

# Imagine 流处理器上流的优化组织方法

杨学军<sup>1)</sup> 曾丽芳<sup>2)</sup> 邓 宇<sup>1)</sup> 唐玉华<sup>1)</sup>

<sup>1)</sup>(国防科学技术大学计算机学院并行与分布处理国家重点实验室 长沙 410073)

<sup>2)</sup>(江南遥感应用研究所 上海 200072)

**摘 要** 流应用的特点以及传统处理器在处理流应用上的不足,使得支持数据并行的流处理器的设计成为当前体系结构研究领域的一个热点.文中针对 Imagine 流处理器体系结构的特点,提出了流分割和流压缩两种流的优化组织方法.模拟结果表明,流分割和流压缩使得流应用程序能充分利用 Imagine 的并行结构、流水结构和多级带宽存储结构,从而减少程序的执行时间.

**关键词** Imagine 流处理器;流应用;流优化;流分割;流压缩

**中图法分类号** TP316

## The Optimization Approaches of Organizing Streams on Imagine Processor

YANG Xue-Jun<sup>1)</sup> ZENG Li-Fang<sup>2)</sup> DENG Yu<sup>1)</sup> TANG Yu-Hua<sup>1)</sup>

<sup>1)</sup>(National Key Laboratory of Parallel and Distributed Processing, School of Computer Science,

National University of Defense Technology, Changsha 410073)

<sup>2)</sup>(Jiangnan Remote Sensing Institute, Shanghai 200072)

**Abstract** Due to the characteristics of stream applications and the insufficiency of conventional processors when running stream programs, stream processors which support data-level parallelism become the research hotspot. This paper presents two means, stream partition (SP) and stream compression (SC), to optimize streams on Imagine. The results of simulation show that SP and SC can make stream applications take full advantage of the parallel clusters, pipelines and three-level memory hierarchy of the Imagine processor, and then reduce the execution time of stream programs.

**Keywords** imagine stream processor; stream application; stream optimization; stream partition; stream compression

## 1 引 言

随着数字技术的普及,一种新型的计算形式——流计算,逐渐出现在各个领域中.斯坦福大学的研究者将流(stream)<sup>①</sup>定义为不间断的、连续的、

移动的数据队列,队列长度可以是定长或不定长的,流元素的组成可以复杂或简单.例如一个三角形的顶点或者是一幅图像的 $8 \times 8$ 的像素区域,或者是一个简单的整数等都可以构成流元素.将要处理的数据组织成流,并对流进行操作的应用,称之为流应用.流应用正成为当前微处理器的主要负载之一.流

收稿日期:2006-08-09;最终修改稿收到日期:2008-05-30.本课题得到国家自然科学基金(60621003,60633050)资助.杨学军,男,1963年生,教授,博士生导师,主要研究领域为并行计算机系统结构、并行操作系统和并行编译.曾丽芳,女,1970年生,博士,副教授,主要研究方向为编译优化和并行计算. E-mail: zlf0000@163.com.邓宇,男,1977年生,博士研究生,研究方向为计算机体系结构.唐玉华,女,1962年生,教授,主要研究领域为计算机体系结构、计算机网络和大规模集成电路等.

① The Imagine Project. Stanford University. <http://cva.stanford.edu/imagine/>

应用具有很高的计算密集性、大量的数据并行性和很少的时间局部性,传统的通用体系结构无法很好地适应流应用的这些特征,因此,斯坦福大学的研究人员基于流处理的理念,开发研制了 Imagine 流处理器<sup>[1]</sup>,包括流体系结构、流编程模型及配套的软件开发工具.作为新兴流体系结构的代表,Imagine 流体系结构提供并行 Cluster 和三级层次带宽,试图解决性能扩展、功耗和带宽瓶颈问题. Imagine 流编程模型则能够揭示出流应用的并发性和生产者-消费者局部性.

流应用将要处理的数据以记录的形式组织成流,流记录中的数据同时被访问,并且在内存中相邻,因而,流具有良好的 Cache 空间局部性.那么,如何优化组织具有良好空间局部性的流,使得流程序能充分利用流处理器的并行和流水结构,从而提高流程序的性能呢?本文针对 Imagine 流体系结构的特点,利用 Isim 模拟器分析了流的不同组织方式对流应用整体性能的影响,提出了流分割和流压缩的优化方法.模拟结果表明,流分割方法能够有效利用 Imagine 的并行功能部件,从而提高 kernel 程序的运行性能;流压缩则能减少主机到 Imagine 的数据传输时间.

