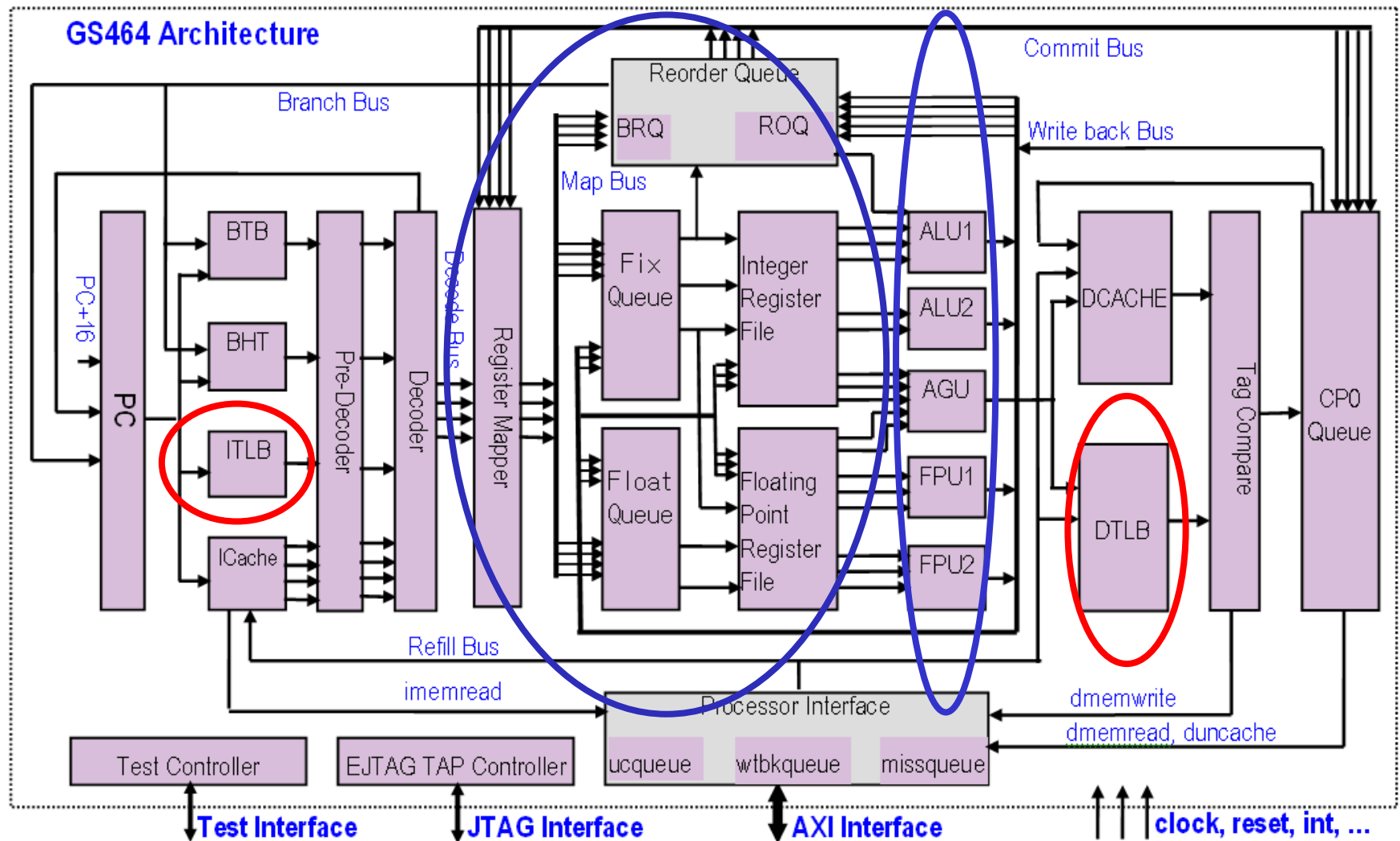


计算机体系结构

胡伟武、汪文祥

龙芯2号处理器核结构图



存储管理

- 虚拟存储的基本原理
- MIPS处理器对虚存系统的支持
- LINUX操作系统的存储管理
- TLB的性能分析和优化

一个访存例子

- 一个程序片段
 - `array = (int*)malloc(0x1000);`
 - `For (i=0;i<1024;i++) array[i] = 0;`
- 软件功能
 - 分配一个1024项的一维整数数组array，并初始化为0;
- 硬件过程？
 - 数组地址分配在什么地址？
 - 数组存在什么地方（内存/硬盘）？
 - 什么时候分配？什么时候存？
 - 虚地址和物理地址如何转换？

利用存储空间分布进行攻击的例子

- 利用缓冲区溢出进行攻击的例子
- 怎么防范？

```
void fa(void) {  
    ...  
    function fb(str);  
    ...  
}  
  
void fb(char *str) {  
    ...  
    char buffer[16];  
    ...  
    gets(buffer);  
    ...  
}
```



