两级分配多可用重命名寄存器

杨 华 崔 刚 刘宏伟 杨孝宗
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 提出了一种新的寄存器重命名机制——两级分配多可用重命名寄存器,简称 2L-MuRR,其特点如下:(1)对重命名寄存器文件(RRF)的占用要经过两级重命名和分配过程,减少了无效占用时间.(2)RRF 被划分为多个不同宽度的字段,根据目标值的大小进行按字段分配,这样使每个重命名寄存器都是多可用的.(3)同一重命名寄存器内的多个目标值的写(或读)操作可以合并成一次完成,减少了 RRF 的访问次数和读/写端口的压力.模拟实验表明,2L-MuRR 显著提高了 RRF 的存储能力(70%以上),能以较少的重命名寄存器满足 SMT 高并行度的要求,有效缓解了寄存器文件的设计压力.

关键词 同时多线程;寄存器重命名;高性能;多可用
中图法分类号 TP302

Multi-Usable Rename Register with 2-Level Renaming and Allocating

YANG Hua CUI Gang LIU Hong-Wei YANG Xiao-Zong
(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

Abstract Great challenges must be overcome for designing the register file for SMT architecture. This paper proposes a novel register renaming scheme—Multi-usable Rename Register with 2-Level Renaming and Allocating (2L-MuRR), which features as follows. First, each renamed value undergoes 2-level of renaming and allocating before write-back to the Rename Register File (RRF), reducing useless occupancy. Second, the RRF is partitioned into several fields with different widths; either a single field or a field combination can be allocated to hold a renamed value, making each rename register multi-usable. Third, each access to a rename register may read or write multiple values in it, alleviating the pressure on the read/write ports. The simulations show that 2L-MuRR improves the RRF capacity greatly, over 70%. By using 2L-MuRR, SMT can achieve the same high performance with fewer rename registers, effectively reducing the design pressure on the register file.

Keywords SMT; register renaming; high-performance; multi-usable

1 引 言

指令级并行(Instruction-Level Parallelism,ILP)和线程级并行(Thread-Level Parallelism,TLP)是现代超标量微处理器提高性能的两种主要方式.具体来说,主要有增加发射宽度、加深流水线、支持多线程并行等三种途径.然而,这三种途径都对硬件支

收稿日期:2005-10-21;修改稿收到日期:2006-04-22.本课题得到国家"十五"预研基金(41316.1.2)、国家自然科学基金(60503015)和教育部博士点基金(20020213017)资助.
杨 华,男,1974 年生,博士研究生,研究方向为片上多线程体系结构、容错处理器体系结构.E-mail: yangh@ftcl.hit.edu.cn. **崔 刚**,男,1947 年生,教授,博士生导师,主要研究领域为容错计算、高性能计算机系统结构等. **刘宏伟**,男,1971 年生,博士,副教授,主要研究方向为可信计算、移动计算. **杨孝宗**,男,1939 年生,教授,博士生导师,主要研究领域为容错计算、可穿戴计算技术.

持提出了更高的要求，如更大的指令窗口(Instruction Window)、更多的运算单元、更大的寄存器文件(Register File，RF)，特别是用于保存指令中间结果的重命名寄存器文件(Rename Register File，RRF)等等.

RRF 用于保存每条指令从重命名段(Rename)到提交段(Commit)的中间结果，其最小数量为发射宽度×重命名到提交的流水线段数，才能保证流水线效率的充分发挥[1,2]. 因此，增加发射宽度和流水线深度会引起 RRF 的巨大增加，此外长延迟指令(如 Cache 缺失)会进一步增大对 RRF 的要求. 然而增大 RRF 会引起一系列的问题，如 RRF 的访问延迟、面积、功耗、读/写端口的压力都会显著增长. 现代处理器的时钟周期很大程度上取决于寄存器文件的访问延迟[3,4]. 大的寄存器文件的访问延迟和功耗已经成为未来处理器进一步提高性能的瓶颈之一[5,6].

1.1 寄存器文件的改进措施

单纯依靠增加重命名寄存器的数量来维持日益提高的处理器性能已变得愈加艰难[2,3]. 为此，研究人员提出了各种改进的寄存器文件组织和分配的措施，我们将其大致归纳为以下 3 类：

(1)通过分散式或层次化的组织来降低访问延迟. 主要有 register caching[4]、层次化组织[3]、multi-banked 划分[7]、半共享跳跃式流水寄存器[8]等. 这类措施实现起来比较复杂，并且不太适合于 SMT 中硬件资源多线程共享的宗旨.

(2)通过推迟分配或提早释放来缓解对 RRF 尺寸和端口数的高要求. 例如：利用程序语义分析[9,10]或检查点机制[11]来提早释放那些可能不再需要的 RR(Rename Register)；利用 VPR(Virtual-Physical Register)技术[5]推迟 RR 的分配等.

(3)基于寄存器目标值分布特性的优化. 基于目标值大小的优化，例如文献[2]中将“窄”的操作数直接存储于重命名表，减少 RRF 的占用；基于目标值时间局部性(temporal locality)的优化，例如文献[12]中将值相同的多个操作数存储于同一个 RR 中，文献[13]利用整数寄存器值的相似性(similarity)实现整数寄存器的压缩存储.

1.2 来自 SMT 的新挑战

SMT(Simultaneous Multithreading)是以动态超标量为基础、支持多线程并行的处理器体系结构[14]，自提出以来就倍受工业界和学术界的关注[10,15]. SMT 处理器中多个线程同时运行，共享并竞争各种硬件资源，动态地将 TLP 转化成 ILP，弥补了单一线程由于 ILP 不足而对硬件资源利用不充分的缺点. 然而多线程并行要求重命名寄存器成倍地增加，进一步加剧了寄存器文件的设计压力[6,10,15]. 实验表明，由于具备较高的 ILP，SMT 的性能对 RRF 的大小非常敏感. 如果 RRF 相对不足，会严重影响到包括 SMT 在内的各种片上多线程体系结构(CMT)潜在性能的发挥[6,10,16].