本文首先介绍 Imagine 的体系结构和编程模式;然后阐明流的两种优化组织方法,并利用 Isim 模拟器给出了具体的模拟性能数据;最后将我们的工作和相关工作进行了比较.

## 2 Imagine 体系结构及编程模型

Imagine 是斯坦福大学开发的一种可编程流处理器,下面依次介绍其体系结构和编程模型.

### 2.1 Imagine 体系结构

Imagine 可以看作专门负责流处理的协处理器,是一个 32 位的体系结构,支持 16 位和 8 位数据

(峰值性能分别为 32 位数据的 2 倍和 4 倍),其体系结构如图 1 所示,包括以下几大部件:主机接口、流控制器、流存储系统、微控制器、8 个运算簇组(Cluster)、网络接口、流寄存器文件(SRF)和局部寄存器文件(LRF).每个 Cluster 包括 8 个功能单元:3 个加法器,2 个乘法器,1 个除法/开方部件,1 个便笺寄存器文件和 1 个簇内通信单元. Cluster 中每个功能单元的输入都是由单独的 LRF 提供.微控制器流出 VLIW 指令到 8 个 Cluster 中,并控制它们以 SIMD 方式执行(与传统向量体系结构的区别:向量体系结构每拍只发送一条指令,依赖于每条向量指令中的并行性来获得高性能;Imagine 则能发送 VLIW 指令,实现了 kernel 内指令级并行).

Imagine 提供了三级带宽层次<sup>[2]</sup>:片外存储器带宽(2.67GB/s),SRF 带宽(32GB/s),Cluster 内 LRF 在运算单元间传输带宽(544GB/s).三级存储带宽层次是流体系结构的关键创新之一,图 2 给出了三级层次存储的连接关系:内存传输将整个流从片外 DRAM 装入 SRF 或将整个流从 SRF 保存到

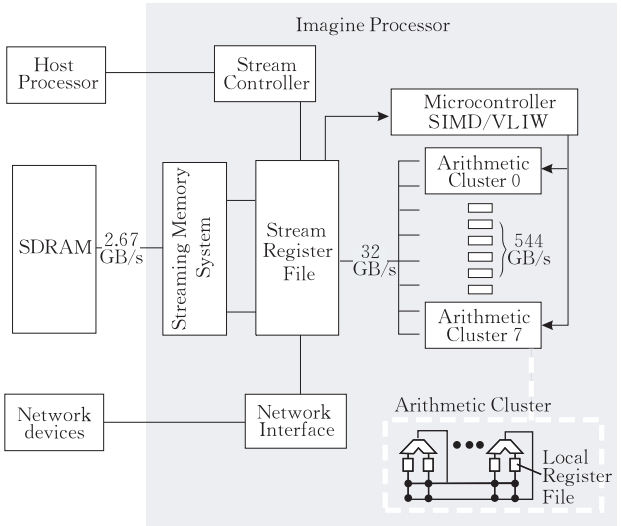


图 1 Imagine 流处理器的体系结构<sup>[3]</sup>

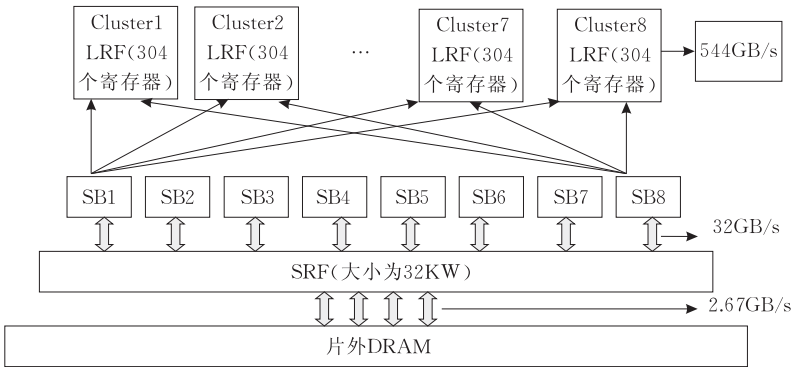


图 2 Imagine 的三级层次存储连接关系

外部内存,可以同时发生多个内存传输. SRF 通过流缓冲器(SB)和 Cluster 中的局部寄存器交互数据, LRF 直接为 Cluster 中的功能部件提供数据. 运行过程中产生的临时数据都存储在 LRF 中, 只有输出流需要回送到 SRF 中(不必传回到片外存储器中), 某些全局数据只有在需要时才存储在片外存储器中. 这种显式的三层存储结构开发了多级数据局部性<sup>[4]</sup>, 形成了一个有效的通信结构, 以维持算术逻辑单元的供数速度.

