文章编号: 1001- 2486(2008) 01- 0032- 05

一种面向分布式寄存器文件的 VLIW 调度新策略*

伍 楠, 文 梅, 张春元
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 新一代面向密集计算的高性能处理器普遍采用分布式寄存器文件来支撑 ALU 阵列, 并通过 VLIW 开发指令级并行。面向分布式寄存器文件的编译成为新兴的研究热点, 在斯坦福大学的 kernelC 编译器 ISCD 中最早提出了面向分布式寄存器的 VLIW 调度问题, 在该领域处于领先水平, 但是没有解决重负载下的分布式寄存器分配问题, 使应用编程受到极大限制。在其基础上提出了一种新的 VLIW 调度策略, 实验结果表明能够很好地解决重负载下的分布式寄存器分配问题。

关键词: 分布式寄存器文件; 编译; 寄存器分配; VLIW
中图分类号: TP303 文献标识码: A

A Novel VLIW Schedule Strategy for Distributed Register Files

WU Nan, WEN Mei, ZHANG Chun-yuan
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Newly-emerging high performance processors for intensive computing generally use distributed register files to support ALU array and to explore instruction level parallelism (ILP) by VLIW. Research of the VLIW compiler for distributed register file has been a hotspot. KernelC compiler ISCD developed by Stanford University was the first of its kind to deal with VLIW schedule for distributed register file and the only compiler actually applied as well. However, it limited the program for its weakness of scheduling the code with heavy workload. Based on ISCD, this paper presents a new VLIW schedule strategy. Experimental results show that it can deal with distributed register file allocation with heavy workload very well.

Key words: distributed register files; compile; register allocation; very long intruction word

计算密集型应用对处理器的计算能力提出了新的要求, 专门针对密集计算的可编程处理器如 Imagine[1], Merrimac[2], TRIPS[3], Cell[4], FT64[6] 等应运而生。它们通过大规模的 ALU 阵列来达到超高的计算性能, ALU 之间分组以 SIMD、MIMD 或多线程的方式处理数据, 对寄存器容量和带宽要求极高。传统的集中式寄存器文件不能满足要求, 因而大多采用分布式寄存器文件, 以共享互连总线的方式连接, 同时采用了超长指令字(VLIW, Very Long Instrction Word) 技术开发指令并行。斯坦福大学的 Scott Rixner、Peter Mattson 等人在 20 世纪 90 年代中期提出微处理器中的分布式寄存器的概念[1], 随后设计完成了与之配套的 kernelC 编译器 ISCD, 并不断完善至今。ISCD 全面引入了面向分布式寄存器的 VLIW 调度问题, 已经实际应用于 Imagine 流媒体处理器和 Merrimac 超级计算机结点。但是由于 ISCD 在设计之初只考虑了媒体应用, 没有过多考虑计算负载更重的科学计算问题, 因此当 ISCD 应用领域扩展到科学计算以后, 大量重负载的程序无法编译调度通过, 极大地限制应用编程, Peter Mattson 也认为这是一个尚未解决的关键问题[5]。

1 背 景

基于分布式寄存器的处理器体系结构采用了分布式的多个寄存器文件代替了集中式的单一寄存器

* 收稿日期: 2007- 05- 26
基金项目: 国家自然科学基金资助项目(60673148; 60703073); 国家“863”计划资助项目(2007AA01Z286); 国家教育部博士点项目(20069998025)
作者简介: 伍楠(1981—), 男, 博士生。

文件, 每个功能单元拥有 1~ 3 个本地寄存器文件(LRF), 本地寄存器文件的读端口直接与对应功能单元的输入端口连接, 每个功能单元只能从自己的本地寄存器中读取操作数。所有寄存器文件的写端口同所有功能单元的输出端口之间以交叉开关方式连接, 使得数据可以在不同寄存器间通信传输。交叉开关保证了每个功能单元的输出端口通过一个交叉点可以无阻塞地与一个空闲寄存器文件写端口相连, 拥有极高的带宽。由于拥有大量的功能单元, 一般采用 VLIW 的方式开发指令级并行。