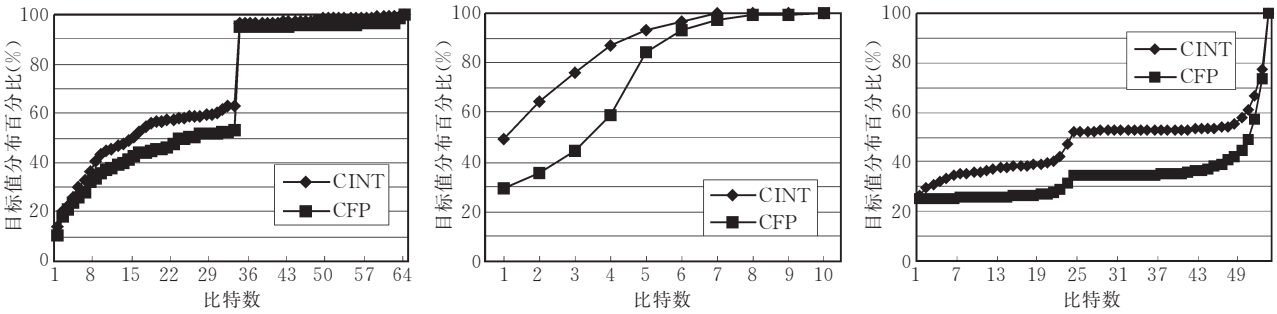
前面列举的三类措施，几乎都是从单线程角度提出的，虽能缓解寄存器文件的设计压力，但是面对多线程并行对 RRF 的巨大需求时，难免捉襟见肘[10,16]. 目前，对 SMT 的研究主要集中在多线程的硬件级调度、分支预测、存储层次、OS 级调度、面向应用的改进等方面，对寄存器文件设计压力则鲜有研究[6,10]. 文献[10]提出一种软件指导下的从不活跃上下文(idle hardware context)提前释放无效寄存器的方法，但该方法对软件不透明，需要扩充指令集并且修改编译器和操作系统. 文献[6]旨在消除多线程对 RRF 的占用冲突. 这两种措施利用了 SMT 中多线程的执行特性，来提高 RRF 的利用效率.

可以预见，在未来的片上多线程(十几甚至几十个线程)处理器中，多线程并行会引起重命名寄存器数以百计地增加[10,16]，太大的 RRF 会引起功耗和延迟激增、读/写端口压力增大等一系列问题[6,10,16,17]. 因此，需要结合多线程体系结构的特性，设计出更加有效的寄存器文件组织方式.

1.3 两级分配多可用重命名寄存器的提出

图 1 给出了我们在基于 Alpha AXP 架构的模拟器上运行 SPEC 2000 所得到的寄存器目标值的统计分布. Alpha AXP[18]是 64-bit RISC 架构，所有的寄存器都是 64bits，整数以 2 的补码表示，浮点数采用 IEEE-754 双精度标准. 图 1 中横坐标表示比特数，纵坐标表示该比特数所能表示的目标值的累计百分比. 以图 1(a)为例，8bits 能表示 CINT 中 41%和 CFP 中 34%的整数；10bits 能表示 CINT 中 44%和 CFP 中 37%的整数；而 34bits 则可以表示 CINT 和 CFP 中的几乎所有整数，分别达到 97%和 95%. 图 1(b)和图 1(c)分别表示浮点数的指数域和尾数域分布. 可见，大部分目标值可以用更少的比特数来表示和保存，即通过“掐头去尾”，截掉重复的符号位和浮点尾数中无效的 0，实现压缩保存.

据此，本文提出了一种新的 RRF 组织方式——两级分配多可用重命名寄存器(Multi-usable Rename Register with 2-Level Renaming and Allocating，2L-MuRR). 在 2L-MuRR 中，每个 RR 都是



(a) 整数的分布　(b) 浮点数指数域的分布　(c) 浮点数尾数域的分布

图 1　寄存器目标值的分布(对 SPEC 2000 的 CINT 和 CFP 程序组分别进行累计平均)

多可用的(multi-usable)，即一个 RR 内可同时存储多个寄存器值. 2L-MuRR 的提出基于以下 3 点：(1)大多数寄存器目标值可以用较少的比特数压缩表示和存储(见图 1)；(2)传统方式下指令在重命名段(Rename)分配 RR，而实际上 RR 的分配可以推迟到执行段(execution)完成后[5,19]，因为这时目标值才真正产生. 2L-MuRR 将 RR 的分配推迟到写回段(WriteBack)，减少 RRF 无效占用. (3)SMT 中多个线程共享 RRF，实验发现各线程的操作数分布有所差异，具备一定的“互补性”，这使得寄存器值按字段压缩更加有效(详见第 4 节). 例如，程序 galgel 的操作数较“宽”(10bits 能表示的操作数低于 18%)，而 vortex 的操作数略“窄”(10bits 能表示的操作数超过 42%)，二者互补，能有效提高 RRF 存储能力.

2L-MuRR 兼备上述第(2)类和第(3)类措施的特点，不但符合 SMT 资源共享的特点，而且结构简单，容易实现，对软件完全透明. 文献[2,12,13]也是利用寄存器目标值的分布特性进行“压缩”. 但是，文献[12]和[13]仅适用于整数寄存器，而且文献[12]要求寄存器值有“时间局部性”，文献[13]要求寄存器值有“相似性”，这些限定条件严重限制了其应用场合. 相比之下，2L-MuRR 可以同时应用于整数和浮点数寄存器，实现更加简单和灵活，不但没有上述“相似性”和“时间局部性”的限制，而且能利用多个线程间寄存器值的分布差异，更有效地压缩存储. 文献[2]仅适用于“非常窄”的寄存器值(小于 11bits)；相比之下，2L-MuRR 适用于各种宽度的寄存器值. 文献[5]中的 VPR 技术靠推迟分配来缓解 RRF 的压力. 与 VPR 相比，2L-MuRR 不但推迟了 RR 的分配，更重要的是利用这个“时间差”实现了寄存器值的压缩存储，极大提高了存储密度.

2　寄存器重命名技术回顾

寄存器重命名(register renaming)技术最早用在 IBM 360/91 的浮点单元中，限于当时的实现条件，直到 20 世纪 90 年代才逐渐被广泛采用，现已成为动态超标量处理器的一个标准特性[1]. 该技术将指令中的逻辑寄存器号(logical register)动态地映射到物理寄存器(physical register)，消除了指令序列中的伪相关——读后写(WAR)和写后写(WAW)，极大提高了 ILP. 相对于程序员可见的体系结构寄存器(AR，即逻辑寄存器)而言，用于重命名的物理寄存器称为重命名寄存器(RR).

2.1　寄存器映射表

寄存器映射表(Register Alias/Map Table，RAT)是核心结构，记录着逻辑寄存器到物理寄存器的最新映射关系. 通常，每条指令在重命名段(Rename)将其目标逻辑寄存器映射到一个物理寄存器，即分配一个物理寄存器，并将这种映射关系记录在 RAT 中. 后继指令如果在逻辑上依赖该目标寄存器作为输入，则通过查询 RAT 即可得到其最新对应的物理寄存器，从而既保持了指令序列的依赖关系，又消除了操作数伪相关，实现了指令的动态超标量发射.

典型的 RAT 有 RAM(Random-Addressed-Memory)和 CAM(Content-Addressed-Memory)两种结构[1]，如图 2 所示. RAM 结构的条目数与 AR 数量相等，用逻辑寄存器号进行随机访问，输出其对应的物理寄存器号. CAM 结构的条目数与物理寄存器的数量相等，寄存器目标值就直接存在这个表中；对逻辑寄存器号进行关联查找(associative lookup)，可直接输出其最新对应的目标值. CAM 中一个逻辑寄存器号可能对应多个物理寄存器(由多次重命名引起)，靠每个条目中的“V”标示其是否最新映射. 图中的阴影部分表示复制的映射记录，用于对程序的冒险执行和精确中断提供支持. RAM 比 CAM 扩展性好，更适合多线程环境对大规模 RRF 要求.