2.2 Imagine 编程模型

Imagine 采用了一种新的层次化的流编程模型. 此模型将数据组织成流, 即同种类型的数据记录构成的序列; 将计算表示成核心(kernel), 每个 kernel 是一个过程, 它对输入流进行处理, 产生输出流. Imagine 在编程时分为两级: 流级(StreamC 编程)和核心级(KernelC 编程), 这两级分别在主机和 Imagine 上运行<sup>[5-6]</sup>.

流级程序描述整个应用的执行过程, 包括流的定义、流的载入 SRF、存回片外 SDRAM、流在 kernel 间的流动过程即执行顺序. 核心级程序具体描述单个 kernel 对流执行操作的过程.

Imagine 流编程模型的目标是能够很好地开发数据并行性. 它支持的 SIMD 方式要求简单的控制机制, 因此, 用于 kernel 中的主要控制结构是循环结构(loop). 典型的情况是, 对输入流的每个元素进行循环, 对它们施加相同的操作. Imagine 流编程模型的另一目标是实现快速的 kernel 执行, 为此, 它要求 kernel 只能在本地的数据上(LRF)进行操作, 不能进行任何的存储访问. 流的访存操作由流级程序完成.

在流的程序设计模型中, 流的访存和通讯对程序员是可见的, 主要是为了减少对应的硬件结构的复杂性, 以尽量减少长线延迟的影响. 在流级程序设计时程序员需要考虑流的调入和组织, 会引出一些问题比如流的组织、kernel 的划分等.

3 流的优化组织方法

流应用将要处理的数据以记录的形式组织成流, 流记录依次被 kernel 处理, 记录中的每个分量总是被 kernel 同时访问, 所以, 对流应用而言, 数据具有良好的空间局部性. 因此, 对流进行优化的关键是如何优化组织数据, 使得流应用能充分利用 Imagine 体系结构的并行 Cluster 和三级存储结构.

我们以一个具备流应用所有特征的小例子(例 1)和快速傅立叶变换(例 2)为例, 对比分析了流的多种组织方式对应用各部分性能的影响, 提出了流分割和流压缩两种优化方法. 下面分别介绍这两种方法.

3.1 流分割优化方法

3.1.1 流分割优化思想

在流应用中, 被 kernel 处理的相关数据以记录的方式组织成流, 流中的记录依次流入 Cluster 中被 kernel 处理, 记录中的每个分量总是被 kernel 同时访问. 例如, 在图 3 所示的程序(例 1)中, 记录 Com 中包含 8 个整型分量和 5 个浮点分量. kernel 程序中的 *in*  $\gg$  *i* 语句将一个流记录读入 Cluster 中, 然后对此记录中的每个分量进行操作, 但编译处理时, 需要 13 条指令才能从 SRF 中读入整个记录, 每条指令读入记录的一个分量. 当然, 由于 kernel 指令是 VLIW, 所以, 各个分量的读入可以和其它不相关的操作并行进行. 包含 13 个分量的单条流的 SRF 利用情况如图 4 所示: 图中示意了各个时钟周期内 Imagine 资源使用情况, 其中, 横轴上标识了可用的各个资源, 纵轴表示各个时钟周期. 从图中可看出, 13 个分量的读入是串行的, 只用到了 8 个流缓冲器中的一个, 但是, 流记录的读入可与其它操作并行执行.

```
record Com{
    int a1,a2,a3,a4,a5,a6; float m1,m2,m3,m4,m5; int d1;
    int d2;
};
kernel test_kc(istream<Com>in1,istream<Com>in2,
    ostream<Com>out) //kernel 程序
{
    loop_stream (in1) pipeline (1) {
        Com i1,o;
        in1>>i1; in2>>i2; //get input records
        o.a1=i1.a1+i2.a1; o.a2=i2.a2+i1.a2; o.a3=i1.a3+i2.a3;
        o.a4=i1.a4+i2.a4;
        o.a5=i1.a5+i2.a5; o.a6=i1.a6+i2.a6; o.d1=i1.d1-i2.d1;
        o.d2=i1.d2+i2.d2;
        o.m1=i1.m1 * i2.m1; o.m2=i1.m2 * i2.m2;
        o.m3=i1.m3 * i2.m3;
        o.m4=i1.m4 * i2.m4; o.m5=i1.m5 * i2.m5;
        out<<o; //write output }
    }
}
streamprog test(String args) //stream 程序
{
    //定义两个长度为 32 个记录的输入流和一个输出流
    im_stream<Com>s1(32); im_stream<Com>s2(32);
    im_stream<Com>temp(32);
    streamLoadFile("test1.in","txt","",s1);
    streamLoadFile("test2.in","txt","",s2);
    swp_kc(s1, s2, temp);
    streamSaveFile("swp.out","txt","d", o1);
}
```