斯坦福大学的 kernelC 编译器 ISCD[7] 在传统 VLIW 调度的基础上引入了通信调度技术, 提出了面向分布式寄存器的 VLIW 编译策略: 将程序划分成若干基本块(根据循环进行划分) ; 按指令驱动的方式对一基本块内的操作进行调度: 第一步选定一个操作, 第二步将操作安排到满足相关性要求的最早的一个周期, 第三步为操作分配一个空闲并可以执行该操作的功能单元, 最后进行通信调度。如果通信调度成功, 则调度下一个操作; 如果失败, 则返回前面的步骤, 重新选择周期或功能单元。当所有基本块的所有指令调度完成以后, 按照传统方式基于图着色算法分配寄存器。

对于多媒体应用, ISCD 的 VLIW 调度策略能够很好发挥作用, 但是当进入科学计算领域以后, 程序的计算量急剧增加, 子程序的基本操作数大幅提高(见表 3) , 对寄存器负载的要求也大大提高, 寄存器分配常常失败。实际上, 即使在媒体应用中也会出现寄存器分配失败造成编译失败的情况(表 3 中的 blocksearch) 。

可以发现造成寄存器分配失败的主要原因: 原有的调度策略只在最后一步才考虑寄存器的分配。其实寄存器分配失败并不是由最后一步的寄存器分配算法造成的, 因为当前面的指令调度和通信调度结果确定以后, 每个变量在何时存储到哪个本地寄存器文件就已经确定了, 最后的寄存器分配算法只是为存储到本地寄存器中的变量分配一个地址编号而已。在分布式寄存器结构下, 寄存器分配算法不能在不同寄存器文件之间分配空间, 只能在本地寄存器中为变量分配空间。因此如果某一时刻, 变量在时间和空间上过于集中, 也就是通信调度到该寄存器文件的变量个数超过该寄存器文件容量时, 寄存器分配就会失败。而在重负载情况下, 这种情况发生的几率非常高。

2　基于 DAG 的基本块重划分调度策略

本文提出在 VLIW 调度的第一个过程——基本块划分时就考虑通信调度和寄存器负载的问题, 并且当最后寄存器分配失败时, 返回第一步, 根据前次调度结果调整基本块划分, 再次进行调度直至成功。可以证明, 如果不停地增加新的基本块直至每个基本块只有一个操作, 则肯定可以调度通过, 因此该反馈经过最多 N (N = 操作个数) 次迭代一定会停止, 实际测试结果表明, 该过程平均反馈 2 次就可以保证调度通过。由于过多地划分基本块可能降低编译结果的性能, 因此第一次进入该调度过程时, 仍按照原来基于循环的基本块划分, 只有当最后通信调度或寄存器分配失败时才返回第一步重新划分, 如果调度成功就不反馈。

当通信调度或者寄存器分配失败时, 就必须重新划分前次调度失败的基本块。基本块的重划分有两种方法: 一种是将原有调度失败的基本块分割成增加新的基本块; 另一种是调整原有基本块的范围。不管何种方法都是基于前面预编译生成的数据倚赖关系图(DAG) 。整个 DAG 可以抽象为一个有向图, 根据循环分支划分以后, 每个基本块的 DAG 是一个非循环有向图, 其每个弱分支是一个非循环的连通有向图(注意不是有向树, 因为其结点的入度可能不为 1) 。图 1 示例了一段程序生成的 DAG。

为了更好地描述基本块的重划分策略, 参考网络流问题中割的概念[8] 定义基本块划分中的割:

定义 1　基本块内的 DAG 图 G = < V, E, Ψ> 是非循环有向图, 其中, V 为结点集, Ψ: $E \rightarrow V \times V$。S 为图 G 中所有入度为 0 的结点的集合, T 为图 G 中所有出度为 0 的结点的集合。

定义 2　设图 G = < V, E, Ψ> , 若 $V1$ 属于 V, $V1$ 与~ $V1$ 之间的边同向(都为输入边或都为输出边, 这一点保证了新生成的基本块不会发生嵌套) , 横跨 $V1$ 和~ $V1$ 之间的边记为 E_g , | E_g | 为割的容量。所有割中| E_g | 最大的为最大割, | E_g | 最小的为最小割。若| $V1$ | = | ~ $V1$ | , 则称为等分割, 等分割既可能是最大割, 也可能是最小割。

