ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.9, September 2009, pp.2320−2331
doi: 10.3724/SP.J.1001.2009.03406
© by *Institute of Software, the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

流处理器上基于参数模型的长流分段技术*

杜 静[1+], 敖富江[1], 唐 滔[2], 杨学军[2]

[1](63880 部队 博士后科研工作站,河南 洛阳 471003)

[2](国防科学技术大学 计算机学院,湖南 长沙 410073)

Parameter Model Based Strip-Mining Technique on the Stream Processor

DU Jing[1+], AO Fu-Jiang[1], TANG Tao[2], YANG Xue-Jun[2]

[1](Postdoctoral Scientific Research Workstation, Unit 63880 of PLA, Luoyang 471003, China)

[2] (School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: jingdu@nudt.edu.cn

Du J, Ao FJ, Tang T, Yang XJ. Parameter model based strip-mining technique on the stream processor. *Journal of Software*, 2009,20(9):2320−2331. http://www.jos.org.cn/1000-9825/3406.htm

Abstract: The Strip-Mining technique is significant for improving SRF bandwidth utilization on the stream processor. It is critical to quantify the program execution time influenced by the strip size for achieving optimal strip size. In order to achieve the theoretical optimal strip size, this paper proposes an optimal strip-mining technique based on a parameter model to minimize the execution time. Firstly, the paper builds a prefetching and reusing optimizations guided parameter model that characterizes the effect of strip size on program behavios. Secondly, based on the model analysis, this paper explores the optimal strip size selection approaches to the computation intensive programs and memory intensive programs respectively. Finally, an optimal strip-mining technique for any program is proposed. The experimental results show that our strip-mining technique can effectively hide and avoid the memory access latency, so as to exploit the powerful computation ability of stream processor.

Key words: strip-mining; imagine; optimal strip size; SRF locality; overlap between memory access and computation

摘 要: 长流分段是提高流处理器上流寄存器文件(stream register file,简称 SRF)带宽利用率的重要途径之一.其中,量化受段大小影响的程序运行时间是获得最优分段的关键.为此提出了一种基于参数模型的长流分段技术,旨在获得理论上的最优分段以最小化程序运行时间.首先,建立了一个预取和重用优化指导的参数模型,以反映段大小对流处理器上程序性能的影响.然后,基于该模型分析,分别研究了计算密集型程序和访存密集型程序的最优分段策略.最后提出一种面向任意程序的最优分段技术.实验结果表明,该长流分段技术能够有效地避免和隐藏片外访存延迟,从而充分开发流处理器强大的计算能力.

* Supported by the National Natural Science Foundation of China under Grant Nos.60621003, 60633050 (国家自然科学基金)
Received 2008-02-14; Accepted 2008-06-03

© 中国科学院软件研究所 http://www.jos.org.cn

关键词: 分段;imagine;最优段大小;SRF 局部性;访存和计算的重叠
中图法分类号: TP311 文献标识码: A

Imagine 流体系结构是一种致力于缓解存储墙问题的新型体系结构.它综合利用大量的运算单元、有效的多级存储层次和多种并行技术[1,2],在媒体处理和信号处理等领域都获得了很高的性能[3,4].科学计算程序是否适合 Imagine 流体系结构是当前的热点讨论问题[5−9].

流寄存器文件(stream register file,简称 SRF)是流处理器片上的一个软件管理的存储器[2].提高 SRF 带宽利用率对于开发流处理器强大的计算能力至关重要.SRF 是存放流数据的主要部件,因此在硬件上流的最大长度不能超出 SRF.而科学计算程序的数据规模通常很大,因此有必要研究对长流的分割技术,以提高 SRF 带宽利用率.

目前流级编译已采用双缓冲机制[10−12]支持任意长度的流.由于该策略会造成 SRF 中参与运算的流长度不匹配,从而浪费 SRF 的存储空间并降低 SRF 的带宽利用率.为了解决现有双缓冲算法的低效问题,我们认为需要在程序级将长流分段,即 strip-mining 技术,使编译器无须提供双缓冲机制,并保持均衡的数据流使得核级计算不会等待,其中获得最优分段大小是长流分段技术的基础.最优分段大小是指程序运行时间最短时所采用的分段长度,因此量化受段大小影响的程序运行时间是获得最优分段的关键.为此,本文提出了一种基于参数模型的最优段大小选择策略,旨在通过对程序运行时间建模以获得理论上的最优分段.首先,建立了一种量化程序运行时间的参数模型,以反映段大小对程序性能的影响.然后,基于对该模型的分析,分别研究了计算密集型程序和访存密集型程序的最优分段策略.最后,提出一种面向任意程序的最优分段技术.实验结果表明,基于参数模型的长流分段技术能够有效地避免和隐藏片外访存延迟,从而减小 SRF 的浪费,提高 SRF 带宽利用率.

