维普资讯 http://www.cqvip.com

计算机科学 1997Vol.24№.4

指令级并行之发展与展望

The Development and Prospect of Instruction-Level Parallelism

胡良校 陈耀强 方滨兴 胡铭曾

(哈尔滨工业大学计算机科学与工程系 哈尔滨 150001) (香港城市大学电脑科学系)

摘 要 Although techniques to achieve some amounts of ILP (Instruction-Level Parallelism) have been present in uniprocessors for over 30 years, new novel techniques for ILP are constantly emerging and almost all of the recent announced high performance processor incorporated ILP. This paper gives a survey of ILP studies, introduces the related technique and processor architectures to exploit ILP, compares the techniques used in recent advanced microprocessor. In conclusion the paper discusses the future direction to explore more ILP.

关键词 Instruction-level parallelism, Fine-grain parallelism, Superscalar, VLIW

所谓指令级并行性又称细粒度并行,主要是相对粗粒度并行而言的,后者是指存在于程序间(主要是进程或线程间)的并行性。顾名思义,指令级并行是指存在于指令一级即指令间的并行性,主要是指机器语言一级,如存储器访问指令、整型指令、浮点指令之间的并行性等,它的主要特点是并行性由处理器硬件和编译程序自动识别和利用,不需要程序员对顺序程序作任何修改。正是由于这一优点,使得它的发展与处理器的发展紧密相连。

在某种程度上,指令级并行性在 40 和 50 年代就已得到机器设计者的注意。现今称作水平微码的并行性在 1946 就出现在 Turing 关于 Pilot ACE[2] 的设计中。事实上,1953 年,Wilkes 和 Stringer 写到"在某种情况下,两个或多个微操作有可能同时发生"[3]。

在 60-70 年代,出现了两种非常著名的试图开发指令级并行性的机器 CDC600[4]和 IBM360/91[5],这两种机器中的很多创造性思想,直到如今,仍广泛出现于各现代高性能处理器的设计中。但是在 70 年代之前,没有人就程序中究竟存在多少指令级并行性作过专门研究。

1970 年,Tjaden 和 Flynn[6]基于他所提出的预译码栈(predecode stack)衡量了单指令流多数据流中的并行度,其结果表明当栈大小为 10 时,平均只有 1.86 条指令可并行执行。

1972 年,Riseman 和 Foster[7]衡量了转移指令对指令级并行度的严重限制。他们发现,如果没有无穷多的硬件资源的支持,程序中存在的指令级并行性很少。

从七十年代开始,以上述两篇文章为代表的一些类似的文章,使得开发指令级并行性的研究在随后将近十年的时间里处于低谷。直到 1984 年,Nicolau 和 Fisher[8]在他们的文章中断言:在假设转移的预测 100%准确并不限制硬件资源的前提下,可以获得的加速比随应用程序的不同在 4 到 988 之间。于是在科学计算的程序中存在大量的指令级并行性之后,开发指令级并行性的研究开始成为热点。

由于 Fisher 等 1984 的实验结果是基于假设转移预测率 100%及不限制硬件资源的前提下得来的,这一假设实际上并不切实际,为了测量在相对实际的情况下在程序中所能开发出的指令级并行性,Jouppi、Sohin、Smith 和 Wall[9-12]分别于 1989 和 1991 年在不同的资源限制下,基于不同的机器结构,使用不同的标准程序对所能获得的指令级并行度进行了研究。但大多数的实验结果都与 Fisher 的结果相去甚远。究其原因,是因为结果表现出存在大量指令级并行性的研究大多针对科学运算程序,对硬件资源往往不加限制,对转移预测率和指令延迟进行了理想化假设。而对于表明指令级并行性很小的研究,一般都是基于某些特殊的机器模型,而且研究都局限于基本块内,忽略了编译和硬件调度机制在基本块间对指令进行调度的能力,有的研究虽然也采用了推测式执行机制,但往往不允许多个控制流同时执行。

维普资讯 http://www.cqvip.com

除此之外，这些人在实验中所用到的处理器模型尽管在那时算是设计能力的上限，但与最近的及正在推出的处理器相比，它们最多只能算是基本配置。最近，Lam 和 Wilson[13]审查了推测式执行的局限性，并指出，若允许处理器在多个方向上进行预测，仍有可能开发出更多的并行性，这导致需要多个程序计数器，这一思想可能是未来处理器结构的发展方向，因为它们是一种界于通用处理器结构和传统的多处理器结构之间的混合结构。

