基于 region 的动态重用技术*)

张可新　张兆庆

(中国科学院计算技术研究所体系结构研究室　北京100080)

Dynamic Reuse Based on Region

ZHANG Ke-Xin　ZHANG Zhao-Qing

(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

Abstract　Empirical observations suggest that many instructions and groups of instructions having the same inputs, and producing the same outputs. Such instructions do not have to be executed repeatedly—their result can be obtained from records where they have been saved previously. This paper proposes an approach that uses compiler technique to exploit reuse for instruction group. In this approach, the compiler first identifies code regions whose computation can be reused during dynamic execution. For each region, a library routing is invoked before the instruction in the region is executed. The function of the library routes is to record several instances of the region execution, including input register values and output register values. When it is found that a region has an instance that can be reused, then all the instructions in the region can be skipped and hence improve the performance.

Keywords　Value profiling, Region, Reuse, Instrumentation library

1. 引言

指令之间的依赖使超标量计算机的处理器每个时钟周期只能执行平均1.7到2.1条指令[5]。同时指令的运行带有很大的重复性。指令重复是指一条静态指令多次动态执行得到的结果是一样的。通常的情况下，对一条计算指令，如果它的输入是相同的，那么其结果也是相同的。但是，也存在输入不同、但结果相同的情况；例如两个数做布尔运算，在很多种情况下，结果都是一样的。在本文中我们说一条指令被重复当且仅当这条指令的输入与输出全都重复。

导致指令会重复执行和程序的编制方法有关。程序执行时对输入数据进行一系列操作，但程序并不包含所有的动态操作指令，而是只包含动态运算的静态映像。这是因为：程序追求更加紧凑的格式；追求更好的通用性，即能对不同的数据进行同样的操作；程序的模块化也是重复的一个原因。那么，程序究竟在多大程度上具有重复性呢？

用 spec95的整数 benchmarks 作为测试程序，编译器采用 gcc (2.6.3版本)，编译选项是 -O3 -funroll-loops -finline-functions，指令集是类 MIPS-1的。为了使测试时间不至于过大，每个 benchmarks 选用1,000,000条静态指令，每条静态指令记录2000次执行动态情况。一条指令是重复的，当且仅当它的输入与输出是以前某次执行的重复。试验数据[4]表明，平均72%以上的动态指令都会被重复执行。

对程序进一步的研究发现，指令不但具有很大的重复性，而且其产生的值有规律可寻，是可以预测的。

2. 相关工作

针对程序动态执行发生的重复特性，可以对程序进行动态优化。主要包括预测和重用技术。

预测是指根据指令前几次的运行结果对下一次的运行结果进行预测，并利用这个预测值进行后面的运算，当这条指令计算完成后，用其结果与预测值进行比较，如果相同则提前做的指令有效，否则这些指令都必须重新执行。预测值与正确结果的比较是通过硬件来实现的，这样才能使花费的代价最小，取得性能上的提高。已有多种技术用来提高预测准确度。包括基于历史记录的预测、基于步长的预测、综合预测法[7]及上下文预测法[8]。

值预测可以在 cpu 得出结果之前利用预测值提前执行后面依赖于这个结果的指令，克服了指令之间的依赖，但这种方法有其限制：被预测的指令还要被执行并用所得结果和预测值进行比较，如果预测结果错误，则所有依赖于这个结果的指令都要被重新执行。如果能提前得出指令真正的运算结果而不只是对它进行预测，就可以不必执行这条指令并且依赖于它的指令也可以提前执行且不会因为误测而重新执行。例如，Sodani 和 Sohi 提出的指令级重用就是每条指令的输入与输出，当发现指令的当前出入与上次输入相同时，则跳过本次执行步骤。

指令级重用需要有硬件支持。指令操作数及结果被写到一个固定大小的重用缓冲区中，每当执行一条指令时，根据采用查询策略的不同，在取指或译码后根据 pc 或根据操作码及操作数查询重用缓冲区中是否有可重用结果。动态指令重用在两方面提高性能：第一，跳过指令某些执行阶段(例如发射，执行结果写回)，可以减少机器资源占用，进而减少资源冲突的可能性。第二，使依赖于这个结果的指令提前执行。指令级重用需要有硬件的支持。当然这种方法也会带来开销，包括查询缓冲区，更新机器状态及更新缓冲区。