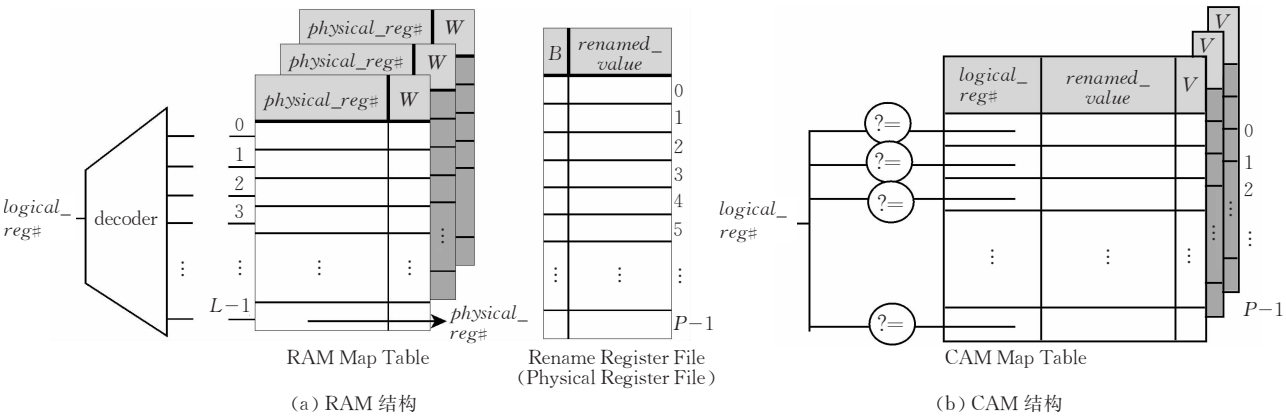


图 2 寄存器映射表的两种结构

2.2 重命名寄存器的存储位置

RR 和 AR 最终都要保存在物理寄存器文件中，为了清晰起见，对应的存储区域分别称为 RRF 和 ARF. 根据二者的位置关系，可分为三种存储结构[1]，图 3 给出了三种结构示意、一些代表性处理器及其推出年代.

(1) 混合式(merged file). 没有独立的 RRF，RRF 与 ARF 共用一个物理寄存器文件，如图 3(a). 该结构的优点是指令提交时省去了目标值从 RRF 到 ARF 的拷贝.

(2) 分立式(stand alone). RRF 与 ARF 分立于不同的物理寄存器文件，如图 3(b). 该结构下 RRF 与 ARF 分工清晰，但指令提交时需要目标值从 RRF 到 ARF 的拷贝.

(3) ROB 嵌入式. 没有独立 RRF，将 RRF 的功能“嵌入”到 ROB(Reorder Buffer)中，如图 3(c). ROB 原本用于记录指令状态和相对顺序，每条指令对应一个单元；通过扩展 ROB 来存储指令的目标值是很自然的事，但增加了 ROB 的设计复杂度，指令提交时也需要将目标值从 ROB 拷贝到 ARF.

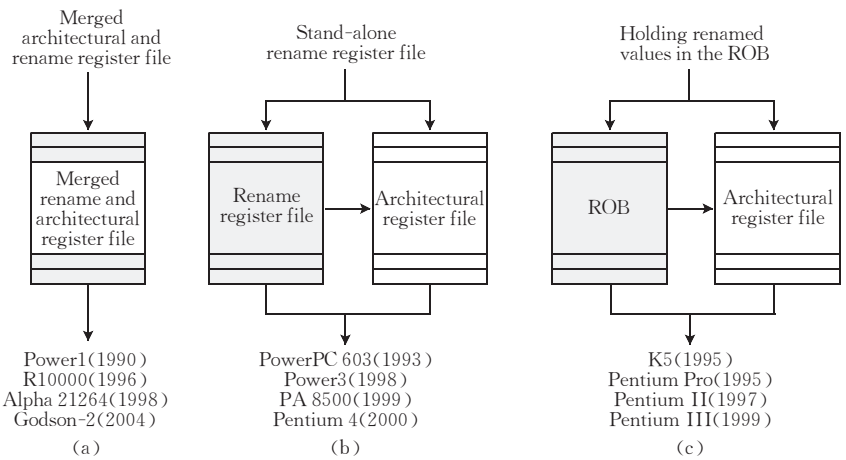


图 3 RRF 的三种存储位置

3 2L-MuRR 体系结构

名如其义，2L-MuRR 采用两级的寄存器重命名和分配，它利用图 1 所示的寄存器目标值的分布特性对其进行压缩存储，来提高 RRF 的存储效率. 为了实现方便和扩展性好，2L-MuRR 采用 RAM 映射表，并采用 RRF 与 ARF 分立式的结构. 本节将全面介绍 2L-MuRR 的基本结构、工作原理、流水线的变动、死锁的解决以及实现代价等方面.

3.1 基本结构

2L-MuRR 由 RAM 结构的寄存器重命名过程(见图 2(a))改进而来，其基本逻辑结构如图 4 所示. 每个 RR 被划分成 N 个不同宽度的字段：$field_1$，$field_2$，…，$field_N$；每个单独的字段或同一 RR 内多个字段的组合都可以存储一个目标值，根据目标值的大小予以分配，实现了 RR 的多可用. 另外，每个 RR 都有一个 N bits 的标签——$busy_flag$，用于指示各个字段是否被占用，每个字段对应 1 bit.

为了实现两级映射和 RR 推迟分配，在 RAM

映射表和 RRF 之间加入了一个称为 V2P Map 的映射表(Virtual-to-Physical Map)，表中的单元称为虚拟寄存器(Virtual Register，VR). 每个 VR 含以下 4 个字段：

(1) B——busy bit，指示该单元是否已被占用；

(2) *rename_reg*#——指示该单元对应的目标值占用哪个 RR，当 W 为 1 才有意义；

(3) *field_mask*——宽度为 N bits，指示该单元对应的目标值占用 RR 的哪几个字段，与 *rename_reg*# 联合对 RRF 进行寻址，当 W 为 1 才有意义；

(4) W——write-back bit，指示该单元对应的目标值是否已经写回到 RRF.

如图 4 所示，引入 V2P Map 后，RAM 映射表保存的是各逻辑寄存器对应的虚拟寄存器号(*virtual_reg*#)，而不再是物理寄存器号(*physical_reg*#). 寄存器重命名过程分两级进行(参见图 6(b))：第一级是在重命名段(Rename)进行，将逻辑寄存器重命名为 V2P Map 中的 VR；第二级在写回段(Write-Back)进行，此时目标值已在执行段(Execution)得出并放在段间缓冲里，可根据其大小在 RRF 中分配适当的字段予以写回，并将分配的 RR 号和字段号记录在相应的 VR 中，供后继指令查询使用. 可见，V2P Map 在整个的寄存器重命名过程中起“中继”的作用，其本身并不保存目标值. V2P Map 逻辑简单，可以做得较大，使更多的指令顺利通过重命名段继续执行.