定义 3　设图 G = < V, E, Ψ> , ($V1$, ~ $V1$) 为 G 的割。若 S 属于 $V1$ 且 T 属于~ $V1$, 则称($V1$,

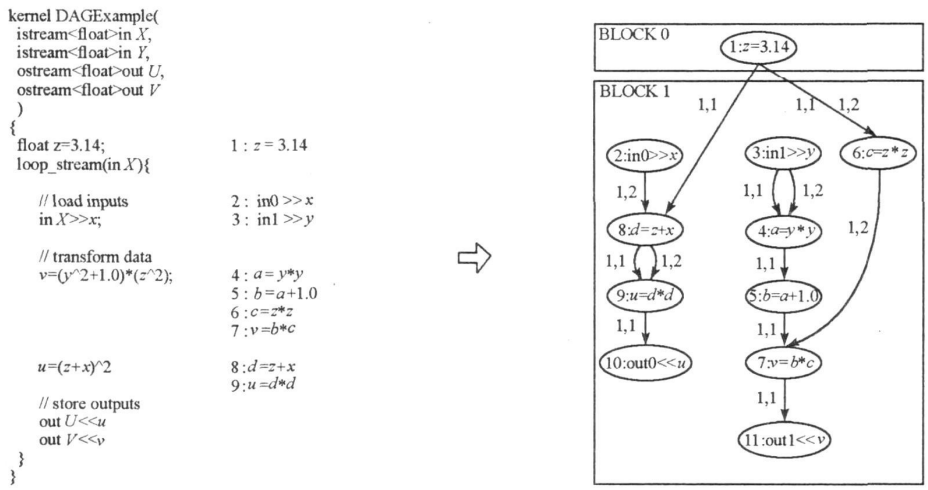

图 1 程序映射生成的数据倚赖关系示例

Fig. 1 Architecture of MASA processor

~V1) 为 G 的分离S 与T 的横割。若存在 $s1, s2 \in S$ 使得$s1 \in V1$, $s2 \in \sim V1$, 或者存在 $t1, t2 \in T$, 使得 $t1 \in V1, t2 \in \sim V1$, 则称($V1, \sim V1$) 为 G 的分离S 或T 的纵割。分离图 G 弱分支之间的割, 可以看作是容量为 0 的纵割。

在计算重负载应用, 特别是科学计算中, 程序的并行性很好, 存在大量的弱分支或者可并行分支, 经过 VLIW 调度以后集中在一个相对较短的时间内执行, 这也正是造成某段时间寄存器过载的主要原因。对于相关性强、并行性差的基本块, 即使操作很多, 由于无法在时间上集中, 也不会造成寄存器过载。通过割将原来的基本块划分成多个新的基本块, 新基本块中的结点数减少。通过选择不同的割策略, 可以有效地控制基本块内变量在时间和空间上的集中程度。横割不会破坏弱分支(或可并行分支) 间的并行性, 因此 VLIW 打包的效率仍然较高, 变量集中程度也较高。纵割破坏原有分支间的并行性, 但仍保持原有关键路径的长度, 因此 VLIW 打包效率有较大降低, 变量的集中程度也较低。除此之外, 不同割策略, 如最大割、最小割、等分割、不等分割等也会改变基本块的负载、并行性等特征, 这些特征之间可能是非正交的。表 1 概括了不同割策略对基本块可能的影响(这里的影响是指新产生的基本块相对于原来的基本块而言, ↑ 表示大幅增加, ↓ 表示大幅减小, → 表示影响较小, – 表示不确定, 可能与同时采用的其他割策略相关)。

表 1 不同割策略对基本块可能造成的影响

Tab. 1 Machine parameter of MASA simulator

	横割	纵割	最小割	最大割	等分割	非等分割
结点数	↓	↓	↓	↓	↓	→
分支间的并行性	→	↓	↓	→	–	→
关键路径长度	↓	→	–	↓	–	→
通信调度难度	↓	↓	↓	↑	–	→
VLIW 调度效率	→	↓	↓	–	↓	→
寄存器负载	↓	↓	↓	↓	↓	→