试验表明可重用指令有时占很大的比例，但因为每条指令都需要在缓冲区中有其历史记录，而缓冲区不够大时需要

*)本课题得到国家自然科学基金(69933020,60103006)资助。张可新　硕士研究生，主要研究方向为控制流优化及基于 region 的动态重用技术。张兆庆　研究员，博士生导师，研究领域包括并行程序设计环境，自动并行化与并行可视化工具以及并行程序的正确性验证等。

采取替换策略更新缓冲区中指令条目，并且为每条指令查询是否可以重用都会带来额外开销，因此这种方法所能带来的性能提高是有限的。

一种改进策略，是把重用粒度提高到基本块。基本块级的重用也需要有硬件的支持。每个基本块用一个块历史缓冲区(block history buffer-BHB)来记录每个基本块的活跃输入和活跃输出及下一个基本块的地址。基本块是在运行时动态识别出来的。基本块的输入与输出也是在运行时动态检测的[3]。

对基本块进行重用可以起到一定的优化作用，但由于需要动态检测基本块及其输入和输出，因此造成了时间和空间上的额外开销。另外由于动态执行时缺乏对 bb 的基本信息，例如基本块的大小、输入和输入个数等，无法对不同的 BB 采取限制策略，因此不会有最好的优化效果。如果在编译时能用一定的规则对程序动态行为进行指导，可以克服上述不足。

3. 基于 region 的重用技术

基于 region 的重用技术是建立在 ORC(Open Research Compiler)编译器的基础上的。ORC 是中科院计算所同 Intel 公司的合作项目，为 Intel 的新一代64位芯片开发开放源代码的先进编译器。ORC 的实现目标是为研究机构和团体提供一个可靠稳定的、开放源代码的编译基础结构，方便其在 ORC 的基础上进行编译器和体系结构的研究；与当前其他 IA-64 编译器相比，具有更高的性能。

在 ORC 编译器的代码生成阶段包括有 edge profiling、region 构造、基于 region 的全局指令调度和寄存器分配、value profiling 技术等。

Profiling 技术是在程序的中间代码中插入插装库函数调用，根据需要可以设计不同的插装库函数完成不同的功能。edge profiling 统计控制流程图中各个基本块的执行频率和控制流边的概率，利用这些信息编译优化程序，value profiling 用于确定一条或一组指令的操作数的值是否有常量或半常量的属性，其他如 path profiling 是统计执行路径的执行频率[9]。

Region 是编译器为了便于控制优化的范围而形成的一组基本块的集合。用 region 作为全局指令调度和寄存器分配的单元，编译器可以根据需要设定不同启发式参数，形成不同范围和大小的 region，使优化达到较优的效果。