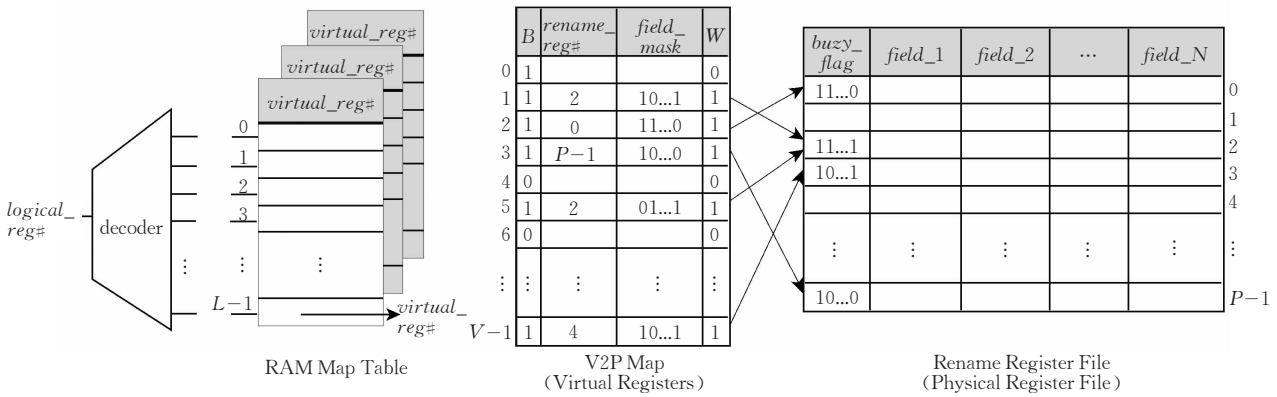


图 4　2L-MuRR 的基本逻辑结构

3.2　存储字段的分配和回收

2L-MuRR 有两个特点：(1)对 RR 的分配推迟在写回段(WriteBack)进行，减少 RRF 的无效占用时间；(2)每个 RR 不再作为一个整体进行分配与回收，而是针对其各个字段的分配与回收，各字段的分配状态记录在 *busy_flag* 中，多个目标值可同时存储于一个 RR 的不同字段中，借此提高 RRF 的存储能力. 例如图 4 中的 1# VR 和 5# VR 的目标值同时存储于 2# RR 的不同字段中. 鉴于整数和浮点数不同的分布特点，宜采用不同的 RR 字段分配方法. 下面结合 64 位整数和 IEEE-754 双精度浮点数(64 位)的压缩表示，介绍字段分配方法和原则如下：

(1) 对于整数，截去其重复的符号位后，将其存入一个或多个字段中.

(2) 对于浮点数，其指数域和尾数域分布特点不同，分别进行处理. 在 IEEE-754 双精度表示中，指数域有 1023 的偏移量. 考虑到偏移量对指数域分布的影响，将其最高位(bit-62)作为指数域的符号位(称为 ES)加以保留，其余 10 位当作整数来压缩，存入一个字段. 对于尾数域，将其低位重复的 0 截去后，存入一个或多个字段. ES 位和符号位 S(bit-63)分别记录于对应的指数字段和尾数字段的最高位上. 可见，为了指数域和尾数域的分别处理方便，一个浮点数至少要占用两个字段.

(3) 一个目标值，无论是整数或浮点数，其占用的所有字段必须在同一个 RR 内，但这些字段可以不连续，靠 VR 中的 *field_mask* 指示. 为了实现和区分方便，规定浮点数的指数字段位于其尾数字段的左侧(参见图 5).

(4) 将目标值存入字段时，如果不能正好填满某个字段，则以位扩展补齐，称为“填充位”.

(5) 分配字段时采用尽量节省字段数的原则. 以整数为例，尽量将其存储在一个足够大的字段中；如果没有足够大的字段，才用多个字段的组合来存储该值.

图 5 给出了一个目标值压缩存储的例子：整数 $L1$ 和浮点数 $D1$ 压缩后写回到同一个 RR 中①，$L1$

① 整数和浮点数操作既可以采用分立的 RRF，也可以共用一个 RRF. 2L-MuRR 对这两种情况都适用. 为了简化，本文采用整数和浮点数共用的 RRF，统一划分字段.

存在 *field*_4 中；*D*1 的指数域存在 *field*_2 中，尾数域存在 *field*_3 和 *field*_5 中；*field*_1 空闲. 图中的黑体数字表示该值的“关键信息”，可用作该值的压缩表示；斜体数字表示“填充位”.

指令提交或从误预测分支恢复时，要回收其占用的 RR 和 VR. VR 的回收只需将 *B* bits 置 0 即可. RR 字段的回收只需简单的逻辑操作：*RR.busy_flag*<=*VR.field_mask* xor *RR.busy_flag*，其中 *RR* 和 *VR* 分别表示该指令对应的重命名寄存器和虚拟寄存器.

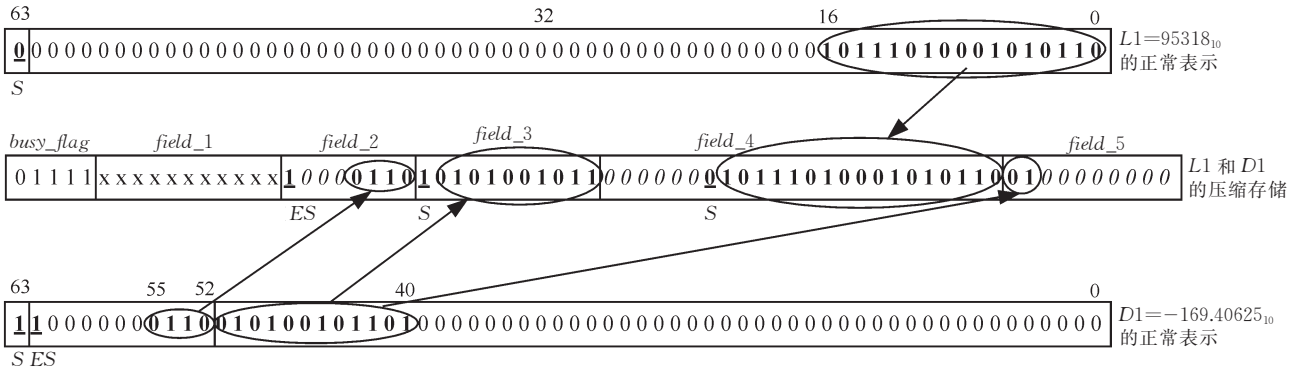


图 5 目标值的压缩存储举例

3.3 存储字段的划分

2L-MuRR 中 RR 字段如何划分，即字段个数以及每个字段的宽度，会直接影响其整体效能的发挥. 如果字段数较多，各个字段的长度则会相对较小，这样会因为每个目标值“填充位”减少而提高 RR 利用率. 但太多的字段数会增加实现的复杂度，如更多的字段组合方式增加查找代价，*busy_flag* 和 *field_mask* 增宽等. 反之，字段数较少则难以发挥 RR“多可用”的优势.