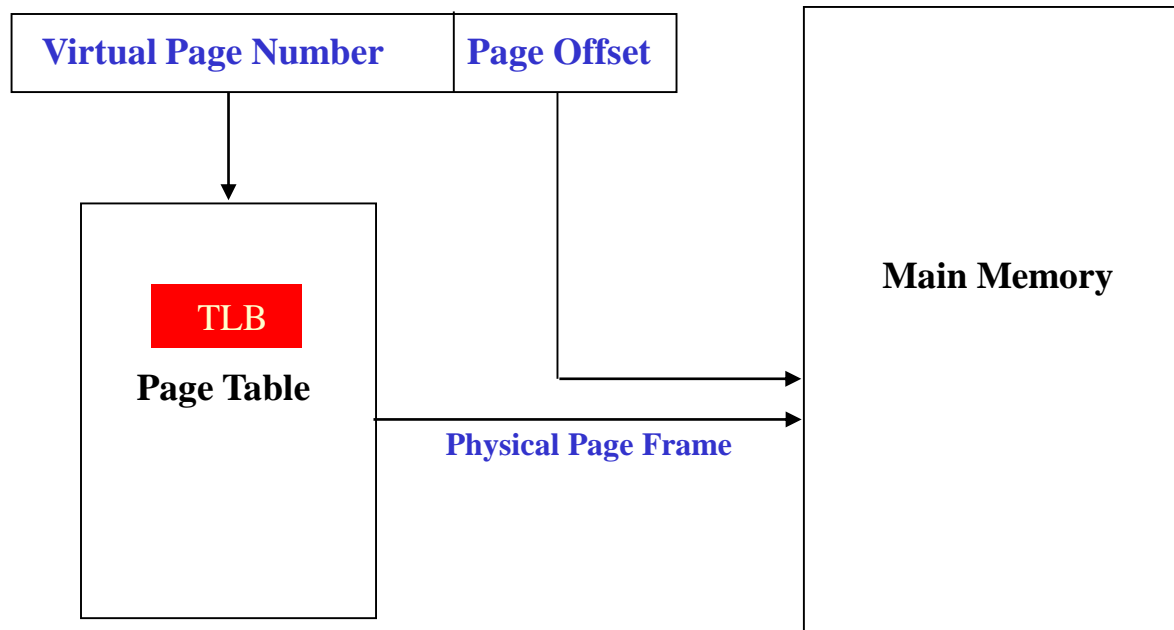
虚拟存储的基本原理

虚拟存储原理

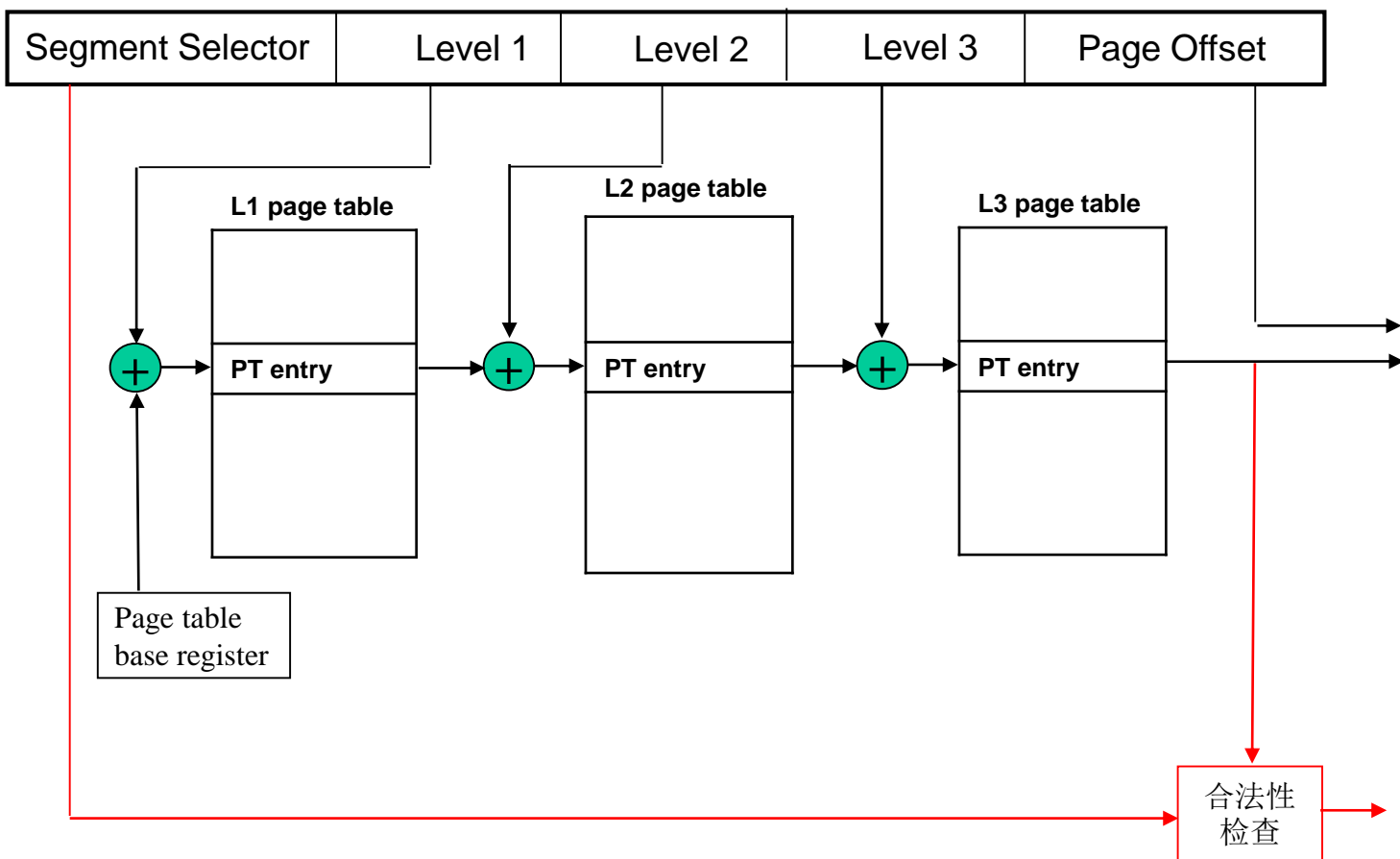
- 虚拟存储是计算机系统发展过程中有里程碑作用的事件
 - 多进程环境下统一的编程空间
 - 多进程环境下的共享与保护
 - 支持大于实际物理内存的编程空间
- 虚实地址分开，建立一种从虚地址空间映射到物理内存的机制
 - 把两个层次的存储转换为一个层次的存储
 - 物理内存实际上是磁盘的一个Cache

虚实地址转换与页表

- 在页的范围内，虚实地址相等
- TLB是页表的cache



多级页表



TLB

- **TLB实际上是操作系统中页表的Cache**
 - TLB主要负责完成用户空间到物理空间的转化
 - 一般与Cache访问同时进行
 - TLB内容：虚地址（Cache的Tag），物理地址（Cache的Data），保护位（Cache的状态）
- **TLB失效处理**
 - TLB失效时需要把相应页表内容从内存取到TLB
 - TLB失效时硬件（如X86的page walker）和软件（如MIPS的特殊例外）来填充TLB

Cache和虚拟存储

比较内容	Cache	虚拟存储
调度单位	块（16B-128B）	页（4KB-64KB）
命中延迟	1-3时钟周期	50-150时钟周期
失效延迟	8-150时钟周期	1,000,000-10,000,000时钟周期
失效率	0.1-10%	0.00001-0.001%
映射前地址	25-45位物理地址	32-64位虚地址
映射后地址	14-20位cache地址	25-45位物理地址
映射者	硬件	硬件（TLB）+操作系统
组织方式	直接相联、组相联、全相联	全相联、组相联（Page Coloring）
替换方式	随机、FIFO、LRU	LRU
写回方式	Write-back、Write-through	Write-back

分清两种映射关系

- **TLB是页表的Cache**
 - 负责地址转换，一页管**4KB**以上数据
 - 页表由操作系统管理，存在系统空间
- **内存是硬盘的Cache**
 - 负责数据缓存，一字节就是一字节
 - 用户程序的数据在用户空间

MIPS处理器对虚存系统的支持

MIPS处理器对虚拟存储的支持

- 分段，段内分页
- **TLB**
- 特殊的控制寄存器
- 特殊指令
- 专用的例外入口

MIPS的访问权限

- **User mode**
 - **EXL=0 and ERL=0 and KSU=10**
- **Supervisor mode**
 - **EXL=0 and ERL=0 and KSU=01**
- **Kernel mode**
 - **EXL=1 or ERL=1 or KSU=00**

MIPS存储空间分段情况

- 32位模式

地址范围	容量	映射方式	Cached	访问权限
0xe0000000-0xffffffff	0.5GB	查找TLB	Yes (TLB)	Kernel
0xc0000000-0xdfffffff	0.5GB	查找TLB	Yes (TLB)	Kernel, Supervisor
0xa0000000-0xbfffffff	0.5GB	地址-0xa0000000	No	Kernel
0x80000000-0x9fffffff	0.5GB	地址-0x80000000	Yes (Config)	Kernel
0x00000000-0x7fffffff	2GB	查找TLB	Yes (TLB)	Kernel, Supervisor, User

MIPS的TLB及相关控制寄存器

- 32位模式
- 全相联
- 32-64项

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	0	0	0	0	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0				
0						MASK											0								
VPN2																G	0			ASID					
0	PFN																			C		D	V	0	
0	PFN																			C		D	V	0	

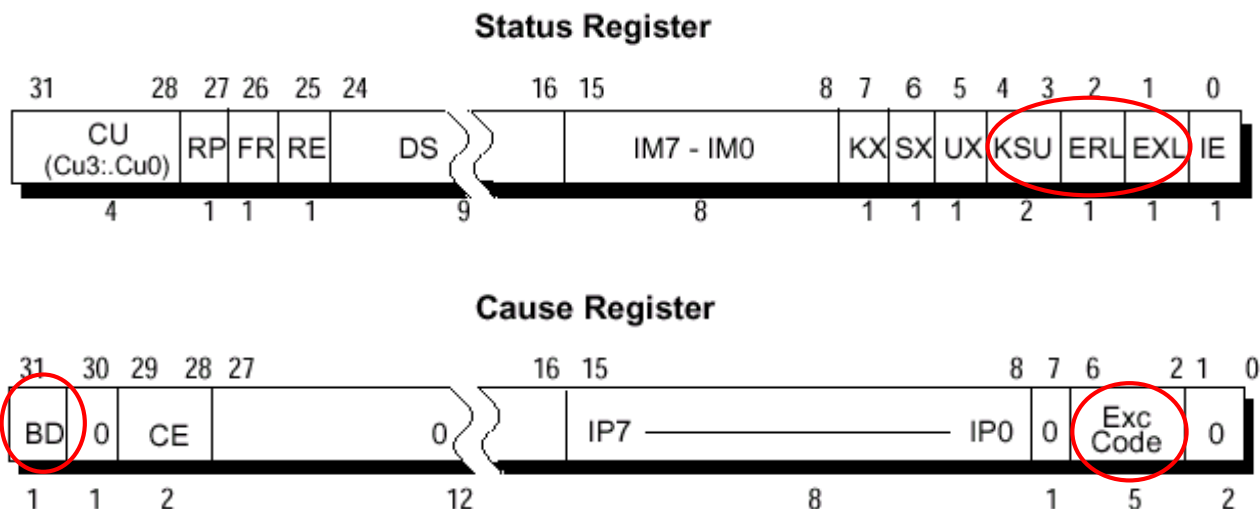
	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 9	0 8	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 0
Pagemask	0						MASK										0															
EntryHi	VPN2																		0				ASID									
EntryLo0	0	PFN																		C				D	V	G						
EntryLo1	0	PFN																		C				D	V	G						
Index	P	0																		Index												
Random	0																		Random													
Wired	0																		Wired													
EPC	EPC																															
BadVAddr	Bad Virtual Address																															
Context	PTEBase								BadVPN2																0							