1 Imagine 流处理系统

Imagine 是一款可编程的单芯片处理器,它包括 48 个运算单元组成的 8 个 SIMD 运算簇(cluster),并提供三级带宽层次:局部寄存器文件(LRF)、流寄存器文件(SRF)、片外存储器(DRAM)[1,2].其中,最内层的 LRF 直接为 Cluster 中的功能部件提供数据,中间层次的 SRF 通过流缓冲器(SB)和 Cluster 交换数据,片外存储系统通过地址生成器(AG)产生不同模式的流.Imagine 体系结构如图 1 所示.

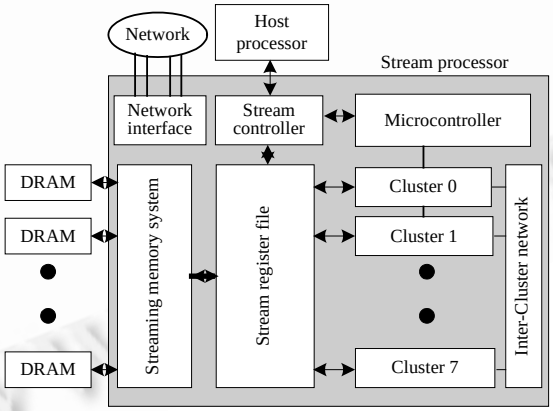


Fig.1 Imagine stream architecture

图 1 Imagine 流体系结构

Imagine 支持一种两级编程模型:流级(StreamC 编程)和核级(KernelC 编程)[12,13],分别在主机和 Imagine 上运行.该模型将数据组织成流,将流上的计算划分成多个计算核心(kernel).两级编程模型体现了计算与访存相分离的思想,有利于编译器进行专门优化,从而使流编程模型能够很好地被映射到底层硬件上.

© 中国科学院软件研究所 http://www.jos.org.cn

2 基于模型的长流分段技术

分段后的数据流是驱动一次计算的基本单位,而最优分段的大小将选取程序运行时间最短时采用的段大小.可见,对给定的程序运行时间,建模是长流分段的关键技术.因此,本节提出基于参数模型的长流分段技术.首先建立程序的参数模型,然后根据该参数模型,依次讨论计算密集型程序和访存密集型程序在各种优化技术下的最优分段策略,最后根据所得出的结论,得到任意程序的最优分段策略.

2.1 建立参数模型

我们对不同预取或重用策略下的程序运行时间建立了一个合理的参数模型.该模型基于一些合理的假设:流分配开销可以完全隐藏,kernel 切换开销隐藏于模型的常数项中,kernel 同构.为了降低建模复杂度,我们采用"先合并后分治"的策略,首先获得单 kernel 的总流长 L 分段后对应的总最优段大小 s_0.然后,根据每条流长 l_i 占总流长 L 的比例,成比例获取 s_0^i,即 $s_0^i = s_0 \cdot l_i / L$, $i = 1...n$.

由于 Imagine 是计算和访存分离的结构,若不考虑流的预取和重用,程序的总运行时间 $T(s)$等于所有 kernel 的启动时间 $T_{in}(s)$、计算时间 $T_k(s)$和结束时间 $T_{out}(s)$的总和,即 $T(s)=\sum(T_{in}(s)+Tk(s)+T_{out}(s))$,其中 s 表示 kernel 中所有流分段大小之和.记 $T_{in}(s)$和 $T_{out}(s)$之和为 kernel 的访存时间 $T_m(s)$.实际运行时通过最大化访存时间和计算时间的重叠度来缩短总的运行时间[14−16].可见,kernel 的启动、计算和结束时间随段大小变化的函数是决定程序运行时间的关键因素.根据 Imagine 的实际参数和流程序的行为,我们对这 3 个函数参数化.首先,通过系统参数的分析,kernel 启动和结束开销取决于指令运行时间、等待时间和传输时间,因此 kernel 启动和结束时间与段大小 s 呈线性关系,即 $T_{in}(s)=k_1s+c_1$,$T_{out}(s)=k_2s+c_2$,其中 k_1 和 k_2 表示 kernel 启动和结束时单位数据载入耗费的时间,c_1 和 c_2 表示 kernel 启动和结束时不随 s 变化的固有时间.其次,由于 kernel 计算附着于数据流,因此一般情况下,kernel 运行时间和段大小存在线性关系,即 $T_k(s)=k_3s+c_3$,其中 k_3 表示 kernel 中段 s 的单位数据载入耗费的时间,c_3 表示 kernel 中不随 s 变化的固有时间.