为了在字段划分和实现代价间找到一个平衡，需再次利用目标值的分布特点. 图 1 中各曲线在经过一些点后变得平缓，如图 1(a)中的 8bits，10bits，34bits 点，图 1(b)中的 8bits 点，图 1(c)中的 24bits 点等. 参照这些“关键点”，并结合 3.2 节的分配方法，我们采用的划分是 11_8_11_24_10，即每个 RR 分为 5 个字段（*field*_1～*field*_5），宽度分别是 11bits，8bits，11bits，24bits，10bits；相应地，*busy_flag* 和 *field_mask* 宽度为 5bits. 各字段总的宽度仍是 64bits（不包括 *busy_flag*），在模拟实验中公平比较.

3.4 流水线的变动

引入 2L-MuRR 的流水线如图 6(b)所示，仍分为取指（Fetch）、译码（Decode）、重命名（Rename）、入队列（Queue）、发射（Issue）、执行（Execution）、写回（WriteBack）和提交（Commit）段. 它与传统流水线（图 6(a)）的区别在于：(1)寄存器重命名和分配过程分布在 Rename 段和 WriteBack 段两级进行，Rename 段将目标寄存器重命名为 VR，而 WriteBack 段分配 RR 并写回指令结果. (2)RR 的数量一般远少于 VR，因此在 WriteBack 段可能没有合适的 RR 供分配和写回，这种情况称为“写回失败”. 写回失败时，该指令重入指令队列，称为 re-enqueue，等待稍后重新发射和执行.

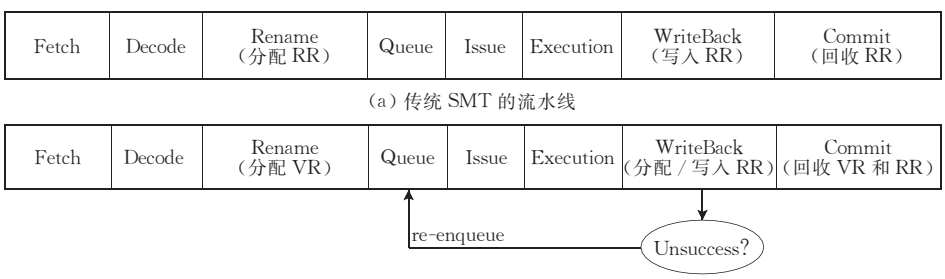


图 6 流水线的变动

3.5 死锁的预防

重入队列的指令，经稍后重新发射和执行，再次到达 WriteBack 段时很可能写回成功，因为这期间其它指令提交后释放了 RR 字段. 但动态超标量处

理器中指令是乱序执行、按序提交的，所以可能会发生“死锁”. 即写回失败的指令 Ia 在逻辑顺序上“早”于写回成功的指令 Ib，Ic，…，但由于是按序，这些指令不能先于 Ia 提交，因而不能释放 RR，而 Ia 因为没有新的 RR 释放而陷入 re-enqueue、Issue 和 Execution 循环过程. 为了避免死锁，我们采用了 VPR 中的 NRR 方法[5]，即为每个线程保留 *NRR* 个寄存器，专门供最早的 *NRR* 条指令写回. 该方法虽然简单，却能完全避免死锁的发生.

3.6　实现代价

从图 4 和图 6 可见，2L-MuRR 增加的硬件代价包括 V2P Map 和寄存器值的压缩/解压缩逻辑这两个方面，下面分别予以分析：

(1) 虚拟寄存器，即 V2P Map：V2P Map 采用随机寻址方式，每个单元也比较窄（*rename_reg*＃的宽度是 RRF 尺寸的对数，*field_mask* 是 5bits），并不保存真正的寄存器值；其实现方式和 VPR 机制中的 GMT Table 类似，其硬件复杂度的增加不大[5,19]. 如前所述，2L-MuRR 把寄存器重命名过程“解耦合”在 V2P Map 和 RRF 两部分配合完成，减少对 RRF 的尺寸依赖. V2P Map 可以做得较大（模拟器中设为指令窗口大小），使更多的指令通过 Rename 段继续执行，消除 RRF 相对较小引起的重命名阻塞.

(2) 寄存器值的压缩/解压缩过程：2L-MuRR 对目标值的压缩只是简单的“掐头去尾”，即截去重复的符号位和多余的 0. 该操作无需计算，一些简单的组合逻辑（相等检测器）即可实现，延迟在 3 级逻辑门左右[20]，而图 6 中 8 级流水线的段延迟在 20 级逻辑门以上[21]，因此 WriteBack 段足以完成“压缩”和写回，这与文献[2]的实现类似，不影响流水线深度和时钟周期. 每个周期，位于 Execution 和 WriteBack 段间缓冲（stage buffer）内的目标值流过该组合逻辑后实现“压缩”，写入 RRF. 为了适应“宽窄”各异的操作数，“压缩”过程是多路并行的，即同时尝试多种不同的截取方式（具体依赖于 RRF 字段划分），从中选择最“窄”的. 因此所有操作数的处理延迟一样，与“宽窄”无关. 同理，“解压缩”过程也很简单，将读出的字段进行符号位扩展和尾数补 0 即可.

此外，还需要在 RRF 的读/写路径上加入若干个多路器（MUX）和字段查找逻辑，以实现 RRF 的按字段读/写. 为了防死锁，还需为每个线程配置 *NRR* 个寄存器. 实验表明，*NRR*＝1 即可完全消除死锁，并且不影响整体性能.

综上，2L-MuRR 对软件透明，硬件代价也很小. 与以往的寄存器压缩措施[2,12,13]相比（参见 1.1 节和 1.3 节），2L-MuRR 具备明显的实现优势：适用于整数和浮点数；适用于各种宽度的值；无需“相似性”和“时间局部性”检测. 与 VPR 相比，2L-MuRR 利用操作数的分布特性实现 RR 的“多可用”，极大提高了 RRF 存储能力，并且 SMT 多线程的“互补性”有助于“多可用”能力的发挥. 下面模拟实验比较了传统寄存器重命名、VPR 和 2L-MuRR.

4　模拟实验环境及结果

SimpleScalar[22]是一套执行驱动（execution-driven）的动态超标量处理器模拟工具，广泛用于新架构思想的模拟和评测中. 我们对 SimpleScalar/Alpha AXP ver3.0d 进行扩充和修改，分别构建了支持传统寄存器重命名、VPR 和 2L-MuRR 的 SMT 模拟器. 主要的扩充和修改如下：

(1) 复制线程上下文，支持多线程体系结构；以 Tullsen 描述[14]为蓝本构建了 SMT 模拟器，实现包括 ICOUNT 取指、OLDEST_ISSUE 发射、Cache 和流水线资源竞争和共享等 SMT 体系结构的基本特性. 该模拟器比较完备，已用于我们对多线程体系结构的研究当中.