与TLB管理有关的指令

- **MFC0, MTC0**
 - 在通用寄存器和控制寄存器之间搬运数据
- **TLBR**
 - 以Index寄存器为索引把TLB内容读到PageMask、EntryHi和EntryLo0/1等寄存器
- **TLBP**
 - 检查EntryHi中指定的虚页是否在TLB中
- **TLBWR, TLBWI**
 - 分别以Random和Index寄存器为索引把Pagemask、EntryHi和EntryLo0/1寄存器的内容写入TLB

发生TLB例外时硬件处理过程

- 置BadVaddr, Context, EntryHi,
- PC=例外入口地址
 - TLB Refill入口=0x80000000
 - 其它入口=0x80000180
- 置Status, Cause



TLB例外类型

- **Refill**
 - 如果查找TLB没有找到一個虛地址匹配（VPN2+ASID/G）
 - 例外入口：80000000（除非exl=1）
- **TLB invalid**
 - 如果找到一個虛地址匹配項，但其v=0
 - 例外入口：80000180
 - 細分為兩種：TLBL for loads, TLBS for stores
- **TLB modify**
 - 如果找到一個虛地址匹配項，其v=1，但D=0且訪問為store
 - 例外入口80000180

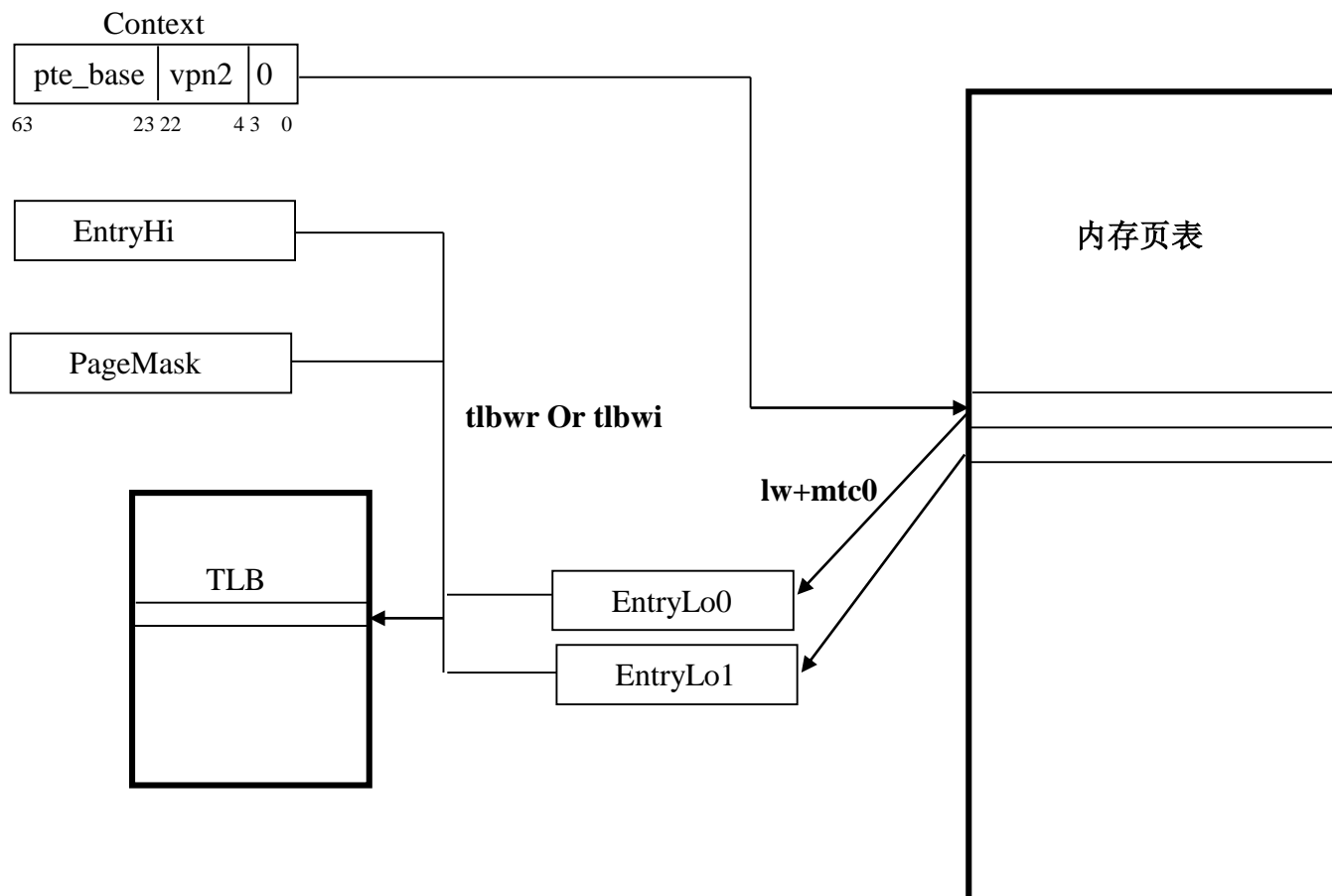
例外返回

- 例外处理器在核心态下进行
 - 不允许在核心态下执行一条用户指令
 - 不允许在用户态下执行指令核心指令
- 例外返回的两种方式
 - **jr+mtc0**: mtc0必须在jr的延迟槽中
 - **eret**: eret没有延迟槽

一种虚拟存储实现方式

- 使得发生例外时context寄存器指向页表中相应项
 - 一维线性页表
 - 内存页表每项8个字节，每对页面占用16字节页表
 - 进程切换时操作系统更改context寄存器中pte_base域，使其指向该进程的页表基地址（pte_base是进程上下文的一部分）
- 一维线性页表需要很大的空间，不能全部分配物理内存
 - 放在kernel mapped的kseg2/kseg3段
 - 需要解决TLB refill重入问题

TLB refill过程



Pagemask在操作系统初始化时由OS设置（固定页操作系统）

Pte_base在进程切换时由OS设置

Vpn2和**EntryHi**在缺页时由CPU设置

TLB refill代码

```
set noreorder
.set noat
TLBmissR4K:
DMFC0    k1, C0_CONTEXT    # (1)
NOP      # (2)
LW        k0, 0(k1)        # (3)
LW        k1, 8(k1)        # (4)
MTC0     k0, C0_ENTRYLO0    # (5)
MTC0     k1, C0_ENTRYLO1    # (6)
NOP      # (7)
TLBWR    # (8)
ERET     # (9)
.set at
.set reorder
```