基于割基本块划分通过 DAG 充分考虑了程序全局情况, 且确定割的算法复杂度仅为 $O(|E|)$[8], 因此当寄存器分配失败时, 能够快速而有效对分配失败的块进行合理重划分, 保证下一次调度分配成功。但是无论如何, 由于表 1 中的特征对与最终编译的结果是非正交的, 况且 VLIW 调度又是一个 NP 完全问题, 很难在指令调度最终完成之前精确预测最终的寄存器分配情况。因此, 必须有一条从分配失败到重划分基本块的反馈回路, 这条反馈回路保证了本文的新策略可以完全解决寄存器分配失败的问题, 并

且可以通过选择割策略在编译后的程序性能和编译成功率之间折衷。

编译 ElSd 程序的寄存器分配结果如图 2 所示, 结果中记录了所有本地寄存器的分配情况, 最大容量(max) 是指本地寄存器文件硬件设计的寄存器个数, 最大分配数(use) 是指某一时刻分配给该寄存器的最大变量数目。当任何一个本地寄存器的最大分配数大于寄存器最大容量时便是寄存器分配失败。根据如图 2 所示的反馈结果选择合理割策略非常重要, 合理选择割策略可以使得反馈次数最小, 还可以保证编译效率损失最小。

REGISTER ALLOCATION FAILURE!

// maximum register allocation:																	
// idx:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
// max:	256	32	2	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
// use:	62	7	2	33	31	31	35	36	40	37	39	25	30	30	22	28	14	20
// idx:	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32			
// max:	32	16	16	16	16	16	16	1	1	1	1	1	1	1	1			
// use:	15	10	3	2	8	1	1	0	0	0	0	0	0					

(a) 原有策略下的寄存器分配结果

ALLOCATE REGISTERS:

// *maximum register allocation:*																		
// idx:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
// max:	256	32	2	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
// use:	62	7	2	27	29	29	29	32	30	29	32	27	18	21	27	22	13	15
// idx:	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32			
// max:	32	16	16	16	16	16	16	1	1	1	1	1	1	1	1			
// use:	16	10	3	2	4	1	1	0	0	0	0	0	0	0	0			

(b) 新策略下的寄存器分配结果

图 2　不同调度策略下的寄存器分配结果

Fig. 2　Kernel-stream graph of Ygx2

3　实验结果

本文在原有的 kernelC 编译器 ISCD 中引入了基于 DAG 的基本块重划分调度策略, 并在其中实现了多种基本块重划分方法, 以命令参数的形式指定, 表 2 列出了可用参数。其中不矛盾的参数可以一起使用, 比如等分横割可以表示为– h– e。非等分割– ue x 中的参数 x 是一个浮点数表示割中| $V1$| : | ˜ $V1$| 的比值, 默认情况为 4. 0, 即 4∶ 1。– all 表示在重划分基本块时, 采用启发式规则, 该规则根据上次寄存器分配的结果(图 2 所示) 选择合适的割策略。默认的规则是: 当所有本地寄存器的最大分配数之和小于所有寄存器总容量的 1. 2 倍时采用非等分横割, 大于 1. 2 且小于 2 时采用等分横割, 大于 2 且小于 3 时采用非等分纵割, 大于 3 时采用等分纵割。

表 2　新 ISCD 中调用不同割策略的参数

Tab. 2　Kernels' performance of Ygx2

横割	纵割	最小割	最大割	等分割	非等分割	启发式
– h	– v	– min	– max	– e	– ue x y	– all

本文还选取了一个测试程序集合, 来源于常用的媒体应用程序和袁国兴等所做的科学计算测试集 IAPCM benchmarks[9]。由于主要针对计算重负载程序, 挑选了其中 10 个基本操作数大于 1000 的重负载子程序, 如表 3 所示。

所有测试程序针对面向科学计算的 FT64 流处理器, 用 kernelC 语言[7] 改写实现。VLIW 调度策略是针对基本操作的, 采用何种高级语言都不会对结果产生影响。为了尽量减少其他因素的干扰, 调用了 ISCD 的随机调度机制– r 200[7], 为每组调度最大测试 200 个随机种子数, 编译成功即停止。表 4 中反馈次数是新策略下调度成功的最小反馈次数。VLIW 条数是调度成功后生成的 VLIW 条数, 若调度失败则设为∞, 由于系统每个时钟周期发射一条 VLIW, 因此结果直接影响到程序的执行时间, 反映了编译效率。