(2) 原 SimpleScalar 中的寄存器重命名过程比较隐秘（宏实现），且采用图 3(c) 的实现方式，寄存器目标值隐含在 RUU 单元（兼备 RS（保留站）和 ROB 的功能）中，直到指令提交. 我们采用 RRF 和 ARF 分立的方式（图 3(b)），在基本 SMT 模拟器的基础上，通过修改以 *ruu_dispatch*、*ruu_writeback* 和 *ruu_commit* 为代表的主要函数，构建了一个明确支持传统寄存器重命名机制的 SMT 模拟器，简称 Trad.

(3) 扩展 Trad，添加 V2P Map 结构，实现两级重命名过程和 RRF 按字段存取，构建了支持 2L-MuRR 的 SMT 模拟器，简称 2L-MuRR.

(4) 扩展 Trad，增加 GMT Table[5]，将 RRF 的分配推迟到 WriteBack 段完成，构建了支持 VPR 机制的 SMT 模拟器，简称 VPR.

模拟器的主要参数设置如表 1 所示，适用于 4～8 线程. 测试程序来自 SPEC 2000，采用 Ref 输入. 表 2 列出了 4-线程测试组合：1＃～4＃每个都由不同线程（两个 CINT 和两个 CFP）组成，代表一般的多

线程;5#、6#和7#分别由操作数"较窄"的 vortex、"较宽"的 galgel、"适中"的 lucas 组成,代表相同线程并发;8#由 vortex 和 galgel 组成,用以说明寄存器值的"宽窄互补".每次运行时先快速略过(没有性能统计)各线程的前 1000M 条指令,然后开始性能模拟,直到运行最快的线程完成 100M 条指令为止;在 40 至 100 之间以步长 20 调整 RRF 的大小,进行了一系列实验;为了对比相对性能,还模拟了 RRF 无限大(在 Trad 中设 RR 数为 256,即指令窗口大小)时的运行,可视为具备性能峰值.

表 1 SMT 模拟器的主要硬件参数

	Trad	VPR	2L-MuRR
死锁的预防、GMT 或 VR 数量	—	$NRR=1$, 4-线程时 $GMT=256$, 8-线程时 $GMT=512$	$NRR=1$, 4-线程时 $VR=256$, 8-线程时 $VR=512$
L1 I-Cache	128KB, 64-Byte block, 4-way 相关, 2-cycle 延迟, LRU 替换		
L1 D-Cache	128KB, 64-Byte block, 4-way 相关, 2-cycle 延迟, LRU 替换		
L2 Cache(Unified)	1MB, 64-Byte block, 4-way 相关, 10-cycle 延迟, LRU 替换		
branch predictor	gshare(2K-entry), 32-entry RAS, 4-way 512-set BTB		
main memory	infinite capacity, 100-cycle 延迟		
Fetch/Decode/Issue/Commit 宽度	12 inst/cycle, ICOUNT 取指, OLDEST_ISSUE 发射		
out-of-order 执行单元	8 Int ALU, 4 Int Mul/Div, 4 FP ALU, 2 FP Mul/Div, 4 Read-port, 2 Write-port, 64-entry ROB/thread, 32-entry LSQ/thread		

表 2 Trad、VPR 和 2L-MuRR 的性能(IPC)比较

RR 数量		不同测试组合情况下的 IPC								平均值
		1#	2#	3#	4#	5#	6#	7#	8#	
		crafty twolf wupwise equake	gzip vpr mesa art	parser vortex swim ammp	eon perlbmk galgel lucas	vortex vortex vortex vortex	galgel galgel galgel galgel	lucas lucas lucas lucas	vortex vortex galgel galgel	
40	Trad	2.72	1.62	0.32	2.32	2.50	1.44	3.07	1.66	1.96
	VPR	4.48	2.98	2.35	4.08	2.97	2.41	3.71	3.10	3.26
	2L-MuRR	5.70	4.84	2.76	5.14	3.23	2.84	4.53	4.08	4.14
60	Trad	3.67	2.31	1.02	3.58	2.98	1.86	3.28	2.24	2.62
	VPR	5.26	3.77	2.56	4.66	3.16	2.65	4.17	3.38	3.70
	2L-MuRR	6.02	5.53	2.97	5.80	3.24	3.03	5.30	4.47	4.55
80	Trad	4.54	2.88	1.50	4.17	3.13	2.05	3.83	2.72	3.10
	VPR	5.80	4.55	2.93	5.16	3.22	2.75	4.39	3.83	4.08
	2L-MuRR	6.20	5.60	3.32	6.01	3.24	3.15	5.53	4.50	4.70
100	Trad	5.08	3.74	1.67	4.95	3.21	2.34	4.23	3.47	3.59
	VPR	6.06	5.19	3.21	5.60	3.23	2.86	4.58	4.15	4.36
	2L-MuRR	6.22	5.60	3.37	6.03	3.25	3.20	5.59	4.50	4.72
Peak(RR 数=256)		6.22	5.60	3.37	6.03	3.25	3.20	5.63	4.50	4.73

4.1 性 能

IPC(Instruction-Per-Cycle)是性能评价的重要指标,表 2 给出了 RR 数量为 40～100 时三种机制的 IPC 对比.显然,2L-MuRR 的性能远远超过了 Trad 和 VPR,特别是当 RR 数量较小的时候.例如:RR 数为 60 时,三者的平均 IPC 分别是 2.62,3.70 和 4.55,2L-MuRR 比 Trad 高出 74%,比 VPR 高出 23%.表 2 中最后一行是 Trad 在 RR 数量不限时所获得的 IPC 峰值(Peak).表中可见,当 RR 数量逐渐增加时:Trad 的性能提升迅猛,反映出传统方式下 SMT 性能对 RR 数量的强烈依赖;VPR 的性能也有明显提升,但离峰值仍有些差距,这说明 VPR 虽能减缓 RR 分配压力,但面对多线程的巨大需求仍稍显不足;而 2L-MuRR 的性能虽然增加缓慢,但始终与峰值差距不大,这说明 2L-MuRR 对 RR 的存储能力显著提高,能以较少的 RR 数量满足 SMT 高并行度的要求.例如:对于各测试组合,RR 数量为 40 时 2L-MuRR 的性能均高于 RR 数量为 100 的 Trad,且多数高于 RR 数量为 80 的 VPR;2L-MuRR 在 RR 数量为 60 时平均性能(4.55)达到峰值(4.73)的 96%,在 80 时基本达到峰值,而 Trad 和 VPR 分别需要 RR 数量超过 160 和 130 才能达到峰值(表中未列出).

当RR数量较少时,3#的 Trad 性能与 2L-MuRR 和 VPR 相比差距悬殊,这是由于传统方式下 ammp 大量的 D-Cache 缺失导致其长时间"独占"RRF,妨

碍其它线程的重命名和执行，而 2L-MuRR 和 VPR 通过推迟分配有效消除了对 RRF 的无效占用.