2.2 计算密集型程序

对于计算密集型程序,kernel 的启动和结束时间小于计算时间,即 $k_1s + c_1 + k_2s + c_2 < k_3s + c_3$.我们建立不同优化策略下计算密集型程序的运行时间随 s 变化的参数模型,从而求得运行时间最短时的最优段大小.

2.2.1 只预取

对于没有可重用数据流的计算密集型程序,其运行时只能通过预取优化来隐藏访存延迟,从而减小程序的运行时间,如图 2 所示.可以看出,对于只预取的计算密集型程序,程序运行过程中 kernel 的访存时间被计算时间完全隐藏,其运行时间 $T(s)$计算如下:

$$T(s) = T_{\text{in}}(s) + n \cdot T_c(s) + T_{out}(s) = k_1 s + c_1 + (L/s)(k_3 s + c_3) + k_2 s + c_2 \quad (1)$$

对公式(1)求导,记 $\mathrm{d}T(s)/\mathrm{d}s = T'(s)$.令 $T'(s) = k_1 - L \cdot c_3 / s^2 + k_2 = 0$,可求得 $T(s)$取极小值时的段大小 $s_{\min} = \sqrt{Lc_3/(k_1 + k_2)}$.从而得知当系统参数固定时,随着 kernel 计算时间和流长度的增长,$s_{\min}$ 呈增长趋势.由于 SRF 容量 C_s 的限制以及只预取优化需要为后续 kernel 预留 SRF 空间,因此, 一个 kernel 的输入流和输出流的规模总和不应超过 C_s 的一半.若 $s_{\min} > C_s/2$,则最优段大小 $s_0 = C_s/2$,否则 $s_0 = s_{\min}$.

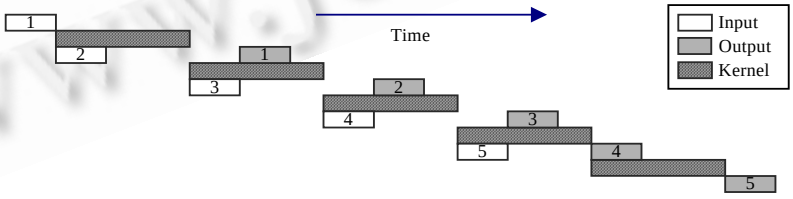


Fig.2 Only prefetching optimization

图 2 只预取优化

© 中国科学院软件研究所 http://www.jos.org.cn

2.2.2 只重用

为了定性分析不同类型的数据流重用对分段的影响,提出以下定义:

定义 1. 若两个 kernel 发生重用,且这两个 kernel 访问的数据流规模总和小于 SRF 容量,则称这次重用为有效 kernel 重用.有效 kernel 重用包括输入重用(input reuse)、输出输入重用(anti reuse)和输出重用(output reuse).

定义 2. kernel 重用度定义为程序中有效 kernel 重用的总次数.

若分段后数据流较长,无法为预取优化预留足够的 SRF 空间,则前一 kernel 未执行完毕,不能提前预取下一个 kernel 的数据流.对于这一类程序,若要缩短运行时间,必须开发分段后 kernel 间的数据流重用.在本节中,我们分析只采用重用优化策略的程序运行情况,如图 3 所示.不失一般性,假设程序的所有 kernel 存在 x 次输入重用,y 次输出输入重用,z 次输出重用.程序运行时间 $T(s)$可计算如下:

$$T(s)=n\cdot\left(T_{in}(s)+T_c(s)+T_{out}(s)\right)-(x\cdot input\ reuse\ overhead+y\cdot anti\ reuse\ overhead+z\cdot output\ reuse\ over\ head)$$
$$=-z_1 s+z_2/s+z_3 \tag{2}$$

其中, $z_1=(x+y)k_1+(x+y+z)k_2$, $z_2=L(c_1+c_2+c_3)$, $z_3=L(k_1+k_2+k_3)-(x+y)c_1-(x+y+z)c_2$.