LINUX操作系统的存储管理

Linux/mips虚拟存储管理

- 虚拟地址空间
- 页表和TLB
 - 页表组织
 - 例外处理函数
 - 一些优化

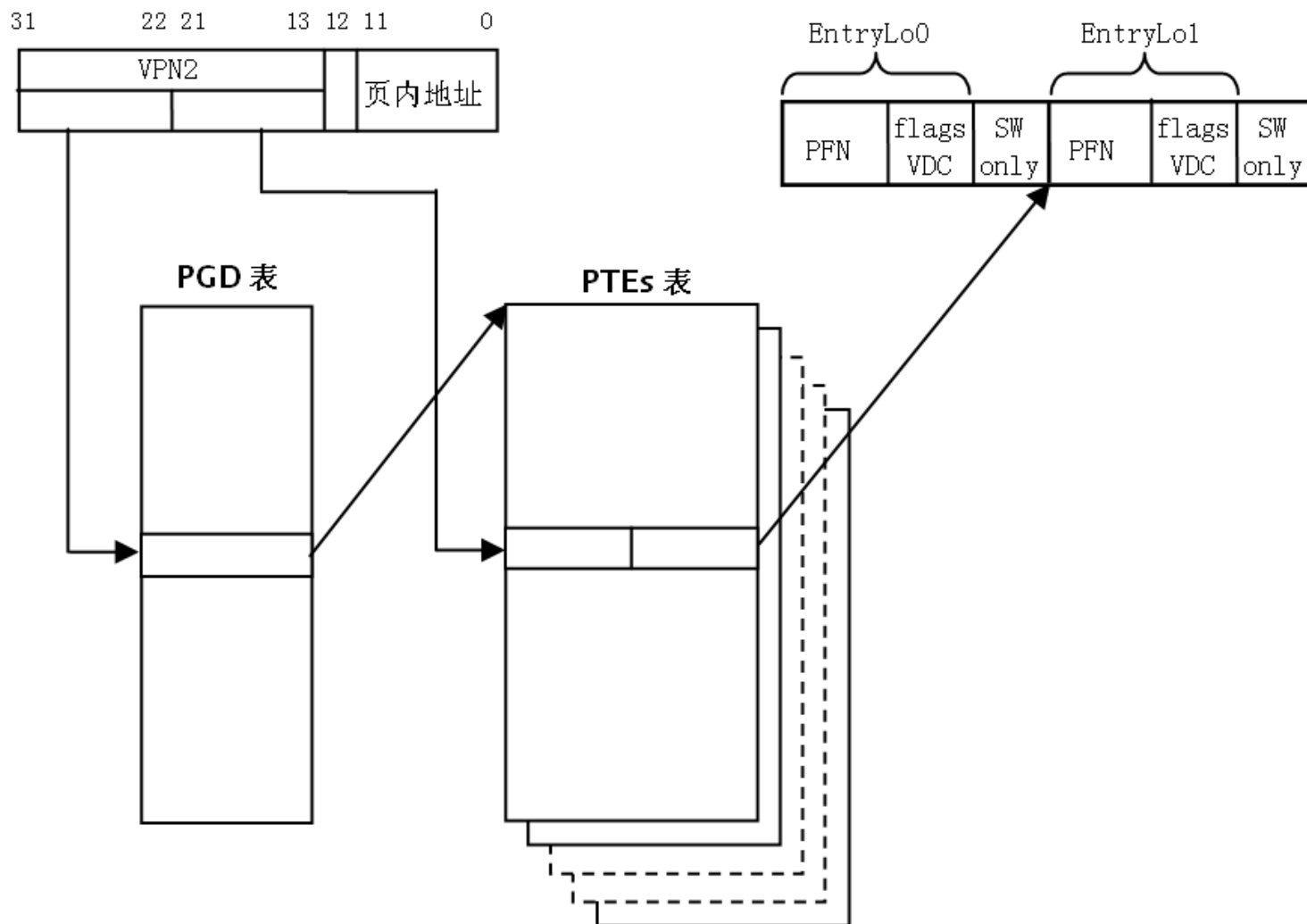
Linux/MIPS虚拟地址空间安排

0xFFFFFFFF	mapped(kseg2/3) 内核模块 vmalloc
0xC000 0000	
0xA000 0000	Unmapped uncached(kseg1) Uncached phy mem, ROM,Register,PCI IO/MEM etc.
0x8000 0000	
0x0000 0000	Unmapped cached(kseg0) 内核数据和代码
	32-bit user space(kuseg) (2GB)
0x0000 0000	

内存中的页表组织（32位情况）

- 两级页表，每项4个字节
 - **PFN**: 物理帧号
 - **Flags**: V、C、D
 - **Exts**: 软件扩展位，用于维护一些硬件没有实现的功能，例如ref位，modified位
- 页表存放在kseg0
 - 页表访问不引起TLB例外
 - 页表存储空间在使用到的时候分配
- 每个进程的页表基地址存放在进程上下文中

Linux/MIPS的两层页表



Linux的tlb重填代码（共18条指令）

```
mfc0 k0, CP0_BADVADDR
srl  k0, k0, 22
lw   k1, pgd_current
sll  k0, k0, 2
addu k1, k1, k0
mfc0 k0, CP0_CONTEXT
lw   k1, (k1)
srl  k0, k0, 1
and  k0, k0, 0xff8
addu k1, k1, k0
lw   k0, 0(k1)
lw   k1, 4(k1)
srl  k0, k0, 6
mtc0 k0, CP0_ENTRYLO0
srl  k1, k1, 6
mtc0 k1, CP0_ENTRYLO1
tlbwr
eret
```

```
#取发生tlb miss的地址
#最高10位是第一级页表的索引
# 取页表入口指针
#每项4个字节,所以索引*4=偏移
#*k1指向下一级页表入口
#context包含失效的虚页号
#取出第二级页表

#算出第二级页表的偏移

#成对存放,一个偶数页
#加一个奇数页
#移出6位软件用的位
#写入偶数页表项

#写入奇数页表项
#写入TLB的一个随机项
#异常返回
```

例子

- **Array=(int*)malloc(0x1000)**
 - 用户程序**malloc(0x1000)**返回虚地址**0x450000**
 - 操作系统在该进程的**vma_struct**链表中记录地址范围**0x450000-0x4501000**为已分配地址，可读可写
- **For (I=0;I<1024;I++) Array[i] = 0**
 - 用户程序试图写**0x450000**，**TLB**查找失败，引起**tlb refill**例外
 - **Tlb refill**从相应页表位置取入页表内容填入**TLB**。但该页表还没初始化
 - 例外返回到用户程序，重新开始访问
 - **TLB**表项找到，但是无效，发生**TLB Invalid**例外
 - 操作系统查找**vma_struct**，判断该地址已分配，处于可写状态，因此为它分配物理页面，并将物理地址填入页表，更新**TLB**
 - 例外返回，写操作再次重试，成功。
 - 用户程序继续写**0x450004,008...**，因为**TLB**项已经存在，将全速运行，除非中间发生进程切换导致其**TLB**项被换出。如果发生被换出的情况，再次运行时将发生**refill**例外从页表取得有效内容，不会再发生**invalid**例外(因此,**refill**频率 >> **invalid**)
- 为什么要分成两次例外？

Linux/MIPS中TLB例外的处理

- `tlb refill exception(0x80000000):`
 - (1) `get badvaddr, pgd`
 - (2) `pte table ptr = badvaddr >> 22 < 2 + pgd ,`
 - (3) `get context, offset = context >> 1 & 0xff8 (bit 21-13 + three zero),`
 - (4) `load offset(pte table ptr) and offset+4(pte table ptr),`
 - * (5) `right shift 6 bits, write to entrylo[01],`
 - (6) `tlbwr`
- `tlb modified exception(handle_mod):`
 - (1) `load pte,`
 - * (2) `if _PAGE_WRITE set, set ACCESSED | MODIFIED | VALID | DIRTY,`
`reload tlb, tlbwi`
 - `else DO_FAULT(1)`
- `tlb load exception(handle_tlb1):`
 - (1) `load pte`
 - (2) `if _PAGE_PRESENT && _PAGE_READ, set ACCESSED | VALID`
`else DO_FAULT(0)`
- `tlb store exception(handle_tlbs):`
 - (1) `load pte`
 - * (2) `if _PAGE_PRESENT && _PAGE_WRITE, set ACCESSED | MODIFIED | VALID | DIRTY`
`else DO_FAULT(1)`

例子

- 程序段
 - `array = (int*)malloc(0x10000);`
 - `for (i=0;i<16384;i++) array[i] = 0;`
- 页大小为4KB，发生了多少次例外？
 - 地址空间为64KB，16页
 - 发生8次TLB Refill（MIPS TLB一项两个连续虚页）和16次TLB Invalid例外
- 页大小为16KB，发生了多少次例外？
 - 发生2次TLB Refill和4次TLB Invalid例外
- 页大小为4KB，上述程序段执行完后发生进程切换
 - 切换后再执行`for (i=0;i<16384;i++) array[i] = i;`
 - 如果进程切换时TLB项被替换，只发生8次refill例外
 - 如果进程切换时，array被调出内存，发生8次TLB Refill和16次TLB Invalid例外，invalid例外处理时把array从硬盘调到内存
- 如果页表也被调出内存？