下面针对各测试组合说明 2L-MuRR 的性能优势：(1)对于所有测试组合，2L-MuRR 的性能都明显高于 Trad 和 VPR.(2)对于 1#～4#、7#、8#，2L-MuRR 的性能优势最为显著，因为这些组合中操作数的“宽窄补互”较好，更能发挥 RRF 按字段分配的优点；其中 1#～4#由不同线程组成，7#虽是相同线程并发，但 lucas 本身互补性较好(操作数“适中”).(3)组成 5#的 vortex 虽然“较窄”，但其 RR 需求量本来就不大(60 左右)，因此 RR 数量较少时 2L-MuRR 的优势才明显.(4)组成 6#的 galgel “较宽”，2L-MuRR 的提升幅度略低，但仍在 10%以上.(5)8#由 vortex 和 galgel 组成，其性能明显高于 5#和 6#，可见“互补性”有助于 2L-MuRR 效能的进一步发挥.

综上，2L-MuRR 能以较少的 RR 数量满足 SMT 高并行度的要求，且适用于各种线程组合(包括相同线程并发)；多线程操作数的“互补性”有助于其效能的更好发挥.

4.2　RR 的存储能力

图 7 给出了三种机制的 RR 占用率(RR_busy_rate)对比，图中各点都表示所有测试组合的平均值.这里规定，对于 2L-MuRR，RR 中任何字段被占用都视为该 RR 被占用.即便如此，2L-MuRR 的 RR 占用率仍远低于 Trad 和 VPR，这是因为：(1)两级重命名推迟了 RR 占用；(2)“多可用”提高了存储密度.随着 RR 数量的增加，Trad 的 RR 占用率下降非常缓慢，反映出 SMT 的性能增长对 RR 数量的巨大依赖.实验表明，当 RR 数超过 150 以后，Trad 的 RR 占用率才开始显著下降(图 7 中未画出).

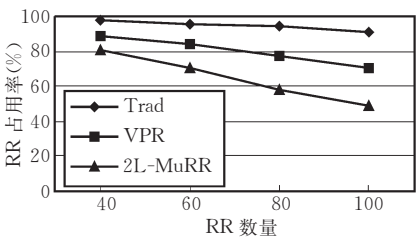


图 7　4-线程时 RR 占用率对比

图 8 直观地给出 2L-MuRR 对 RR 存储能力的提高，其中 Inst_per_RR 代表平均每周期每个 RR 中保存的目标值的个数，RR_busy_rate 含义与图 7 相同，二者的商(Inst_per_RR/RR_busy_rate)可视为 RR 的存储能力.在 RR 数量为 40～120 时，该值保持在 1.7～2.0 左右，这可以理解为 2L-MuRR 将每个 RR 存储能力提升到原来的 1.7～2.0 倍.

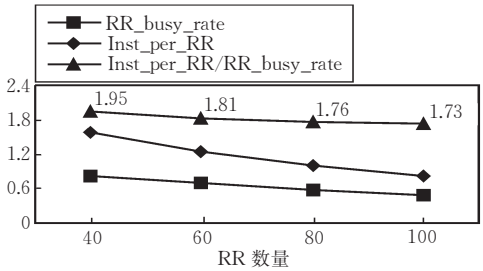


图 8　2L-MuRR 中 RR 存储能力

4.3　字段划分的均衡性

图 9 给出了 RR 数为 60 时 2L-MuRR 各字段占用次数的相对比例，其它 RR 数的运行结果与此类似.可见，除 *field*_5 外各字段利用的比较均衡，这是因为字段分配时多数“窄”目标值(10bits 以下)都被前 3 个字段(*field*_1～*field*_3)所“截获”的原因.

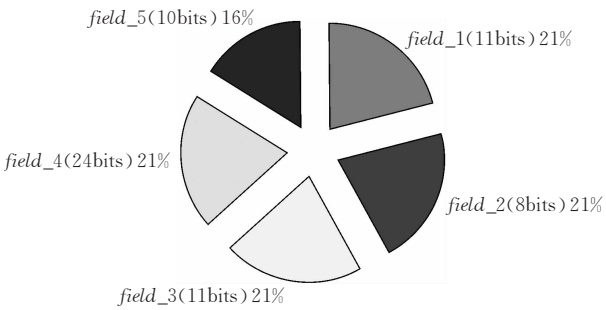


图 9　各字段占用的相对比例

4.4　RRF 写入次数的减少

在 2L-MuRR 中，对 RR 的一次读/写可以访问其中多个目标值.因此 2L-MuRR 能减少对 RRF 的访问次数，减轻 RRF 读/写端口的压力.这里我们只给出对 RRF 写入次数减少的统计：在同一周期写回的多条指令，如果分配到同一个 RR 中的不同字段，则可将其写入过程合并为一次完成.实验表明，对 RRF 的写入次数可平均减少 8%左右(最高 12.7%，最低 3.9%).

4.5　更多线程时的表现

当线程级并行度增加时，2L-MuRR 的优势将更加明显.为了验证这一观点，我们构造两个 8-线程测试组合进行了实验，分别是 crafty-twolf-wupwise-equake-parser-vortex-swim-ammp 和 gzip-vpr-mesa-art-eon-perlbmk-galgel-lucas.图 10 和图 11 分别给出了当 RR 数量为 80～200 时平均 IPC 和 RR 占用率对比.可以看出，RR 数量为 120 时 2L-MuRR 已基本达到 IPC 峰值，明显高于 Trad

和 VPR,其 RR 占用率却远低于 Trad 和 VPR.

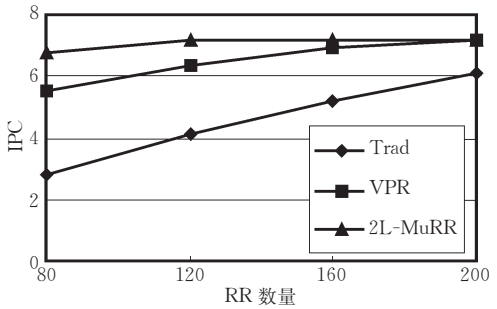


图 10 8-线程时 IPC 对比

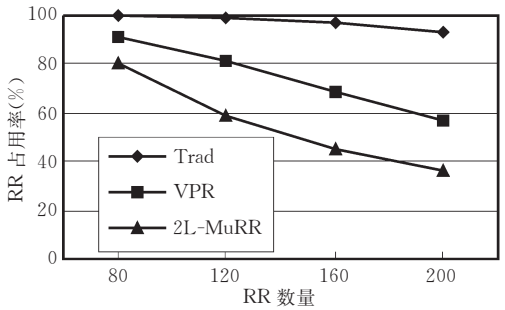


图 11 8-线程时 RR 占用率对比

5 结 论

2L-MuRR 利用目标值分布特性实现了对 RR 的字段划分和复用,并通过两级重命名过程推迟了对 RR 的分配,使每个 RR 的利用效率大为提高.相对于传统的寄存器重命名方式,2L-MuRR 能以较少(一半甚至更少)的 RR 满足 SMT 中多线程并行对 RR 数量的巨大要求.例如:4-线程 SMT,RR 数为 60 时 2L-MuRR 的性能已达到峰值的 96%,超过 RR 数为 100 的 Trad 和 RR 数为 80 的 VPR.此外,2L-MuRR 能明显地减少 RRF 的访问次数(8% 写访问),缓解读/写端口的压力.2L-MuRR 对于软件是透明的,所增加的硬件代价是可以接受的.