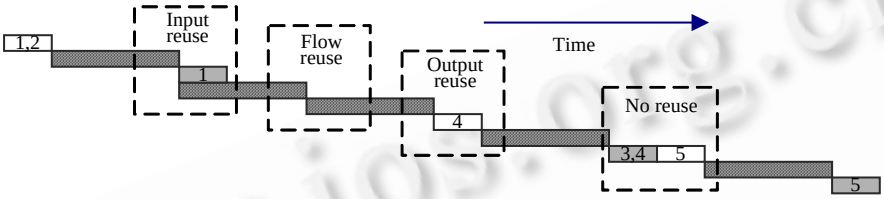


Fig.3 Only reuse optimization

图 3 只重用优化

对公式(2)求导, $T'(s)=-z_1-z_2/s^2$.由于 $z_1>0,z_2>0$,所以对于任意段大小 s,$T'(s)<0$,可见,$T(s)$在区间$(-\infty,+\infty)$上是严格单调减函数,程序的运行时间随着 s 的增大呈递减趋势.受限于 SRF 容量 C_s,s 最大不超过 SRF 容量,即最优段大小 $s_0=C_s$.

2.2.3 同时考虑预取和重用

对于计算密集型程序,由于预取优化能够隐藏访存延迟,重用优化能够避免访存延迟,因此同时采用预取和重用优化的程序运行情况类似于只采用预取优化的程序运行情况,如图 4 所示,kernel 重用对于程序运行时间没有影响.程序运行时间 $T(s)$和 s_0 的计算方法与第 2.2.1 节相同.

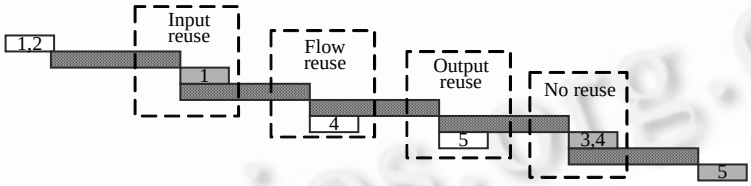


Fig.4 Combined prefetching and reuse

图 4 同时采用预取和重用优化

2.2.4 3 种方案的选择

上面 3 节分别给出了针对计算密集型程序的 3 种预取和重用优化方案,即只预取方案、只重用方案和同时考虑重用和预取方案.为了研究这 3 种方案的适用范围,本节讨论对于不同计算密集型程序,如何在这 3 种方案中选取合适的优化方案的方法.给定计算密集型程序,由于采用方案 1 和方案 3 的程序运行时间相同,因此当 kernel 间没有重用或重用度低时,考虑采用较为简单的方案 1:只预取优化.若 kernel 重用度高,可以采用方案 2.因此确定 kernel 重用度分界值是方案选择问题的关键.

输出重用的本质是一种重复输出操作,所以重用源语句在程序语义上是冗余语句,能够被编译器识别并被删除.因此,本节所讨论的重用度不包含输出重用次数.设给定程序的重用度为 R.根据以上分析,方案 2 在段大小

© 中国科学院软件研究所 http://www.jos.org.cn

为 $s=C_s$ 时程序运行时间最短,最短运行时间为

$$T_2(C_s)=(L/C_s)(k_3C_s+c_3+k_1C_s+c_1+k_2C_s+c_2)\ -(R\cdot(k_1C_s+c_1+k_2C_s+c_2)),$$

显然,$T_2(C_s)$随着 R 增大而减小.方案 1 在段大小为 $s=\sqrt{Lc_3/(k_1+k_2)}$ 时程序运行时间最短,最短运行时间为 $T_1\left(\sqrt{Lc_3/(k_1+k_2)}\right)=2\sqrt{Lc_3(k_1+k_2)}+c_1+c_2+Lk_3$,显然,该最短运行时间不随 R 的变化而发生改变.不同方案的 kernel 重用度对程序运行时间的影响如图 5 所示,图中两直线的交点记为 R_0,计算方式如公式(3)所示.R_0 是不同方案性能差异的转折点,即我们求解的目标——分界重用度.当 kernel 重用度 $R>R_0$ 时,采用方案 2(只重用)并取尽可能大的段时程序将取得较好的性能;否则,应采用方案 1.