图 3 例 1:stream 和 kernel 源码

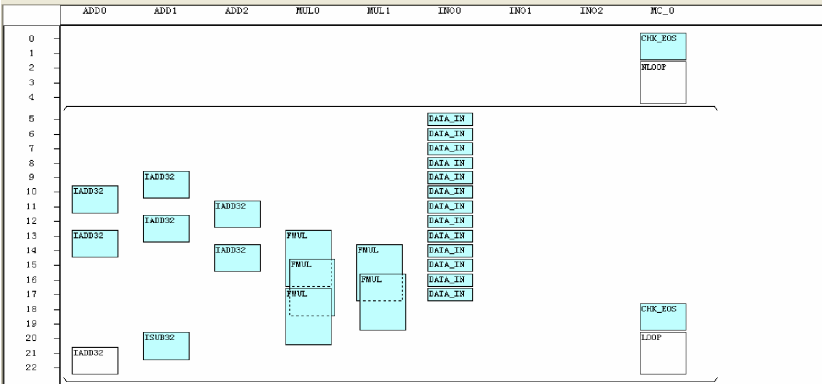


图 4 单条流时的 SRF 利用情况

由于流处理器在支持数据并行的同时强调存储带宽,比如,Imagine 流处理器的 SRF 不仅支持每个时钟周期向每个 Cluster 提供 1 个字(共 8 个字),而且还可同时支持 8 条流进出每个 Cluster. 如果将 Com 记录分割成两个记录 Com1 和 Com2:

```
record Com1{
    int a1,a2,a3,a4,a5,a6; int d1; int d2;
};
record Com2{
    float m1,m2,m3,m4,m5;
```

};

然后,将所有输入流按记录分量划分为两条流  $in1$  和  $in2$ ,在 kernel 中用  $in1 \gg i1$  和  $in2 \gg i2$  两条语句分别读入两个流记录. 由于 Imagine 可同时支持 8 条流进出每个 Cluster,因而,语句  $in1 \gg i1$  和  $in2 \gg i2$  可以并行执行. 此时,仅需要 8 条指令就可以将所有 13 个分量读入 Cluster 中. 两条输入流时 SRF 的利用情况如图 5 所示.

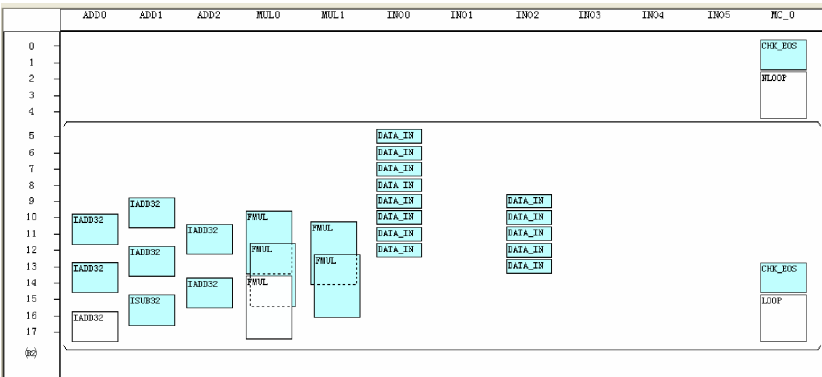


图 5 两条流时的 SRF 利用情况

由此可见,对流记录进行分割可以有效利用 SRF 的高带宽,从而支持 Cluster 内的并行计算. 我们将通过对流进行分割来增加 kernel 程序中流的数目,从而支持 Cluster 内并行计算的优化方法称为流分割优化方法.

3. 1. 2 流的分割方法

流可以有如下两种分割方法. 对不同的程序,可以使用某种分割方法,也可以综合使用两种分割方法.

(1) 流记录分割,即如上节所述的将大记录中的分量进行分割,形成不同的流,每条流的长度和分割前相同. 假设原始大记录包含  $N$  个分量,将大记录分割为  $M$  个小记录(在 Imagine 中,  $M \leq 8$ ),每个小记录中包含的分量个数分别为  $S_1, S_2, \dots, S_m$  ( $S_1 +$

$S_2 + \dots + S_m = N$ ). 为了充分利用 SRF,在满足程序约束的条件下,分割流记录时应该使得对任意的  $k$ ,

有  $\sqrt{\sum_{i=1}^m (S_i - S_k)^2}$  最小. 满足此条件的分割就不会

造成某个小记录因其分量远远大于其它记录的分量而再次成为性能瓶颈. 流记录分割方法能增加 kernel 程序的流数目,并且能减轻编译相关性分析的复杂度,从而能很好地发挥 Cluster 的并行性能. 此方法的缺点:多条流分别从主机传入 Imagine 中会增加额外的启动延迟开销.