ORC 编译器对基于 region 的动态重用可提供三方面的支持：value profiling、动态重用 region 的形成、动态重用插装库的设计。

图1为带 value profiling 的基于 region 的 dynamic reuse 的处理流程图。

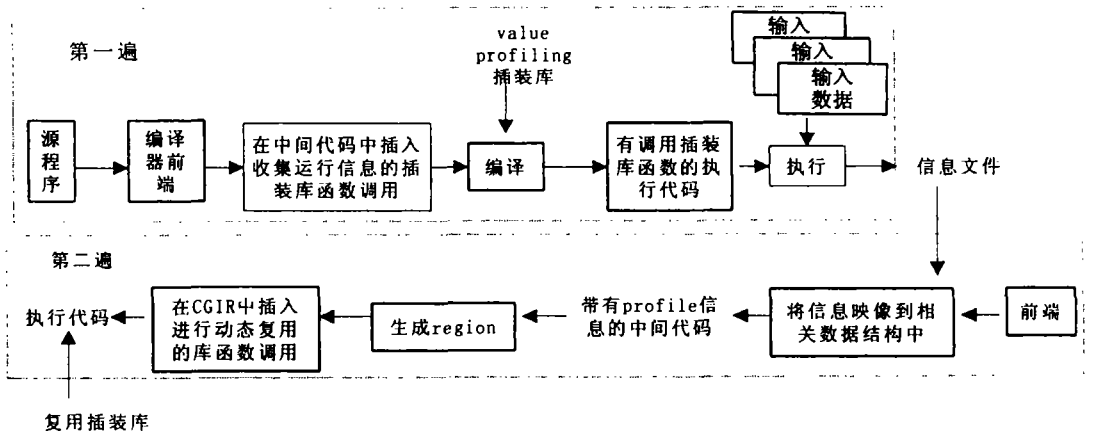


图1 基于 region 的 dynamic reuse 的处理流程

采用这种方法可以不需要额外的硬件支持，完全由软件来完成以上工作。

3.1 value profiling

profiler 向编译器产生的中间代码中插入一些指令(Instrumentation)，进行库函数调用，这些库函数记录并保存基本块在运行中的重用率。基本块的重用率是指基本块一次执行时的输入与输出与其上次执行时的输入输出完全相同的次数占基本块总的执行次数的比例。在第二次编译中，基于 profiling 的代码优化器获取 profiler 产生的基本块重用率反馈(feedback)信息，并利用此信息形成动态重用 region。

value profiling 实现的结构如下：

a)插装(instrumentation)：向程序的中间表示中插入调用插装库函数的指令，同时设置传递给插装库函数的参数。

b)value profiling 插装函数库(instrumentation library)：这部分函数负责统计收集程序运行时的信息；申请和释放保存信息所使用的存储空间；将信息保存到反馈文件中。

c)反馈(annotation)：读取反馈文件，获取程序运行时的重用频率信息，并将信息标注到中间数据结构中，供 region 形成时查询。

插装库函数完成的功能为：

1) 计算每个基本块执行的次数 c(bb)。

2)计算每个基本块重复执行的概率 p(bb)。

将插装库函数获得的信息，附在控制流图上，在形成 region 时，可以参看控制流图上的信息。

3.2 动态重用 region 及其生成算法

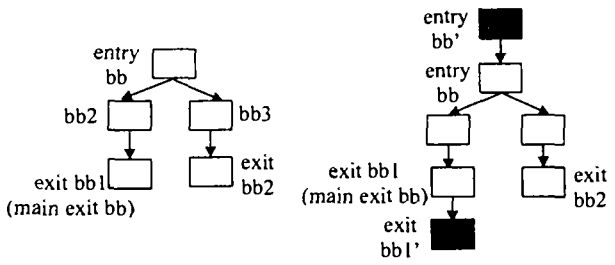
本文提出基于 region 的动态重用技术：利用 value profiling 的反馈信息，形成动态重用 region。动态重用 region 的形成过程与 ORC 中的已有的 region 形成框架类似，先生成多入口多出口的 region。有以下四个步骤，种子 BB 选择、后继选择、前驱选择和子路径扩充。不同的是需要的启发式参数，原 region 在选择前驱、后继及扩展旁路径时要考察基本块的执行频率，而在这里是要考察基本块的重用频率，这在 value profiling 时已经获得。然后再做尾复制，使其成为单入口多出口的 region。多入口多出口的动态重用 region 生成算法如下：

```
//Find a hot frequently reusable BB path
FormReusableHotPath(BB * bblist)
{
```

```
Find_BB_with_max_reusable_Frequency(bblist);
while !(y in R)&& SUCC_SUIT(x)
{
  R=R∪{y}
  x=y
  y=FindMostReuseFrequencySUCC(x);
}
/* select path of desirable predecessors */
y= FindMostReuseFrequencyPRED (x)
while size(R)<MAX_SIZE&& !(yin R)&& PRED_SUIT(x)
{
  R=R∪{y}
  x=y
  y=FindMostReuseFrequencyPRED (x)
}
}
//Extend the hot path
For each BB
{
  // Find out a BB y which are most frequently connected by BB set in
     R
  y=FindFrequentReusableConnectedBB(R);
  R=R∪{y}
}
```

其中函数 FindMostReuseFrequencySUCC(x)是查找一个 x 的后继中重用率最高的基本块 y，并且 y 的执行次数大于1，重用率大于 pR。