总之,2L-MuRR 能有效缓解 SMT 体系结构随并行度提高而对寄存器文件的巨大压力,适合于未来处理器多线程、宽发射、深流水的发展方向.可以预见,在未来线程数达到十几甚至几十的片上多线程处理器[16]上物理寄存器以百计甚至千计的规模增长,2L-MuRR 将发挥更大的性能优势.

参 考 文 献

1 Sima D.. The design space of register renaming techniques. IEEE Micro, 2000, 20(5): 70～83

2 Lipasti M. H., Mestan B. R., Gunadi E.. Physical register inlining. In: Proceedings of the 31st International Symposium on Computer Architecture, Munich, 2004, 325～335

3 Balasubramonian R., Dwarkadas S., Albonesi D. H.. Reducing the complexity of the register file in dynamic superscalar processors. In: Proceedings of the 34th IEEE/ACM International Symposium on Microarchitecture, Austin, 2001, 237～248

4 Postiff M., Greene D., Raasch S. *et al*. Integrating superscalar processor components to implement register caching. In: Proceedings of the International Conference on Supercomputing, Serento, 2001, 348～357

5 González A., González J., Valero M.. Virtual-physical registers. In: Proceedings of the 4th International Symposium on High-Performance Computer Architecture, Los Alamitos, 1998, 175～184

6 Yang H., Cui G., Yang X. Z.. Eliminating inter-thread interference in register file for SMT processors. In: Proceedings of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies. Dalian, 2005, 40～45

7 Cruz J. L., González A., Valero M., Topham N. P.. Multiple-banked register file architectures. In: Proceedings of the 27th International Symposium on Computer Architecture, Vancouver, 2000, 316～325

8 Rong Hong-Bo, Tang Zhi-Zhong. A novel register file structure supporting for software pipelining of nested loops. Journal of Software, 2000, 11(3): 401～409(in Chinese)
(容红波,汤志忠.一种支持多重循环软件流水的寄存器结构.软件学报, 2000, 11(3): 401～409)

9 Martin M. M., Roth A., Fischer C. N.. Exploiting dead value information. In: Proceedings of the 30th IEEE/ACM International Symposium on Microarchitecture, Triangle Park, 1997, 125～135

10 Lo J. L., Parekh S. S., Eggers S. J. *et al*. Software-directed Register deallocation for simultaneous multithreaded processors. IEEE Transactions on Parallel and Distributed Systems, 1999, 10(9): 922～933

11 Martinez J. F., Renau J., Huang M. C. *et al*. Cherry: Checkpointed early resource recycling in out-of-order microprocessors. In: Proceedings of the 35th IEEE/ACM International Symposium on Microarchitecture, Istanbul, 2002, 3～14

12 Jourdan S., Ronen R., Bekerman M. *et al*. A novel renaming scheme to exploit value temporal locality through physical register reuse and unification. In: Proceedings of the 31st IEEE/ACM Symposium on Microarchitecture, Dallas, 1998, 216～225

13 González R., Cristal A., Ortega D. *et al*. A content aware integer register file organization. In: Proceedings of the 31st International Symposium on Computer Architecture, Munich, 2004, 314～324

14 Tullsen D. M., Eggers S. J., Levy H. M.. Simultaneous multithreading: maximizing on-chip parallelism. In: Proceedings

of the 22nd International Symposium on Computer Architecture, Santa Margherita Ligure, 1995, 392～403

15 Marr D. T., Binns F., Hill D. L. *et al*. Hyper-threading technology architecture and microarchitecture. Intel Technique Journal, 2002, 1(1): 4～15

16 Spracklen L., Abraham S. G.. Chip multithreading: Opportunities and challenges. In: Proceedings of the 11th International Symposium on High-Performance Computer Architecture, San Francisco, 2005, 248～252

17 Xiao Gang, Zhou Xing-Ming, Xu Ming, Deng Kun. SMA: A speculative multithreaded architecture. Chinese Journal of Computers, 1999, 22(6): 582～590(in Chinese)
(肖　刚,周兴铭,徐　明,邓　鹍. SMA:前瞻性多线程体系结构. 计算机学报, 1999, 22(6): 582～590)

18 Kessler R. E.. The Alpha 21264 Microprocessor. IEEE Micro, 1999, 19(2): 24～36

19 Monreal T., González A., Valero M. *et al*. Delaying physical register allocation through virtual-physical registers. In: Proceedings of the 32nd International Symposium on Microarchitecture, Haifa, 1999, 186～192

20 Uyemura J. P.. Introduction to VLSI Circuits and Systems. Zhou R. D. translate. Beijing: Publishing House of Electronics Industry, 2004(in Chinese)
(尤耶缪拉 J. P. 著. 超大规模集成电路与系统导论. 周润德译. 北京:电子工业出版社, 2004)

21 Shen J. P., Lipasti M. H.. Modern processor design: Fundamentals of superscalar processors. Zhang C. Y. *et al*. translate. Beijing: Publishing House of Electronics Industry, 2004 (in Chinese)
(沈 J. P. 著. 现代处理器设计——超标量处理器基础. 张承义等译. 北京: 电子工业出版社, 2004)

22 Austin T., Larson E., Ernest D.. Simplescalar: An infrastructure for computer system modeling. IEEE Computer, 2002, 35(2): 59～67

YANG Hua, born in 1974, Ph. D. candidate. His research interests mainly focus on chip multithreading architecture, fault-tolerant processor architecture.

CUI Gang, born in 1947, professor and Ph. D. supervisor. His current research interests include fault-tolerant computing, and high-performance computer architecture.

LIU Hong-Wei, born in 1971, Ph. D., associate professor. His current research interests include reliable computing, and mobile computing.

YANG Xiao-Zong, born in 1939, professor and Ph. D. supervisor. His current research interests include fault-tolerant computing, and wearable computing.

Background

In the forthcoming chip multithreading era, a large multi-ported register file is indispensable for SMT processors with deeper pipeline, wider issue, and more thread-level parallelism. However, larger register file incurs longer access delay and more power consumption, both of which are critical to the overall performance, and very likely to become a bottleneck for future SMT processors. Taking advantage of the operand distributing features, this paper presents a novel register file organization that empowers SMT processors to achieve higher performance with fewer registers, effectively alleviating the design pressures.

This work is supported by the 10th 5-Year pre-research project under grant No. 41316. 1. 2, the National Natural Science Foundation of China under grant No. 60503015, and the Ph. D. Supporting Foundation from the Ministry of Education of China under grant No. 20020213017. An important objective of the projects is to research and develop high-performance and reliable computer systems, which ultimately rely on high-performance and reliable processor architectures.