(2) 流截断分割. 此方法仍然将 kernel 所访问的所有数据融合到一个大记录中,stream 程序利用流索引将流分成不同的段,每段对应 kernel 程序的

一个输入流. 例如, 将一个长度为 64 个记录的流  $s1$  分成  $s1(0:31)$  和  $s1(32:63)$  两段, 作为两条独立的流流入 Cluster 中. 这种方法不仅能增加 kernel 程序的流数目, 而且能缩短输入流的长度. 但缺点是有的流不适合进行这种分割, 并且, 流的索引还可能会导致 SRF 与 Imagine 内存之间的额外通信开销.

3.2 流压缩方法

我们在用 Isim 模拟器验证 3.1 节中流分割优化效果的过程中, 发现流应用整体性能受主机到 SRF 的数据传输时间的限制. 比如, 例 1 在模拟过程中得到的性能数据可以明显说明(如表 1 所示).

表 1 例 1 的模拟性能数据

执行阶段	所花费的周期数
将数据从主机文件装入 SRF	8655
将 kernel 微码从 SRF 装入微控制器中	507
kernel 执行	93

假设主机到 SRF 的传输通道的带宽为  $k\text{GB/s}$ , 要传输的数据大小为  $x\text{GB}$ , 则传送的时间为  $(x/k+t_s)$ , 其中,  $t_s$  为启动延迟. 显然, 传输时间与数据量的大小成正比, 与带宽成反比. 减少主机到 SRF 的数据传输时间的两种办法就是提高带宽和压缩流. 我们在编程过程中可能做到的优化就是压缩流. 对应用程序中的某些数据, 由于其特殊性, 可以通过较小的输入流来计算较大的使用数据. 下面, 以图 6 中例 2 所示的快速傅立叶变换中的旋转因子为例来说明这一优化思想. 例 2 是二维快速傅立叶变换的 C 语言代码, 在移植到 Imagine 上时, 我们将每个阶段的循环组织成 kernel, 这样, 对长度为  $N$  的流, stream 程序要调用  $\log_2 N$  次 kernel.

```
void fft(int n, double * A_re, double * A_im, double * W_re,
double * W_im)
{
    for (m=n; m>=2; m=m>>1){ /* 对一个长度为 N 的输入
流, 分 log2N 个阶段 */
        mt=m>>1;
        for (g=0, k=0; g<n; g+=m, k++){ /* 在每个阶段, 数
组被分成 n/m 段分别计算 */
            w_re=W_re[k]; w_im=W_im[k];
            for (b=g; b<(g+mt); b++) /* 对数组的某一段进行
计算 */
                t_re=w_re* A_re[b+mt]-w_im* A_im[b+mt];
                t_im=w_re* A_im[b+mt]+w_im* A_re[b+mt];
                u_re=A_re[b]; u_im=A_im[b];
                A_re[b]=u_re+t_re; A_im[b]=u_im+t_im;
                A_re[b+mt]=u_re-t_re; A_im[b+mt]=u_im-
t_im; }}}
}
```

图 6 例 2:fft 的 C 源代码

从图 6 我们可以看出: 对二维快速傅立叶变换, 若输入流有  $N$  个元素, 则 kernel 程序需要  $N/2$  个

旋转因子, 并且, 每个阶段所需的旋转因子  $w$  的数量是不同的, 每个阶段需要用到的  $w_{re}$  和  $w_{im}$  的情况如下:

- 第 1 阶段:  $w^0$ ;
- 第 2 阶段:  $w^0, w^1$ ;
- 第 3 阶段:  $w^0, w^1, w^2$ ;
- ...
- 第  $\log_2 N$  阶段:  $w^0, w^1, w^2, \dots, w^{n/2}$ .

在 Imagine 上实现上述代码时, 我们将对每一阶段的计算划分成一个 kernel 程序, 则 stream 程序中需要循环调用此 kernel 程序  $\log_2 N$  次, 每一次调用对应不同的输入流和  $w$  值. 针对  $w$  数组的特殊性, 我们不是在每次调用 kernel 时装入一个不同的  $w$  流, 而是使用一个相同的  $w$  流, 并且  $w$  流的长度不需要  $n/2$  个元素. 我们可以将  $w$  划分为 3 类:

第 1 类(twiddle):  $w^0, w^{16}, w^{32}, \dots, w^m$ , 其中,  $m=N/2-16$ , 共  $N/32$  个元素.