不忘初心

- 为什么访存指令的执行这么复杂？
 - 虚拟化：结构设计复杂一些，用户用起来简单一些
 - 结构设计：虚实地址转换，支持比实际内存容量更大的访问空间
 - 用户程序：每个进程都感到整个CPU和内存都是“我的”
- 进一步虚拟化
 - 运行多个OS，让每个OS都感到整个CPU和内存都是“我的”
 - 两次地址转换：guest VA \Rightarrow guest PA (host VA) \Rightarrow host PA，两个不同的TLB以及一个“影子TLB” (guest VA \Rightarrow host PA)
 - 地址例外时guest OS和host OS切换
- OS的“摩尔定律”终结：2020年OS的复杂度也到头了
 - 把CPU最后一点“家当”（核心态功能）也虚拟化掉了

TLB的优化

TLB性能分析和一些优化

- 防缓冲区溢出攻击优化
- 硬件性能优化
- 软件性能优化

利用TLB的保护机制防范攻击

- 利用缓冲区溢出进行攻击的例子
- 龙芯处理器通过可执行保护防止缓冲区溢出攻击
 - **TLB增加可执行位**

```
void fa(void) {  
    ...  
    function fb(str);  
    ...  
}  
  
void fb(char *str) {  
    ...  
    char buffer[16];  
    ...  
    gets(buffer);  
    ...  
}
```

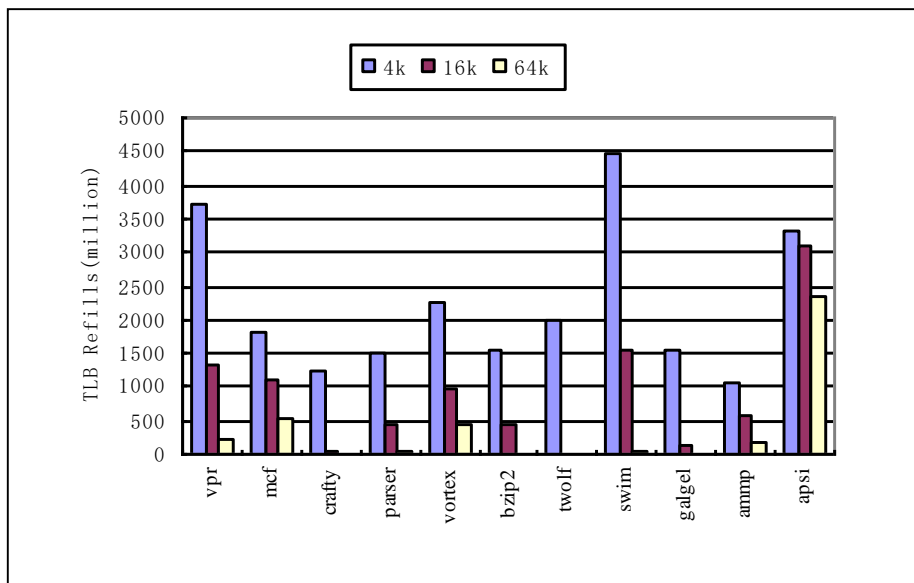


TLB相关性能数据

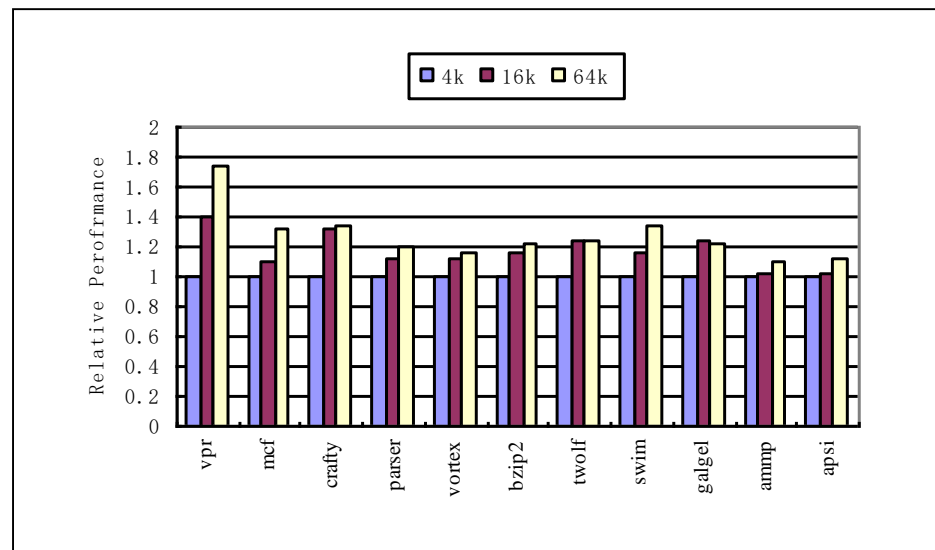
- **TLB miss**处理的时间可以占到高达**40%**的运行时间，占**40~90%**的内核运行时间
- **SPEC CPU2000**大约**1/4**的程序有比较显著的**TLB miss**
 - 早期多数CPU的TLB是全相连的**32-128**项，如果每项**4KB**，能映射的空间只有几百**KB**，越来越难满足现代程序的需求
- **性能优化方法**
 - 增加**TLB**覆盖空间大小，降低**TLB**失效概率
 - 降低**TLB**失效开销