尾复制算法和原 region 类似，首先选择一个主出口基本块，计算对应这个主出口需要去掉的叶子基本块，然后进行尾复制去掉旁入口基本块，使其成为一个单入口多出口的 region。唯一的入口基本块被作为重用的开始点，主出口基本块作为重用的结束点。用软件进行重用比有硬件支持的重用有较大的时间开销：进行库函数调用时保存和恢复现场最多需要28条指令，重用开销还包括记录和比较 region 的 live-in 的值。所以要取得性能上的提高，需要 region 足够大，并对 region 的重用率也有一定的要求。Region 的重用率和基本块的重用率类似，是指 region 一次的执行的输入与输出完全与上次执行相同的次数占 region 总执行次数的比例。利用基本块的重复率可以近似估计 region 的重复率，也可以针对 region 再作一次 value profiling，这样可以得到精确的 region 的重用率。现在暂时利用基本块的重用率作为对 region 的重用率的估计值。关于 region 的大小，其重用率以及输入值个数的关系是如何影响性能的将在第四节中进行论述。



a)插装前的 region　　b)插装后的 region

图2　插装前后的动态重用 region

如上图，在每个动态重用 region 的入口基本块前和主出口基本块后分别插入一个基本块，进行插装库函数调用。利用数据流分析可以得出入口 bb 的 live-in 集合，live-in 指被 region 引用的在 region 外部定值的变量。因为 region 是单入口的，所以入口 bb 的 live-in 也就是整个 region 的 live-in。在入口插装的 bb 将这些值作为参数传递给库函数。通过数据流分析也可以得出 region 主出口 bb 的 live-out，live-out 指被 region 外部引用的在 region 内部定值的变量。在主出口后面插装的 bb 把这些值作为参数传递给库函数。如果执行是从非主出口出去的，则不作处理。

3.3　重用插装库

重用插装库是进行重用插装后的可执行代码运行时需要调用的库函数的集合。这些函数负责记录、更新程序运行时的动态信息以供程序后续执行部分作为重用参考。在每个 region 之前插入的 BB 中，进行库函数调用，库函数的实现的功能是：1)如果这个 region 是第一次被执行或者此次输入与已记录的输入值不同，把此次执行的输入值记录到 computation instance 的 INPUT 中，如下图所示。2)如果此次输入值与已记录值相同，且合法位被置，则跳过整个 region 到主出口之后插入的基本块。在每个 region 主出口之后插入的 BB 中，插入的库函数调用实现的功能是置合法位。

重用记录信息结构图如图3。

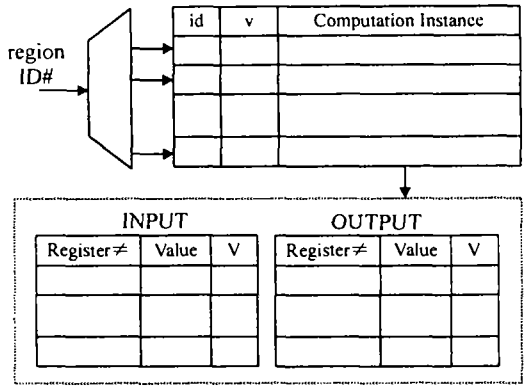


图3　插装库函数中用来存储 region 重用信息的结构图

每个 region 对应一个标志(id)。每个案例中包含有输入、输出寄存器号及其值。在执行时若 region 的输入与已记录输入值相同并且合法位被置位，那么这次计算是可以重用的。如果没有可以重用的计算，则重新记录本次输入和输出值。

4. 动态重用 region 约束条件分析

进行插装库函数调用的开销包括保存及恢复现场，对于第一次执行，需要记录 region 输入和输出值；对于非第一次执行，要比较本次输入值和已记录的输入值，如果不同还要重新记录输入和输出值；如果相同，用已记录输出值对本次输出赋值。保存及恢复现场的开销是固定的，包括保存和恢复全局数据指针寄存器、dedicated 寄存器等共28条指令。所以总的时间代价取决于输入及输出的个数。大部分 Region 的 live-in 的个数在几个到几百个之间，需要用相同数目的指令数来比较或保存这些值，要使重用获得整体性能上的提高，必须使其所花费的指令周期数少于正常执行所用的指令周期数，这里暂时以指令数目来代替指令周期数作为衡量 region 执行时间的单位。假设某 region 的重用率为 p，其 live-in 的数目为 i，此 region 包含的总的指令数目为 n，则利用动态重用技术执行程序时，此 region 被平均执行的指令数目为 $(28+i)*p+(28+2i+n)*(1-p)$，而不用重用技术时此 region 的执行指令数为 n，要使重用获得性能上的提高，需满足以下条件：

$$(28+i)*p+(28+2i+n)*(1-p)<n$$

即　$$ip+2i+28<2ip+np \qquad (1)$$

例如对 spec2000中 bzip2.c 有一个过程为：

```
void makeMaps ( void )
{
  Int32 i;
  nInUse = 0;
  for (i = 0; i < 256; i ++)
```

```
    if (inUse[i]){
        seqToUnseq[nInUse] = i;
        unseqToSeq[i] = nInUse;
        nInUse ++;
    }
}
```