第 2 类(rotate):  $w^0, w^1, w^2, w^3, w^4, w^5, w^6, w^7$ , 共 8 个元素.

第 3 类(interp):  $w^8$ , 共 1 个元素.

其它的数据可以很快由上面这几类数据算出, 如  $twiddle \times rotate$  可以算出第  $0 \sim 7, 16 \sim 23, \dots, w^m - w^{(N/2-9)}$  个元素,  $twiddle \times interp \times rotate$  可以算出  $8 \sim 15, 24 \sim 31, \dots, w^{(N/2-8)} \sim w^{n/2}$  个元素. 所以, 只要装入  $(N/32+9) \times 8$  个元素(每个 Cluster 上一份), 就可以计算出其它  $(N/4-72)$  个元素. 此优化方法即不给 kernel 程序带来很多的计算量, 又可以少装入  $(N/4-72)$  个数据. 这不仅大大减少了主机到 SRF 所传递的数据量, 而且, 由于装入的  $w$  较小, 可以充分利用 Imagine 的 LRF 来存储这些  $w$  的元素, 从而减少 Cluster 与 SRF 的交互次数.

4 性能测试

我们用斯坦福大学提供的 ISim 模拟器来进行优化性能测试. 本章首先介绍 ISim 模拟器, 然后给出具体的性能数据.

4.1 ISim 模拟器

ISim 模拟器是 Imagine 流处理器的软件模拟平台. 开发包中还包括 IScd、IStream、IDebug 和 SchedViz 等功能模块. ISim 是一个时钟精确的模拟器, 可以运行 Imagine 应用程序, 并准确获得 Imagine 关于 Cluster 与功能单元、微控制器、SRF 和存储系统等的性能结果. IScd 是 KernelC 调度器, 用

来编译 Imagine 处理器上执行的 kernel 程序. IStream 是 StreamC 编译器, 用来编译流级程序. IStream 负责流在存储器和 SRF 的分配, 并采用 strip-mining 和 double-buffering 技术处理较长的流. IDebug 是一个交互型功能级模拟器, 用于在开发阶段对 Imagine 应用程序进行调试. SchedViz 是一种调度可视化工具, 能够监控 Imagine 是如何按时间顺序使用不同资源的.

4.2 性能模拟结果

4.2.1 流分割优化性能测试

我们在 ISim 上模拟了两个例子的不同流组织方式对 kernel 的影响, 统计了如下 3 个与 kernel 相关的性能数据:

- (a) 生成的 kernel VLIW 指令数.
- (b) 将 kernel 微代码从 SRF 装入微控制器 (MPC) 的时间.
- (c) Kernel 一次运行所花的时间.

由于例 1 非常简单, 我们可以综合利用流的两种分割方法来组合不同的流, 所以, 对例 1 做了 2 条输入流 1 条输出流、4 条输入流 1 条输出流、4 条输入流 2 条输出流和 6 条输入流 2 条输出流 4 种流组织方式下的性能测试, 在上述 4 种情况下, 从主机装入 Imagine 的流的总长度均为 32 个记录.

对于例 2, 由于算法的限制, 我们做了如下两种情况下的测试:

(a) 将复数的实部和虚部融合成一个记录, 1024 个记录形成一条流, 然后在 stream 程序中利用流截断分割法将流分成前后两部分, 作为两个长度为 512 个记录的流分别流入 kernel 中, 加上旋转因子独立成一条流, 共有 3 条输入流 1 条输出流.

(b) 保持上述截断分割思想, 但用流记录分割法将输入数据的实部和虚部分开成两条独立的流, 从而形成了 6 条输入流和 2 条输出流.

流的不同组织方式对 kernel 程序运行时间的影响见表 2, 对 kernel 指令数及 kernel 代码装入时间的影响见图 7. 我们还利用 ISim 工具包中的 SchedViz 调度可视化工具, 监控到例 1 在 4 种不同的流组织方式下 Imagine 的资源使用情况, 见图 8~图 11. 从图 8 可知, 例 1 中的数据组织成 2 输入流 1 输出流时, 用到 3 个输入输出接口 (INO) 资源, 程序模拟执行需要 22 个时钟周期; 图 9 显示, 例 1 中的数据组织成 4 输入流 1 输出流时, 用到 5 个输入输出接口 (INO) 资源, 程序模拟执行需要 19 个时钟周期; 例 1 中的数据组织成 4 输入流 2 输出流时, 用到 6 个输入输出接口 (INO) 资源, 程序模拟执行需要 17 个时钟周期, 如图 10 所示; 例 1 中的数据组织成 6 输入流 2 输出流时, Imagine Cluster 与 SRF 接口的 8 个 SB 全部被利用, 并且能并行工作, 程序模拟执行仅需 13 个时钟周期, 如图 11 所示. 由此可知, 流分割优化能显著提高 Imagine 资源的利用率.