增加页大小后性能显著提高

- 增加页大小后，TLB失效明显减少
 - 16KB页时128页有2MB
 - 通过软件配置pagemask可以增加/减少页大小

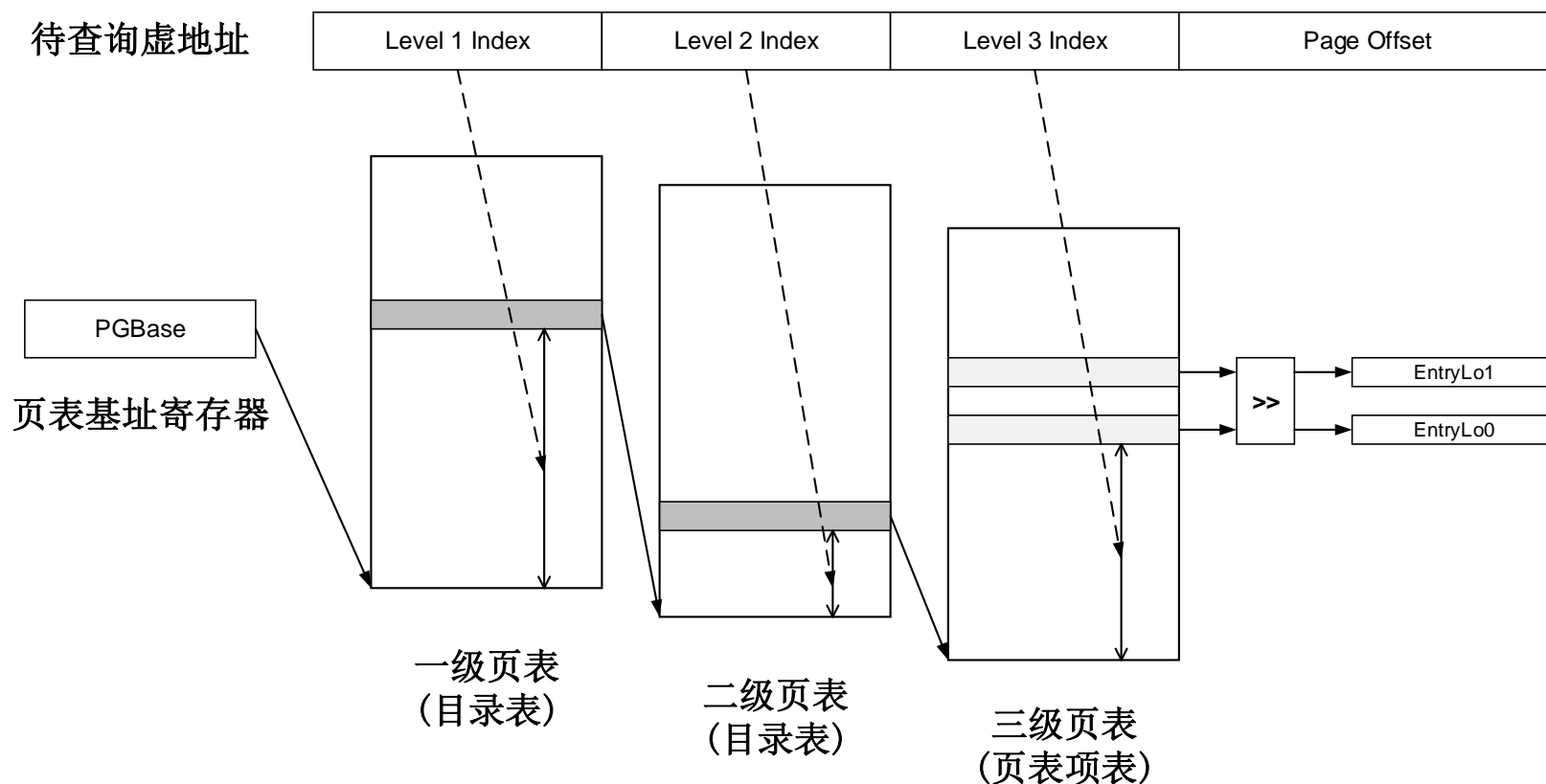


(a) TLB refills of 4KB, 16KB, and 64KB page size



(b) Performance of 4KB, 16KB, and 64KB page size

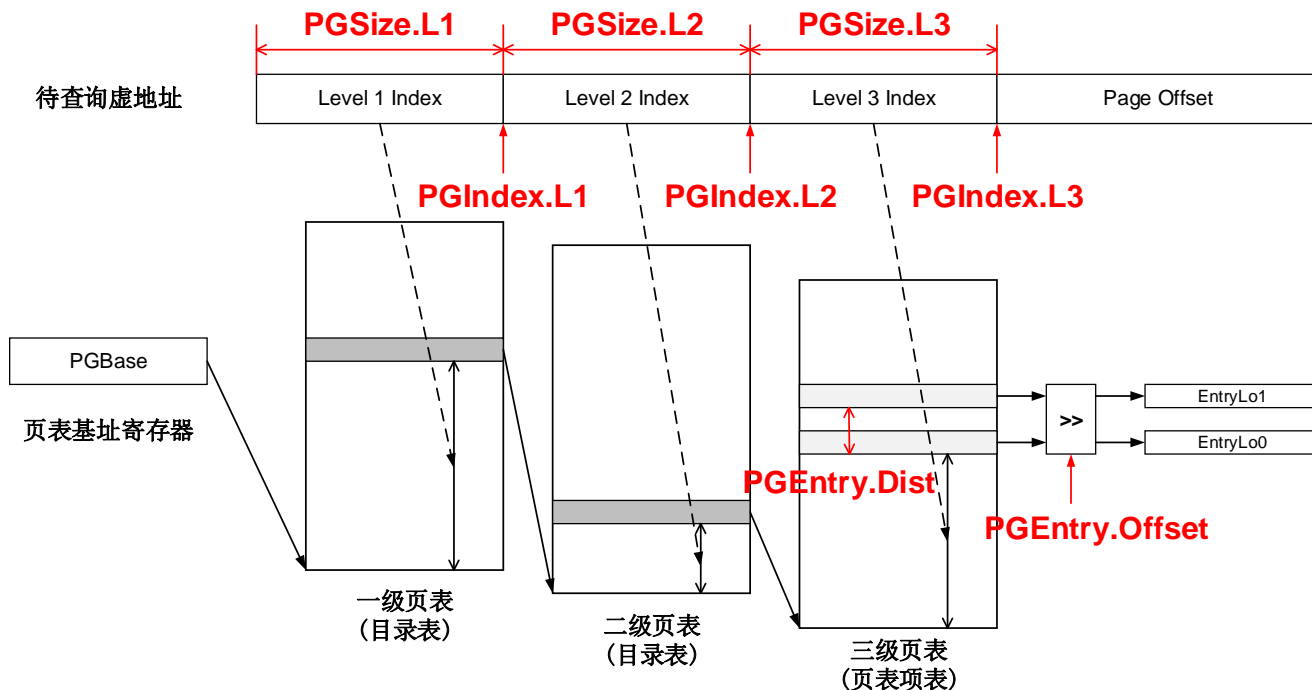
MIPS-Linux多级页表查找过程示意



- MIPS原Context寄存器设计不再适合多级页表的快速查找
 - 原设计针对一级页表

加速MIPS-Linux多级页表查找(1)

- 首先通过软件可配置寄存器来描述具体的页表结构
 - **PGBase**: 页表基址寄存器
 - **PGSize**: 描述各级页表的大小
 - **PGIndex**: 描述各级页表索引值在待查询地址中的起始位置
 - **PGEntry**: 描述奇偶页表项之间位置关系, 以及页表中软件用的位的宽度
- **PGBase**是进程上下文一部分, 其余寄存器操作系统启动时配置



加速MIPS-Linux多级页表查找(2)

- 其次定义专门的页表访存指令
 - **LDDIR dest, src, #N**: 将src寄存器中的值作为第N级目录页表的基址, 同时根据CP0.BadVaddr中存放的待查询地址以及新增的PGSize、PGIndex配置寄存器信息, 访存得到第N+1级页表的基址, 写入到dest寄存器中。
 - **LDPTE src, #0/#1**: 将src寄存器中的值作为页表项页表的基址, 同时根据CP0.BadVaddr中存放的待查询地址以及新增的PGSize、PGIndex、PGEntry配置寄存器信息, 访存得到偶数号/奇数号页表项的内容, 将页表中纯粹软件用的位移除后, 写入到CP0.EntryLo0/CP0.EntryLo1中。

MIPS-Linux TLB重填代码优化

优化前

```
mfc0 k0, CP0_BADVADDR
srl k0, k0, 22
lw k1, pgd_current
sll k0, k0, 2
addu k1, k1, k0
mfc0 k0, CP0_CONTEXT
lw k1, (k1)
srl k0, k0, 1
and k0, k0, 0xff8
addu k1, k1, k0
lw k0, 0(k1)
lw k1, 4(k1)
srl k0, k0, 6
mtc0 k0, CP0_ENTRYLO0
srl k1, k1, 6
mtc0 k1, CP0_ENTRYLO1
tlbwr
eret
```