一、指令级并行技术的发展

1.1 推测式执行

在基本块内只有少量的指令级并行性，欲开发基本块之间的并行性就必需对转移指令进行特殊处理。推测式执行是指处理器在某种机制的支持下，对于一条认为极有可能要执行的指令，即使在本来不知道这一条指令是否应被执行的情况下仍允许处理器执行这条指令，这就需要处理器有能力取消任何本不应该执行的指令所产生的影响。推测式执行是一项开发更多指令级并行性的重要技术，其中主要包括使用条件式指令[14](Conditional or Predicated Execution)、将转移预测与寄存器换名[15]和重序缓冲[16](reorder buffer)技术相结合的基于硬件的推测式执行等。

推测式执行的概念起源于 IBM 360/91 中的有限意义的推测。近期的处理器大都把 360/91 中的动态指令的概念与一个缓冲器相接合以实现结果按程序中原来的顺序写入。1988 年，J. E. Smith 和 Pleszkun[16]考查了缓冲在实现精确中断中的作用，并描述了重序缓冲的概念。1990 年 Sohi[17]描述了加上寄存器换名和动态调度技术的想法，使得使用推测机制成为可能，Patt[18]其及同事描述另外一种被称为 HPSm 的方法，也是 Tomasulo 算法的一种扩展，且支持类似推测式的执行。

推测式执行在支持多指令分发的处理器中的作用是 Smith，Johnson 和 Horowitz[11]在 1989 使用重序缓冲技术评价的，他们的目标是为了研究在非科学运算的代码中使用多指令分发和推测式执行所能获得的并行性。在随后的一本书中，M. Johnson[19]描述了推测式超标量处理器的设计。

1.2 指令的动态调度

指令的动态调度技术是指由硬件来安排指令的执行以减少流水线的停顿(Stalls)，其主要目的是让指令尽早执行，即尽量不因指令序列前面的指令的停顿影响后面的指令的执行，即乱序执行。其中，"记分板"(Scoreboarding)[4]技术和 Tomasulo[5]算法是动态调度技术中的两项经典技术。

1964 年，CDC 公司的在其 CDC6600 第一次使用了著名的"记分板技术"，和"乱序执行"的概念，掀开了指令的动态调度技术的新一页。其主要思想是通过一个记分板来保证数据依赖关系进而控制指令的发送、执行与结果写入，它可以充分开发程序中的指令级并行性，使程序的真相关引起的停顿减到最小，采用这项技术，一条指令如果与在它前面的指令之间存在写写相关或写读相关，则它不能被发送到功能部件，如果在要将结果写回时发现有读写相关，则其结果要延迟写入。

IBM 360/91 中使用的 Tomasulo 算法实际上是"记分板"技术的一种扩展。在 Tomasulo 算法中，由于使用寄存器换名技术，一条指令不会因为数据相关而只会因为结构相关被禁止发送到功能部件的保留站，即只要保留站未被占满，指令的分发不会因为数据依赖和资源冲突的存在而停止，因而能做到乱序发送。在保留站中一条指令只要两个操作数准备好后，即可以开始执行。Tomasulo 算法是指令动态调度技术历史上的一个里程碑，360/91 中使用的许多与 Tomasulo 算法相结合的技术，如数据标志、寄存器换名、内存冲突的动态检测、一般化的相关通道以及指令预测等等，直到今天仍被广泛使用。

1.3 循环展开、软件流水与踪迹调度

循环展开(Loop unrolling)[20]是针对循环结构而言的，其主要思想是如果在编译时可以确定循环的次数，那么可以把循环体重复多次而扩展为顺序程序，这样不仅可以减少循环转移的开销，而且由于基本块增大，块内的进一步优化也成为可能。

由于循环展开使代码空间有较大程序的增大，而且对 Cache 的命中率也有不良影响，因而提出了软件流水(Software pipelining)[21]的概念，其主要思想是在不改变程序语义的前提下，使循环的不同迭代的操作重叠，以利用硬件资源的并行性，从而缩短循环程序的执行时间。

踪迹调度(Trace scheduling)[20]技术是 VLIW 机器中广泛采用的生成 VLIW 目标程序过程中的一项重要技术，它包括两个独立的过程：一是"路径选择"过程，找出最有可能执行的操作序列。二是"路径压缩"过程，其任务是将选定的路径碾压成数量较少的宽指令。