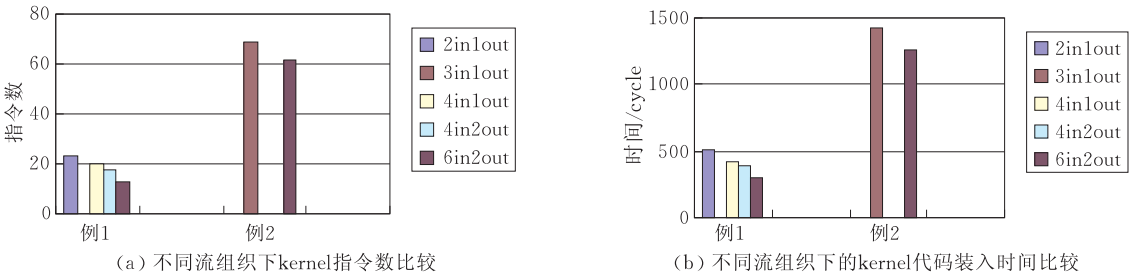


图 7 流的不同组织方式对 kernel 指令数及 kernel 代码装入时间的影响比较

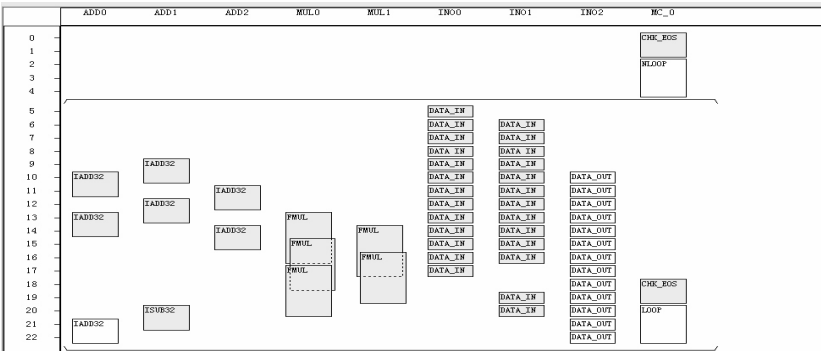


图 8 将例 1 中的数据组织成 2 输入流 1 输出流时 Imagine 资源使用情况





## 5 相关工作比较

ISim 工具包中的 profile 编译器 istream<sup>[5]</sup> 支持半自动的 stripmining 优化和软件流水优化. Stripmining 优化的主要思想是将大于 SRF 容量的初始输入流截断成若干个段,使得 kernel 产生的中间流能保存在 SRF 中,从而充分实现生产者-消费者局部性. 我们提出的流分割是将经 stripmining 优化后的流进行分割,增加 kernel 的输入输出流数目,从而提高 kernel 程序的运行效率. 将一条流分割为两条流,相当于为这条流额外分配了一个空闲的访问接口,从而增加了对原来这条流的访问速度. 由于流分割不会改变输入流的总大小,所以对 stripmining 不会有任何影响. 传统的软件流水优化则是将循环划分成多个段,使得一个迭代的一段与另一迭代的另一段重叠执行. 当一个 kernel 产生的输出流被存回内存中,再以不同的访问模式作为第 2 个 kernel 的输入时,第 2 个 kernel 必须等到前一个 kernel 的输出流重新组织完成后才能执行. 此时,软件流水可以将此内存访问与第 3 个不存在数据相关性的 kernel 执行重叠起来,从而隐藏内存访问延迟<sup>[6]</sup>. Sermulins<sup>[7]</sup> 等基于 Streamit 流编译器研究了提高程序的 cache 利用率的优化,提出了重复执行 actor 以提高指令 cache 局部性、将相邻 actor 进行合并以提高数据 cache 局部性及标量替换三种 cache 局部性优化方法. 这里的 actor 相当于本文中的 kernel. 他们的方法侧重于对 kernel 进行优化来提高数据局部性,我们的流分割和流压缩方法主要考虑利用空闲资源加速访存并根据应用的特征用计算换取访存. Griem<sup>[3]</sup> 等研究并实现了关于最短路径的传递闭包在 Imagine 流处理器上的优化实现方法. 他们的优化侧重于根据程序访存的特点来重用数据,减少冗余的访存,提高计算密集性. 在他们的优化基础上,应用我们的方法还可进一步提高性能.