优化后

```
dmfc0 k0, pgbase

lddir k0, k0, 1 (一级页表)

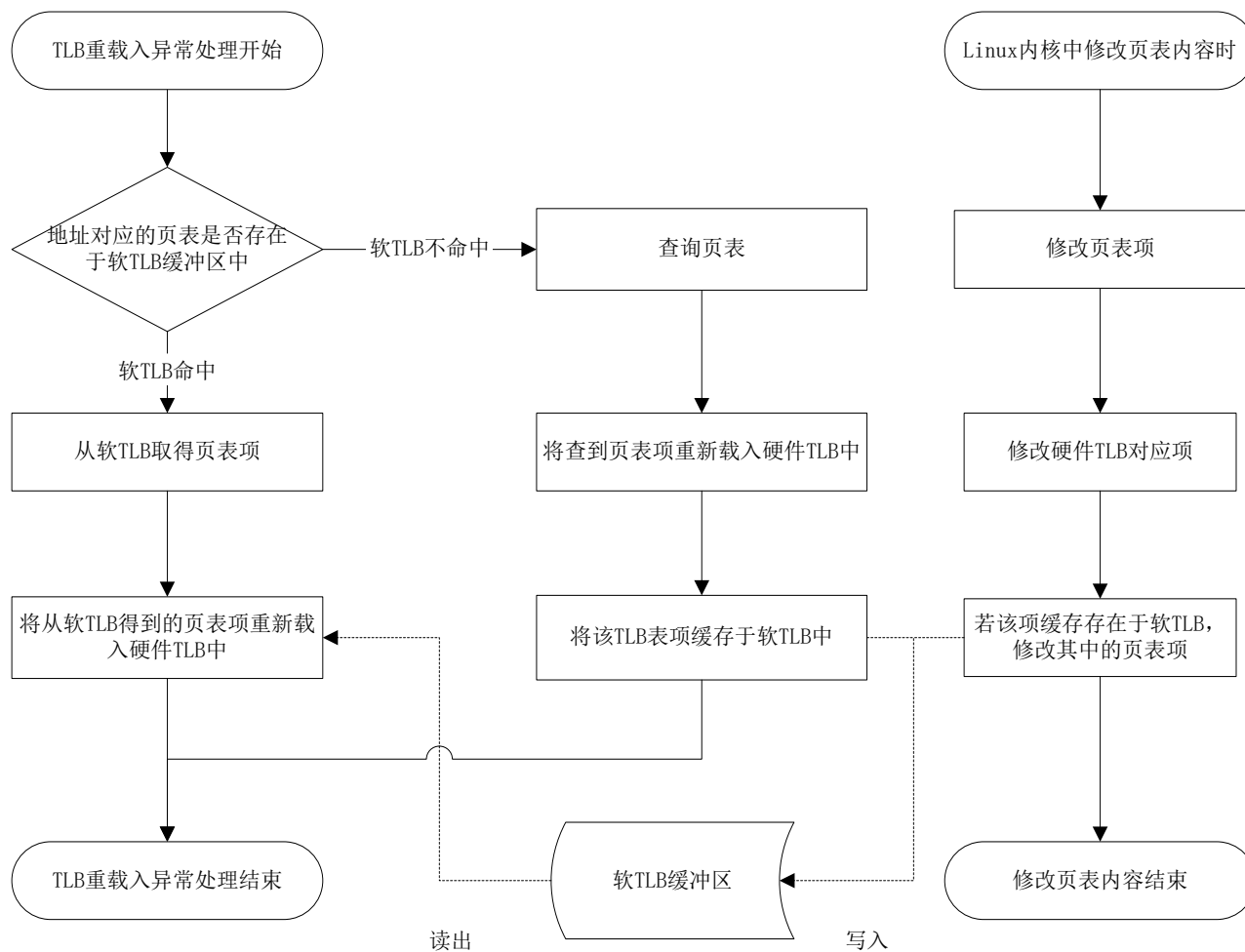
lddir k0, k0, 2 (二级页表)

ldpte k0, 0 三级页表/偶数页)
ldpte k0, 1 (三级页表/奇数页)
tlbwr
eret
```

软TLB的方案

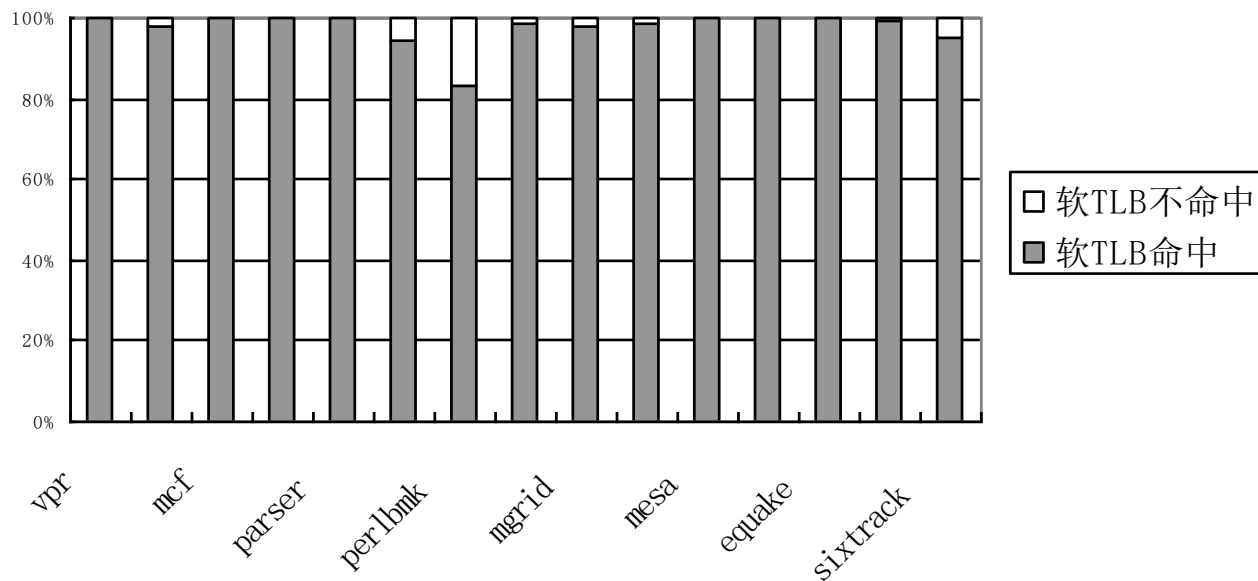
- 目的：主要减少TLB重载入异常处理的时间，提高TLB重载入异常处理的效率。
- 原理：通过减少TLB重载入异常处理过程Cache Miss的次数来减少TLB重载入异常处理的时间