维普资讯 http://www.cqvip.com

二、主要处理器结构与早期处理器

从结构上来说，开发 ILP 的处理器主要有三种：最主要的是超长指令字结构 VLIW(Very Large Instruction Word)[22]和超标量结构(Superscalar)[23]，另外还有一种去耦结构(Decoupled Architecture)[24]。其中，超长指令字结构主要依赖于编译的力量发现程序中的并行性，并将可以并行执行的操作尽可能组装在一个指令中，硬件只需机械地完成指令中指定的操作即可。为了获得适度的并行性，这种结构的处理器的指令字必需具有足够的长度。这种结构可以开发较大的并行性，但其一个致命缺点是缺乏兼容性，这也是最终导致失败的原因。与此相反，超标量结构则主要依靠硬件来发现程序中的并行性，将可以并行执行的指令分发给不同的功能部件使之得以在同一周期执行，即由硬件来进行指令的动态调度。由于这种结构具有良好的软件兼容性，所以当今的高性能处理器无一不采用这种结构。去耦结构的处理器则由于性能不如超标量结构的处理器。因而不可能战胜超标量结构的处理器而成为通用处理器的主流。

2.1 VLIW 结构的处理器

1981 年，Charlesworth 提出了浮点处理系统 AP-120B[21]，这是第一个在一条指令中使用包含多个操作的宽指令字的处理器系统。在其编译器中和手写的汇编函数库中都运用了软件流水的思想。由于它是一个后端处理器，多指令分发技术在通用处理器中的许多问题都被忽略了。

Standford 大学的 MIPS 处理器有能力在一条指令中包含两个操作，但是后来由于性能的原因在其商业化版本中去掉了这一特点。

1983 年 Fisher 和他在 Yale 大学的同事，提出了设计一个指令字宽为 512 位的处理器的建议，并首次命名为超长指令字 VLIW[23]。并利用 Fisher 在 1981 年为水平编码而开发的踪迹调度技术，为这种处理器生成了代码。最著名的 VLIW 计算机--Multiflow 的 TRACE[25]计算机就是主要基于 Yale 大学的基本概念所开发出来的。Multiflow TRACE 共有两个系列六个品种：7/200，14/200，28/200，7/300，14/300，28/300。其最基本的 7/200 有一个 256 位的指令字，包含七个可同时执行的操作：4 个整型操作，2 个浮点操作和一个转移操作。Multiflow TRACE 计算机卖出了 100 台以上，但由于是由一个小公司开发一套新的指令系统，VLIW 结构本身不具备兼容性，以及不具备与 RISC 微处理器竞争的能力等诸多问题，最终导致了 Multiflow 的失败。

几乎与此同时，Cydrome 公司正在建造一个类似 VLIW 的处理器 Cydrome Cydra 5[26]，Cydra 系统主要由六个用于通用应用程序处理的交互处理器，两个输入/输出处理器，一个厍于控制台的服务处理器和一个 VLIW 机构的数值处理器构成。所有的处理器都通过一条公用总线相互联接，并通过它与存储系统联接。其指令有两种格式：多操作模式指令和单操作模式指令，前者为 256 位，包含六个处理操作和一个用于控制指令部件的操作；单操作模式指令只含一个操作。在 Cydra 5 中采用了动态寄存器换名技术，并对软件流水线提供额外的支持。Cydra 5 是第一个融合了软件和硬件，包含了条件转移指令，旨在开发指令级并行的处理器。与 Multiflow 相比，它更多地依赖于硬件，主要是在向量代码上获得有竞争力的性能。最后，由于存在与 Multiflow 类似的问题，也未能在商业上获得成功。

2.2 超标量结构的处理器

IBM 在同时分发多条指令方面曾作了些开拓性的工作。早在 1970 年左右，就有一个叫做 ACS 的项目在进行中，它包含了多条指令分发的概念，但遗憾的是最后未能变成产品。最早的动态分发指令的超标量处理器是 John Cocke 提出的，他在 80 年左右的几次谈话中描述了超标量处理器的关键思想，并命名了超标量 Superscalar 这个词。最原始的设计是 America[23]，IBM 的 Power-1 体系结构(RS/6000 生产线)主要基于这些思想[27]。

