 所示. 图 8 表明：$MLP_{\text{mean}}^{\text{Base}}$ 的平均值只有 1.33，而 $MLP_{\text{mean}}^{\text{OLSM}}$ 的平均值可达到 4.59. 因此，尽管 OLSM 模型中的最大绝对延迟比基本 EPIC 模型中要长些，但平均 MLP 较高，存储延迟隐藏较充分，有助于提高实际 IPC，因此，总体性能比基本 EPIC 模型反而要好得多.

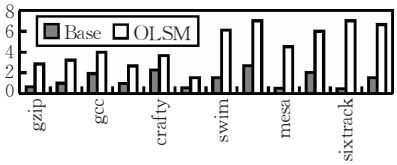


图 8　两种模型下的 MLP_{mean}

6　结　论

本章在讨论编译延迟语义的基础上，深入研究

和分析了基于 EPIC 的基本执行模型及其在更大限度开发 *MLP* 方面存在的不足,提出了一种硬件实现代价低且能更大限度地开发 *MLP*、隐藏存储延迟的新的指令锁步执行模型——OLSM 模型. 基于该 OLSM 执行模型,进一步提出了一种更适合 EPIC 机制的层次存储结构. 最后,理论和实际 SPEC2000 测试结果表明:与基于基本 EPIC 执行模型的传统存储结构相比较,这种基于 OLSM 锁步执行模型的新型层次存储结构能充分利用结构中的大量存储资源,更大限度地开发并行或顺序程序(线程)中的 *MLP*.

参 考 文 献

[1] Colwel R P et al. A VLIW architecture for a trace scheduling compiler. IEEE Transactions on Computers, 1988, 37(8): 967-979

[2] Deng Rang-Yu, Xie Lun-Guo, Xing Zuo-Cheng. Exploiting memory level parallelism in EPIC architecture. Journal of Computer Research and Development, 2004, 41(Supplement): 259-264(in Chinese)
(邓让钰,谢伦国,邢座程. 开发 EPIC 结构中的存储级并行. 计算机研究与发展, 2004, 41(增刊): 259-264)

[3] Schlansker M S, Rau B R. EPIC: An architecture for instruction-level parallel processors. HPL Technical Report: HPL-1999-111, 2000

[4] Schlansker M S, Rau B R. EPIC: Explicitly parallel instruction computing. IEEE Micro, 2000, 33(4): 44-58

[5] Samuel D Naffziger, Glenn Colon-Bonet et al. The implementation of the itanium 2 microprocessor. IEEE Journal of Solid-State Circuits, 2002, 37(11): 24-32

DENG Rang-Yu, born in 1972, associate professor. His research interests include high performance computer system, micro-architecture and micro-electronics design.

CHEN Hai-Yan, born in 1967, associate professor. Her research interest is micro-architecture.

XING Zuo-Cheng, born in 1961, professor. His research interests include micro-professor and micro-electronics design.

XIE Lun-Guo, born in 1947, professor. His research interests include high performance computer performance, micro-processor architecture.

ZENG Xian-Jun, born in 1966, professor. His research interest is micro-processor architecture.

Background

This work is part of the project "High Performance Processor for General Purpose", which is supported by the National High Technology Research and Development Program (863 Program), grant No. 2002AA110020. This project is to resolve the key technology in the CPU design, develop a chip called YHFT64-1 which is compatible with the IA-64 instruction set architecture, setup a test environment for this chip. The authors research the problem of instruction execution model and memory hierarchy.

This work is also part of the project "The Design Technology of the Power Self-Adaptive in SOC", which is supported by the National Natural Science Foundation of China (NSSC), grant No. 90207011. This project is to research the self-adaptive low power technology in SOC through compiler directed resource controlling. This paper resolves the problem how the compiler schedule instruction sequences, control the use of resources to reduce power.