软TLB的方案（续）

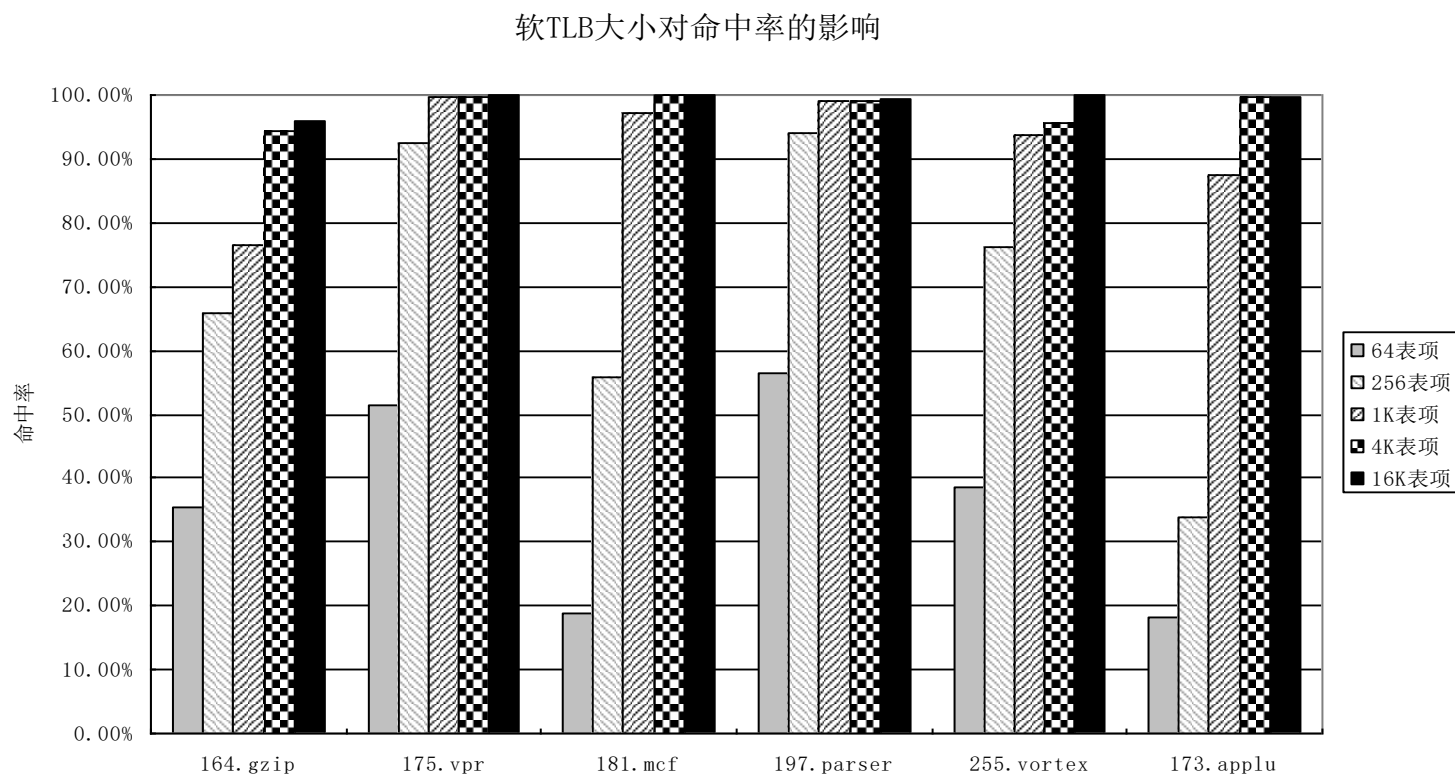


软TLB的命中率

软TLB在ref规模下的命中率（4K的软TLB表项，直接映射）

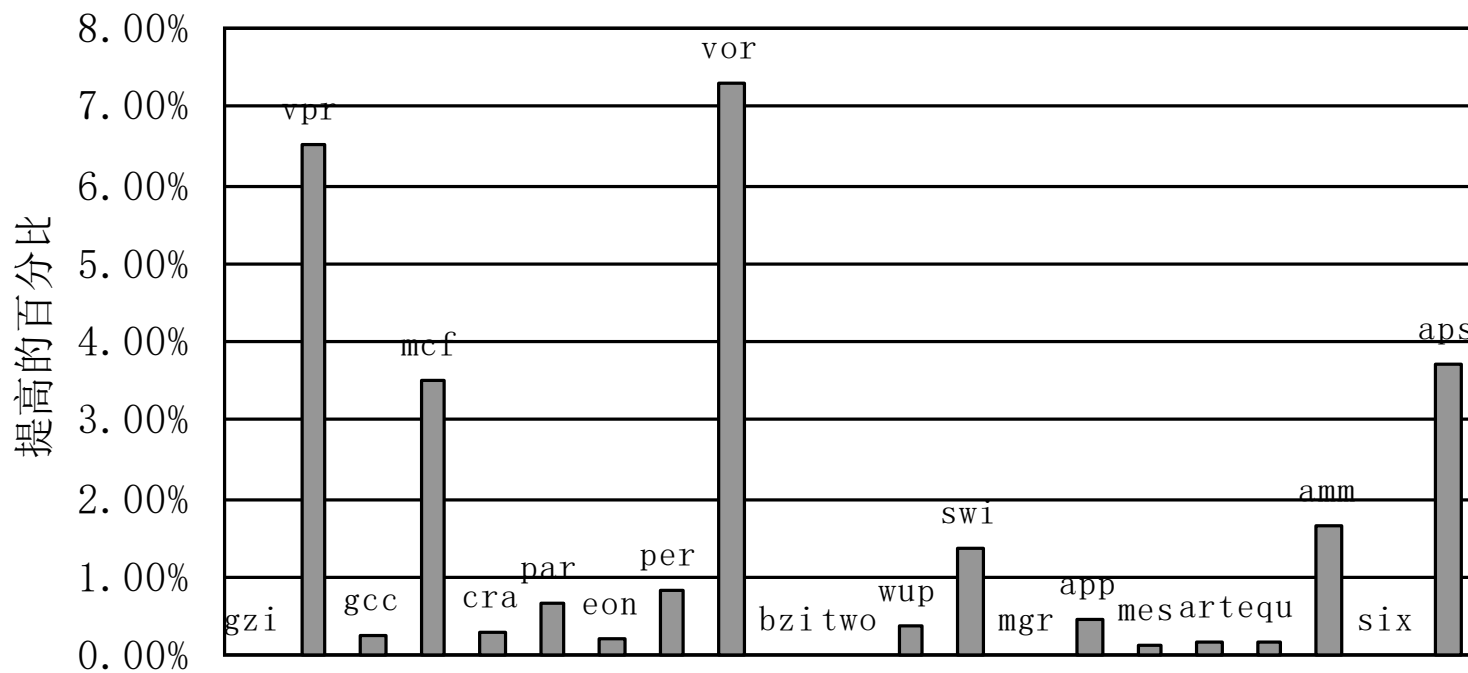


缓冲区大小对软TLB命中率的影响



软TLB对SPEC性能的提高

软TLB对SPEC2000在ref规模下分数的提高



常见处理器的结构参数

		Intel Ivybridge	AMD Bulldozer	IBM Power7	GS464E
前端	一级指令缓存	32KB, 8 路, 64B/行	64KB, 2路, 64B/行	32KB, 4 路, 128B/行	64KB, 4 路, 64B/行
	指令TLB	128 项, 4 路, L1 ITLB	72 项, 全相联, L1 ITLB 512 项, 4 路, L2 ITLB	64 项, 2 路, L1 ITLB	64 项, 全相联, L1 ITLB
	分支预取	BTB (8K-16K?项) 间接目标队列 (? 项) RAS (? 项) 循环检测	512 项, 4 路 L1 BTB 5120 项, 5 路 L2 BTB 512 项 间接目标队列 24 项 RAS 循环检测	8K 项本地 BHT 队列, 16K 项全局BHT 队列, 8K 项 全局 sel 队列 128 项间接目标队列 16 项 RAS	8K项本地 BHT 队列 8K项全局BHT 队列 8K项 全局 sel 队列 1K项间接目标队列 16项 RAS 循环检测
乱序执行	ROB	168 项	128 项	120 项	128 项
	发射队列	54-项 统一	60-项 浮点(共享) 40-项 定点, 访存	48-项 标准	32-项 浮点; 32-项 定点; 32-项 访存
	寄存器重命名	160 定点; 144 浮点	96 定点; 160 浮点(双核共享)	80 定点,浮点; 56 CR; 40 XER, 24 Link&Count	128 定点; 128 浮点/向量; 16 Acc; 32 DSPCtrl; 32 FCR
运算部件	执行单元	ALU/LEA/Shift/128位 MUL/128位 Shift/256位FMUL/256位 Blend + ALU/LEA/Shift/128位 ALU /128bit Shuffle/256位 FADD + ALU/Shift/Branch/ 128位 ALU/128bit Shuffle/256位 Shuffle/256位 Blend	ALU/IMUL/Branch + ALU/IDIV/Count + 128位FMAC/128位 IMAC + 128位FMAC/128位 XBAR + 128位 MMX + 128位 MMX/128位 FSTO	2 定点 + 2 浮点/向量 + 1 转移 + 1 CR	2 定点/转移/DSP + 2 浮点/向量
	向量宽度	256位	128位	128位	256位

常见处理器的结构参数

		Intel Ivybridge	AMD Bulldozer	IBM Power7	GS464E
访 存	访存单元个数	2 取+ 1 存	2 取/存	2 取/存	2 取/存
	Load/Store队列	64-项 Load 队列, 36-项 Store队列	40-项 Load 队列, 24-项 Store 队列	32-项 Load队列, 32-项 Store 队列	64-项 取/存 队列
	Load/Store 宽度	128 位	128 位	256 位 load, 128 位 Store	256 位
	TLB	100项全相联, L1 DTLB, 512项4路, L2 TLB	32项全相联, L1 DTLB, 1024项8路, L2 TLB	64项全相联, L1 DTLB, 512项4路, L2 TLB	32-项全相联L1DTLB, 每项两页 1024项8路L2 TLB, 每项两页
	L1D	32KB, 8 路, 64B/行	16KB, 4路, 64B/行	32KB, 8 路, 128B/行	每核64KB, 4 路, 64B/行
	L2	每核256KB, 8路, 64B/ 行	双核共享2MB, 16 路	每核256KB, 8路, 28B/ 行	每核256KB, 16路, 64B/行
	LLC	8个核20 MB	4个核8 MB	8个核32 MB	4个核4 MB~8MB
	L1失效队列	10	?	8	Unified 16
	L2失效队列	16	23	24	
	L1 Load-to-use	定点4时钟周期, 浮点/向量5时钟周期	4时钟周期	定点2时钟周期, 浮点/向量3 时钟周期	定点3时钟周期, 浮点/向量5 时钟周期
	L2 Load-to-use	12 时钟周期	18-20 时钟周期	8	22 时钟周期
	多层次硬件预取	有	有	有	有

作业