## 6 结束语

本文通过分析两个例子在不同流组织方式下的性能,提出了流截断分割和流记录分割方法来适当增加流入 kernel 程序的流数目,从而充分利用 Imagine 的存储特征和流水功能,提高 kernel 程序的

运行效率的优化方法,并针对模拟运行中存在的性能瓶颈,提出了利用流压缩来减少主机到 Imagine SRF 的数据传输时间的优化方法. 模拟结果表明,这两种方法能提高 Imagine 的资源利用率,改善流应用程序的运行效率.

Imagine 流体系结构的并行和三级存储特征给流编译器及应用编程带来了许多优化空间. 课题组近年来围绕流处理器的编程编译还进行了很多研究工作,主要有:1)在编译技术方面,研究了从循环和向量到程序的自动转换问题,以减轻程序员的负担;研究了基于循环变换的流变量复用技术、基于 D&C 矩阵的优化流变换技术以及基于重用的双缓冲流技术,这些编译优化技术极大的提高了程序在流处理器上的执行效率. 2)应用以上编译优化技术,对更广泛的科学计算程序进行了实验. 这些程序包括 NPB 测试程序集、SPEC2000 的部分程序以及 Laplace 变换、线性方程求解器等程序. 这些实验的结果表明,流处理器在科学计算领域有着广泛的应用前景,同时也展示了以上编译优化技术的有效性.

## 参 考 文 献

- [1] Kapasi Ujval J, Dally William J et al. The imagine stream processor//Proceedings of the IEEE International Conference on Computer Design. Freiburg, Germany, 2002; 282-288
- [2] Rixner S, Dally W J et al. A bandwidth-efficient architecture for media processing//Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture. Dallas, TX, 1998; 3-13
- [3] Griem G, Oliker L. Transitive closure on the imagine stream processor. Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, USA; Paper LBNL-54903, 2003
- [4] Kapasi U J, Rixner S et al. Programmable stream processor. IEEE Computer, 2003, 36(8): 54-62
- [5] Das A, Mattson P et al. Imagine Programming System User's Guide 2.0. Stanford University, Stanford, CA, USA, June, 2004
- [6] Jung Ho Ahn, Dally W J et al. Evaluating the imagine stream architecture//Proceedings of the 31st Annual International Symposium on Computer Architecture. Munchen, Germany, 2004; 14-25
- [7] Sermulins J, Thies W, Rabbah R, Amarasinghe S. Cache aware optimization of stream programs//Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems. Chicago, Illinois, USA, 2005; 115-126





**YANG Xue-Jun**, born in 1963, professor, Ph. D. supervisor. His main research interests include parallel computer architecture, parallel operating system and parallel compilation.

**ZENG Li-Fang**, born in 1970, Ph. D. , associate professor. Her main research interests include compiling optimization and parallel computing.

**DENG Yu**, born in 1977, Ph. D. candidate. His main research interests focus on computer architecture.

**TANG Yu-Hua**, born in 1962, professor. Her main research interests include computer architecture, computer networks and large scale integrated circuit.

**Background**

This paper addresses the problem of optimizing the performance of programs on the stream processor, which is an emerging architecture that addresses the problem of memory wall. As the stream processor architecture exposes more fine-granularity parallel mechanisms and memory operations to the programmers, the performance is heavily determined by the programmers. The previous works on this domain mainly focus on the organization of streams from arrays and the optimizations of kernels. The most testing programs are from media applications.

This paper provides a method to improve the utility of the hardware of the stream processor, i.e. the usage of stream buffers from another point of view. Further more, it provides a method to reduce the memory transfers according to the characteristics of the applications. The two methods are effective to both media and scientific applications.

This work is supported by NSFC projects: Key Technologies of Peta-flops High Performance Computing

(60621003) and High Productive Parallel Computer Architecture (60633050). High performance computing is a foundational, foresighted and strategic crucial technology for solving significant application problem of nation and national defense construction. These projects revolve the long term demand of high performance computing from the national significant application domain and hold the significant transformation opportunity of the international high performance computing domain technology route.

In this direction, this research group has carried out a 64-bit stream processor FT64, which is the first implementation for scientific applications. A lot of papers have been presented in ISCA07, ICPP07, ISPA06&07, ACSAC06&07 and so on.

This paper explores the optimization methods on the stream processor through programming and gives the indications for the design of compiler.