表 3 测试程序集合及其特征

Tab. 3 Function unit and LRF configuration of MASA and imagine

应用背景/核心子程序	预编译后的总基本操作条数	最初基于循环划分的基本块个数	最大基本块包含的基本操作条数
H.264 Encoding: 视频处理中的 H.264 编码, blsh 运动搜索子程序是其中计算量最大的部分			
Blocksrh	1125	4	409
Span-based Polygon Rendering, 3D 图形渲染			
sortfrag	1482	6	826
ygx2: 二维拉格朗日和欧拉结合法, 主要应用于流体力学, IAPCM			
Uvxr	3075	5	1381
ElSd	2014	3	1936
CAPA0: 快速傅立叶变换, IAPCM			
camain	2431	6	975
polynoms	1556	3	1053
DOSE2: 蒙特卡洛法, 主要应用于粒子输运, IAPCM			
gfia	2855	4	1496
EPT 2: 强 stiff 系统的块迭代法, 主要应用于非平衡系统动力学, IAPCM			
eee	1568	3	837
gear	1887	3	1254
ZZZ1: 跳点格式解二阶常微分方程, 主要应用于分子动力学, IAPCM			
loopKJ	1936	2	1102

表 4 VLIW 调度结果

Tab. 4 Bandwidth hierarchy

		无	-m -ne	-m -e	-m -mi	-m -ma	-v -ne	-v -e	-v -mi	-v -ma	-all
Blocksrh	反馈次数	0	1	1	1	1	1	1	1	1	1
	VLIW 条数	∞	415	488	696	514	426	770	882	423	495
sortfrag	反馈次数	0	0	0	0	0	0	0	0	0	0
	VLIW 条数	326	326	326	326	326	326	326	326	326	326
ElSd	反馈次数	0	4	1	2	2	2	1	1	1	2
	VLIW 条数	∞	856	1478	1621	1254	1830	1503	1640	1524	890
Uvxr	反馈次数	0	5	3	3	3	4	2	3	2	3
	VLIW 条数	∞	1602	2620	1690	1091	1213	1644	1355	1323	1190
camain	反馈次数	0	3	2	2	3	2	1	1	2	1
	VLIW 条数	∞	455	1127	1025	1099	600	854	775	810	538
polynoms	反馈次数	0	2	1	2	2	3	1	1	1	1
	VLIW 条数	∞	550	1533	1420	1145	933	957	896	914	665
gfia	反馈次数	0	4	2	2	2	3	3	2	3	2
	VLIW 条数	∞	921	1247	1389	895	1023	942	903	1025	860
eee	反馈次数	0	2	1	1	2	1	1	1	1	1
	VLIW 条数	∞	474	625	604	566	795	751	683	548	489
gear	反馈次数	0	6	3	5	3	4	2	2	3	2
	VLIW 条数	∞	872	2280	937	1927	1108	1624	2124	1424	913
loopKJ	反馈次数	0	4	4	3	4	4	3	3	3	3
	VLIW 条数	∞	695	799	886	819	721	821	845	804	756
平均反馈次数		0	3.1	1.7	2.1	2.2	2.4	1.5	1.5	1.7	1.6
新老策略总平均值		0					1.98				

(下转第 62 页)

表 1 角反射器图像的主瓣和旁瓣大小

Tab. 1 Mainlobe and sidelobe level of the reflector

补偿方法	距离分辨率 (m)	距离 PSLR (dB)	距离 ISLR (dB)	方位分辨率 (m)	方位 PSLR (dB)	方位 ISLR (dB)
未补偿系统	0.2940	-11.84	-2.079	0.6620	-14.48	-8.47
未补偿色散	0.3280	-11.27	-6.79	0.1640	-14.27	-1.54
测量数据	0.2160	-16.47	-5.79	0.1560	-15.70	-1.46
经验公式	0.2200	-16.92	-6.34	0.1640	-15.75	-1.23