1964 CDC 公司推出了第一台 CDC 6600 计算机，CDC6600 的出现无论在指令级并行研究的历史上，还是微处理器研究的历史上都有重大影响。CDC6600 的很多特点在当时都是独一无二的创举。除了记分板技术外，CDC6600 是第一个大量使用多功能部件的处理器，它有 10 个功能部件：整型加法部件和整型移位部件各一个，有两个整型递增部件(Increment)，两个整型乘法部件，一个逻辑转移部件，一个浮点加法部件和一个浮点除法部件。同时它还有一个采用分时流水线的外围设备处理器。第一次使用了乱序执行的概念，同时还有一个 FORTRAN 编译器的指令调度器。CDC6600 的执行模型实际上是超标量处理器的雏形。

IBM 在 1960 年左右介绍并在 1967-1968 年间推出了它的第一个 IBM 360/91 计算机。与

维普资讯 http://www.cqvip.com

CDC6600 相比，IBM360/91 的功能部件少得可怜．它只有一个整型加法器，一个浮点加法器和一个浮点乘除部件。但在试图重新安排动态执行的指令顺序，以使功能部件尽可能忙碌方面远比 CDC 6600 更为野心勃勃。其指令的动态调度技术则是当今超标量处理器的关键技术。IBM 360/91 引进了许多新的概念，包括数据标志、寄存器换名、内存冲突的动态检测、一般化的相关通道以及 Tomasulo 算法和指令预测的使用。许多 360/91 中的思想在九十年代被广泛使用之前几乎被埋没了近 25 年。IBM360/91 这一开发指令级并行的先驱者和 CDC 6600 一起开辟了超标量处理器结构的新纪元。

2.3 去耦结构的处理器

J. E. Smith 和他在 Wisconsin 的同事于 1984 年提出了包含多条指令分发和有限动态流水线调度的"去耦"[24]体系结构的想法，这个结构的一个关键特征就是通过一系列的寄存器队列来去掉存取部件和执行部件间的紧耦合，使之变成松散耦合系统。同时也通过队列来保证存取部件和执行部件内指令的正确顺序。所谓存取部件和执行部件实际主要是指整型部件和浮点部件，整型部件完成地址计算、存储器存取，浮点部件完成浮点指令的执行。由于浮点指令一般需要多个周期才能完成，同时由于队列的存在，使得整型部件与浮点部件之间是松散耦合关系。1987 Smith 等介绍的 Astronautics ZS-1[28]计算机就采用了这一方法，使用队列来联接 Load/Store 部件和操作部件。IBM Power-2 体系结构也以类似的方式使用了队列。

三、现代高性能的处理器

由于集成电路工艺与技术的飞速发展，现代高性能的处理器在开发指令级并行性的能力方面与 CDC 6600 和 IBM 360/91 相比，不可同日而语。当代高性能的处理器在一个芯片上实现的功能比当时整个计算机系统的功能还复杂得多。象 Power PC620 就在一个芯片中包含了三个定点部件、一个浮点部件、一个存储器装入和写入部件和一个转移处理部件。片内除有一个 32K 的指令 Cache 和一个同样大小的数据 Cache 外，还有一个支持 1M 到 128M 片外 Cache 的接口。其每个周期最多可以发送四条指令，而且支持指令的乱序发送与乱序执行。除此之外，它还支持精确中断。

表 1

处理器	推出时间	时钟频率	发送结构	调度技术	发送能力	最多能发送各类指令				性能指标 (SPEC92)	
						存储访问	整型操作	浮点操作	转移操作		
IBM Power-1	1991	66	动态	静态	4	1	1	1	1	60int	80FP
HP 7100	1992	100	静态	静态	2	1	1	1	1	80int	150FP
DEC Alpha 21064	1992	150	动态	静态	2	1	1	1	1	100int	150FP
Super SPARC	1993	50	动态	静态	3	1	1	1	1	75int	85FP
IBM Power-2	1994	67	动态	静态	6	2	2	2	2	95int	270FP
MIPS TFP	1994	75	动态	静态	4	2	2	2	1	100int	310FP
Intel Pentium	1994	66	动态	静态	2	2	2	1	1	65int	65FP
Cyrix M1	1994	100	动态	动态	2	2	2	1	1	优于 P5	不详
HP PA7200	1994	140	动态	静态	2	1	2	1	1	175int	250FP
DEC Alpha 21164	1995	300	静态	静态	4	2	2	2	1	330int	500FP
Sun Ultra SPARC	1995	167	动态	静态	4	1	1	1	1	275int	305FP
Intel P6	1995	150	动态	动态	3	1	2	1	1	>200int	不详
AMD K5	1995	100	动态	动态	4	1	2	1	1	130int	不详
HaL R1	1995	154	动态	动态	4	1	2	1	1	255int	330FP
Power PC 620	1995	133	动态	动态	4	1	3	1	1	225int	300FP
MIPS(T5)R1000	1995	200	动态	动态	4	1	2	2	1	300int	600FP
HP 8000	1996	200	动态	静态	4	2	2	2	1	>360int	>550FP