$$R_0=\frac{z_2'}{z_1'}=\frac{(L/C_s)(c_1+c_2+c_3)+L(k_1+k_2)-(c_1+c_2)-2\sqrt{Lc_3(k_1+k_2)}}{(k_1+k_2)C_s+(c_1+c_2)} \tag{3}$$

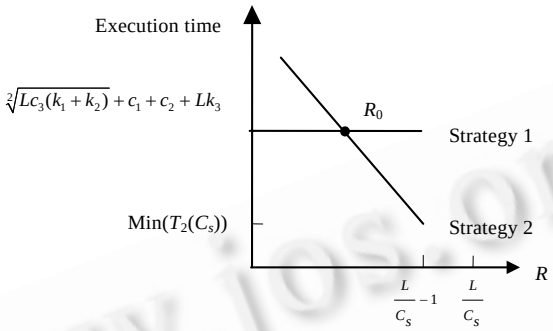


Fig.5 Effect of kernel reuse degree on program execution time

图 5 Kernel 重用度对程序运行时间的影响

2.3 访存密集型程序

对于访存密集型程序,kernel 的启动和结束时间大于计算时间,即 $k_1s+c_1+k_2s+c_2>k_3s+c_3$.其中,我们称 kernel 计算时间小于任意启动或结束开销的程序($k_1s+c_1>k_3s+c_3$ 或 $k_2s+c_2>k_3s+c_3$)为绝对访存密集型程序.我们建立不同优化策略下访存密集型程序的运行时间的参数模型,从而求得最小运行时间处的最优段大小.

2.3.1 只预取

对于采用只预取优化的访存密集型程序,程序运行过程中最多有一个 kernel 的计算时间不能被访存时间隐藏,其余 kernel 的计算时间都可被访存时间隐藏,如图 6 所示.程序运行时间 $T(s)$计算如下:

$$T(s)=n\cdot(T_{in}(s)+T_{out}(s))+\delta\cdot T_k(s)\ =\delta k_3s+(Lc_1+Lc_2)/s+Lk_1+Lk_2+\delta c_3 \tag{4}$$

其中,δ为无法隐藏的 kernel 时间权值,$0\le\delta\le1$.对公式(4)求导,令 $T'(s)=\delta k_3-(Lc_1+Lc_2)/s^2=0$,可求得 $T(s)$取极小值时的段大小 $s_{\min}=\sqrt{(Lc_1+Lc_2)/(\delta k_3)}$.从而得知,当系统参数固定时,随着流长度的增长或$\delta$的减小,$s_{\min}$ 呈增长趋势.可见对于访存密集型程序,能否隐藏计算时间是影响段大小的关键因素.对于绝对访存密集型程序,δ=0,此时,程序运行时间 $T(s)=(Lc_1+Lc_2)/s+Lk_1+Lk_2$,可以看出,段 s 越大程序运行时间越短,但 s 不能超过发生双缓冲的容量限制.由于 SRF 容量限制,若 $s_{\min}=C_s/2$,则最优段大小 $s_0=C_s/2$,否则 $s_0=s_{\min}$.

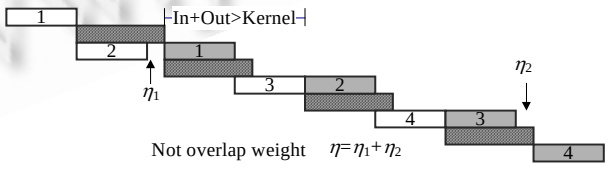


Fig.6 Only prefetching optimization

图 6 只预取优化

© 中国科学院软件研究所 http://www.jos.org.cn

2.3.2 只重用

对于采用只重用优化的访存密集型程序,为了简单化,假设 kernel 计算时间大于任意启动或者结束延迟.不失一般性,假设程序的所有 kernel 存在 x 次输入重用,y 次输出输入重用,z 次输出重用.程序运行情况如图 7 所示,在时间轴,重用后将 kernel 时间取代访存时间,所以程序运行时间 $T(s)$和 s_0 的计算方法与第 2.2.2 节相同.