当采用相同型号的不同天线录取相同场景的数据时,采用经验公式的补偿结果也同样非常优良,因此验证了该方法可以有效补偿相同类型的不同天线。

4 结束语

将 Archimedean 天线的群延迟分为色散部分和非色散部分,利用 Archimedean 螺旋线的几何特性揭示了天线固有的色散特性,有效地解决了天线特性未知情况下的补偿问题。将天线群延迟的非色散部分等效为附加系统延迟,利用外部定标体(如角反射器)在回波域近似呈现为双曲线的特性,提出了基于 Hough 变换的延迟估计方法,并利用曲线的对称性降低 HT 的维数,提高了系统延迟补偿的效率。轨道数据的处理结果验证了所提方法的有效性。

参 考 文 献:

[1] Lacko P R, Franck C C, et al. Archimedean-spiral and Log-spiral Antenna Compensation[C]// SPIE, April 2002, 4742: 230-236.

[2] Lacko P R, Clark W W, Sherbondy K. Studies of Ground Penetrating Radar Antennas[C]// 2th International Workshop on Advanced GPR, Delft, Netherlands, May, 2003: 24-29.

[3] 《数学手册》编写组. 数学手册[M]. 北京: 高等教育出版社, 1979: 398-399.

[4] 林昌禄. 天线工程手册[M]. 北京: 电子工业出版社, 2002: 381.

[5] Illingworth J, Kittler J. A Survey of the Hough Transform [J]. Computer Vision, Graphics, and Image Processing, 1988, 44(2): 87-116.

[6] 袁卫鹏, 施鹏飞. 模糊随机 Hough 变换算法[J]. 上海交通大学学报, 2002, 36: 1825-1828.

[7] Yegulalp A F. Fast Backprojection Algorithm for Synthetic Aperture Radar[C]// IEEE Radar Conference, Massachusetts, USA, April 1999: 60-65.

(上接第 36 页)

4 结 论

面向分布式寄存器的新调度策略改进了原有策略中对寄存器分配失败基本块的处理过程。不管基于何种基本块重划分方法,最多反馈 6 次、平均反馈 2 次就可以编译成功。由于新策略算法考虑了全局情况,降低了通讯调度复杂度,可以有效减少编译尝试的随机数个数和通信调度时间,实验中老策略大多尝试 200 个随机数仍无结果,而新策略编译成功时尝试随机数个数都在 30 以内。研究发现,编译成功率和编译效率是部分矛盾的。例如,横割拥有较好的编译效率但成功率却较低,为了编译成功可能要反馈更多的次数,而重划分基本块又会带来性能的下降;纵割虽然成功率较高,编译效率却较低;因此必须在编译成功率和编译效率之间综合考虑并进行折衷。基于启发式规则的调度策略(-all)由于考虑了上一次调度的反馈结果,可以更"聪明"地选择合适的割方法,因此在编译成功率和编译效率上都可以取得较好的效果。

参 考 文 献:

[1] Rixner S, Dally W J, et al. A Bandwidth-efficient Architecture for Media Processing [C]//The 31st Annual ACM/IEEE International Symposium on Microarchitecture, USA, 1998.

[2] Dally W J, Erez M, et al. Merrimac: Supercomputing with Streams [C]// SC'03, USA, 2003.

[3] Sankaralingam K, et al. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS architecture [C]// The 30th Annual International Symposium on Computer Architecture, USA, 2003.

[4] Flachs B, et al. A Streaming Processor Unit for a CELL Processor [C]// IEEE International Solid-state Circuits Conference, USA, 2005.

[5] Mattson P. A Programming System for the Imagine Media Processor[D]. Dept. of Electrical Engineering., Ph. D. Thesis, Stanford University, 2001.

[6] Wen M, et al. FT64: Scientific Computing with Streams [C]// Hipc 2007, India, 2007.

[7] Mattson P, et al. Imagine Programming System Developer's Guide [EB]. http://cva.stanford.edu

[8] Cormen T H, Leiserson C E, et al. Introduction to Algorithms (Second Edition) [M]. 北京: 高等教育出版社, 2002.

[9] 袁国兴, 等. 评几种高档微处理器在运算科学计算问题时的性能[EB]. http://www.ccw.com.cn.