维普资讯 http://www.cqvip.com

表1是近几年来所公布的各主要处理器一览表[29]，从表中可看出，几乎所有的处理器都具有并行发送多条指令的能力，显然今后的处理器也将包含这一能力。另外，还可看到，提高处理器的性能有两条主要途径：一是依靠指令的动态调度和推测式执行来获取高性能；二是使用静态调度并竭力提高处理器的速度来获得较高性能。但目前还看不出那种途径具有更多的优越性。

四、指令级并行研究的展望

从目前的形势看，开发指令级并行性的处理器中，由于VLIW结构的处理器不具有目标代码的兼容性，超标量处理器就占绝对优势。今后，VLIW结构的处理器如果仍不能解决这一主要问题，也不会有多大起色。最近几年出现的TTA[30]，DS[31]，PEWs[32]，MISC[33]，Multiscalar[34]等结构，要么仅仅是一种概念型结构，要么缺乏目标代码的兼容性，因此也很难在通用处理器市场上与超标量处理器一争高低，最终的命运只能与VLIW相同。

然而，超标量处理器要想开发出更多的指令级并行性，获取更高的IPC(Instructions Per Cycle)，必需解决如下三个主要问题：及时提供指令，及时提供数据，增大程序中的并行性。

为了及时提供所需的指令，一个可能的途径是借助编译技术增大基本块，这等价于减少转移指令，使指令流被中断的机会减少。另一种途径是进一步优化转移处理策略，比如说转移预测的准确率。对超标量处理器性能影响最严重的是转移指令，由于5%的误预测率都会带来较高的性能损失，因此如何在允许的条件下提高转移预测率以及探索别的处理转移指令的方法，将是今后的主要方向[35,36]。

为了及时提供所需的操作数，必需有较好的数据预取策略[37]和更为有效且命中率更高的Cache结构和存储系统。

增大程序中并行性，最主要的是增大基本块的大小。途径之一是采用新的编译技术如超块技术[38,39]等来增大程序的基本块。途径之二是采取如多条件码域[40]和条件式执行等更为有效的转移处理策略尽量减少转移指令的数量。最后多条转移指令的同时预测也是一个非常有希望的方向。

除此之外，由于多存储体系的使用，处理器的通讯延迟有随处理器性能提高而增大的趋势，且INTERNET日益普及，使得与外界的交流，通讯越来越重要，因此，在提高处理器性能的同时，减少通讯延迟将是未来的发展方向。

（参考文献共39篇略）

（上接第35页）

$$F1(X)=\frac{X-1}{0-1}GG(1,1)+\frac{X-0}{1-0}GG(1,2)=-2(X-1)+3X=X+2$$

$$F2(X)=\frac{X-3}{2-3}GG(2,3)+\frac{X-2}{3-2}GG(2,4)=-3(X-3)+4(X-2)=X+1$$

所以 $F(X)=(X+2)+(X-0)(X-1)(X+1)=X^3+2$，即完全拟合。

四、讨论

由实例可见，本文的方法对多项式的拟合是完美的，因此对于能用多项式拟合的数值都可以较好地完成，而且有算法简单，易于实现的特点。但仍需要考虑以下问题：

- 可能需要依据某种技术从轴上对D进一步处理以确定更精确的函数式；
- 在简单情况下可用比较的方式确定更精确的函数式；
- 在不要求拟合函数的光滑性时，可直接采用区间划分所得到的线性分段函数作为目标函数。
- 在数值有误差的情况下是否可用比较的方式确定更精确的函数式？即该方法的鲁棒性。
- 递推过程的 Δ_i 应采用适当精度，该如何选取？
- 对本方法的误差考虑；
- 它对概率与统计学中的参数估计是否有参考坐标？

参考文献

[1]杨青云，《数据处理方法》，冶金工业出版社，1990

[2]徐萃薇，《计算方法引论》，高等教育出版社，1982

[3]李爱中，模型发现和智能决策支持系统工具的研究，哈工大博士论文，1991

[4]Lin, XiaoFen & Ungar, Lyle, Inventing Theoretical terms in Inductive Learning of Function Search and Constructive Methods, Methodologies for Intellingent Systems, 1989.4