该过程被 uncompressStream 过程间接调用多次，具有很高的重复率。用 ORC 编译器的-O2选项编译出的可执行代码需要执行的指令数是36+36×256+36=9261条。其中第一个36是循环体之前的指令数，第二个36是循环体中的指令数目，乘以循环的次数256，最后9条指令是循环体之后的指令数。如果把整个过程作为一个重用 region，则 i=256，n=9261，要使性能有提高，只需重用率 p>6%即可。

结论及其今后的工作 目前插装和参数化的 region 形成工作已经完成，现在正在进行的工作是如何获得更准确的 value profiling 信息用来指导 region 的生成。

动态优化是静态编译优化的一个重要补充，它可以利用静态优化不能捕捉到的信息对程序进行优化。但程序动态行为究竟存在着多大的可优化潜力，还有待于对程序的动态特性作更进一步的研究，相应地也需要在体系结构方面对这些工作加以支持。

参 考 文 献

1 Sodani A, Sohi G S. Dynamic Instruction Reuse. In: the Proc. of 24th annual Intl. symposium on computer architecture, 1997. 112～205

2 Molina C, Gonzlez A, Tubella J. Dynamic Removal of Redundant Computations. In: Proc. of the ACM Int. Conf. on Supercomputing, Rhodes(Greece), June 1999

3 Huang J, Lilja D J. Exploiting Basic Block Value Locality with Block Reuse. University of Minnesota Supercomputing Institute Research Report UMSI98/145, Aug. 1998

4 Sodani A, Sohi G S. An Empirical Analysis of Instruction Repetition. In: Proc. of Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 1998

5 Gonzlez A, Gonzlez J, Valero M. Virtual-Physical Registers. In: Proc. IEEE 4th. Int. Symp. on High-Performance Computer Architecture, 1998. 175～184

6 Kennedy K. optimizing compilers for modern architectures. an imprint of academic press

7 Wang K, Franklin M. Highly accurate data value prediction using hybrid predictors. In: 30th Annual Intl. Symposium on Microarchitecture, Dec. 1997

8 Sazeides Y, Smith J. The Predictability of Data Values. In: 30th Intl. Symposium on Microarchitecture, Dec. 1997

9 Calder B, Feller P, Eustace A. Value profiling. In: Proc. of the 30th Annual IEEE/ACM Intl. Symposium on Microarchitecture (MICRO-97), Dec. 1997. 259～269

第九届全国 Petri 网学术年会征文通知

由中国计算机学会 Petri 网专业委员会主办的第九届 Petri 网学术年会将于 2003 年 8 月下旬在杭州召开（杭州电子工业学院承办）。会议将对 Petri 网理论及其在各个领域中的应用开展广泛深入的讨论。现发出征文通知。

1. 征文范围

1) Petri 网理论研究；

2) Petri 网工具开发；

3) Petri 网在相关领域的应用研究；

4) 相关的并发模型及仿真技术研究。

2. 征文要求

1) 凡在正式刊物或其它学术会议上发表过的论文不再征用；

2) 投送论文无论录用与否概不退稿，请作者自留底稿；

3) 会议论文集以《仿真学报》增刊出版（部分优秀论文将推荐到正刊），被录用的论文将要求按《仿真学报》要求的论文格式编辑打印，寄出软盘，并交纳版面费；

4) 投送论文请写明作者姓名、通讯地址（包括邮政编码），以便联系。

3. 重要日期（邮寄以邮戳日期为准）

征文截稿日期：2003 年 3 月 31 日

发出录用或修改通知日期：2003 年 4 月 30 日

编辑打印好的稿件（软盘）返回日期：2003 年 5 月 31 日

4. 论文请寄：北京航空航天大学软件学院 邮政编码：100083 联系人：金梅、赵小苹

电话：010－82316078，82314660 Email：petri@buaa.edu.cn 传 真：010－82332496

中国计算机学会 Petri 网专业委员会

2002 年 12 月