2.3.3 同时考虑预取和重用

对于同时采用预取和重用优化的访存密集型程序,我们只考虑总的 kernel 重用次数 R,其程序运行情况如图 8 所示.程序运行时间 $T(s)$可计算如下:

$$T(s)=R(k_3s+c_3)+(L/s-R)(k_1s+c_1+k_2s+c_2)=R\big(k_3-(k_1+k_2)\big)s+(Lc_1+Lc_2)/s+Lk_1+Lk_2-R(c_1+c_2) \quad (5)$$

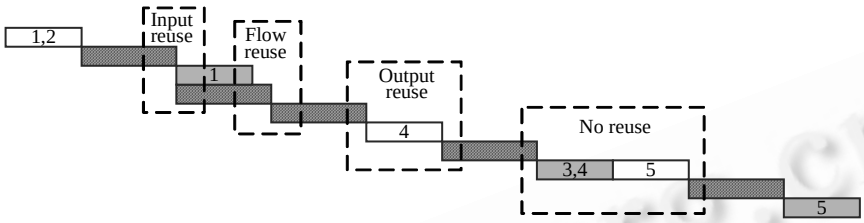


Fig.7 Only reuse optimization

图 7 只重用优化

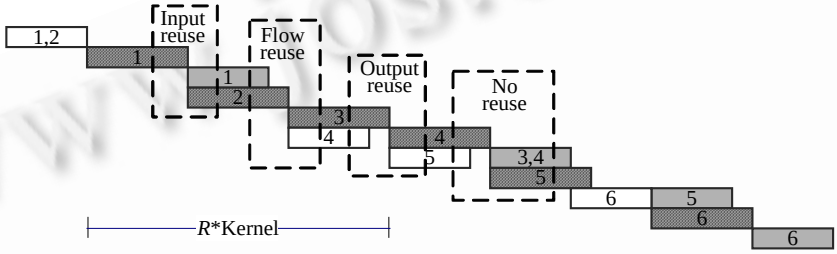


Fig.8 Combined prefetching and reuse

图 8 同时采用预取和重用优化

为了求解 $T(s)$最小时的段大小,下面分情况讨论:

(1) 当 $k_3-(k_1+k_2)>0$ 时,$T(s)$存在极小值.对公式(5)求导,令 $T'(s)=R(k_3-k_1-k_2)-(Lc_1+Lc_2)/s^2=0$,可求得 $T(s)$取极小值时的段大小 $s_{\min}=\sqrt{(Lc_1+Lc_2)/\big(R(k_3-k_1-k_2)\big)}$.并且随着 kernel 重用度的增大,$s_{\min}$减小.

(2) 当 $k_3-(k_1+k_2)\le 0$ 时,$T'(s)<0$,$T(s)$为区间$(-\infty,+\infty)$上的严格单调递减函数.因此,程序的运行时间随着段 s 的增大呈递减趋势.考虑 SRF 容量限制,$s_0=C_s/2$.并且随着 kernel 重用度增大,$T(s)$减小.

2.3.4 3 种方案的选择

上面 3 节分别给出了针对访存密集型程序的 3 种预取和重用优化方案,即只预取方案、只重用方案和同时考虑重用和预取方案.为了研究这 3 种方案的适用范围,本节讨论对于不同访存密集型程序,如何在这 3 种方案中选取合适的优化方案的方法.给定访存密集型程序,若 kernel 间没有重用或重用度低,可以考虑采用方案 1.若 kernel 重用度高,可以采用方案 2 或者方案 3.因此确定 kernel 重用度分界值 R_0 是方案选择问题的关键.本节以绝对访存密集型程序为例,说明优化策略的选择方法以及重用度分界.对于非绝对访存密集型程序,需要考虑无法隐藏的 kernel 时间权值δ,可类比讨论.

根据以上分析,3 种方案的最短程序运行时间可以如下计算:方案 1 的最短程序运行时间为 $T_1(C_s/2)=2L(c_1+c_2)/C_s+L(k_1+k_2)$,且 $T_1(C_s/2)$ 不随 R 变化而改变.方案 2 的最短程序运行时间为 $T_2(C_s)=-\big((k_1+k_2)C_s+c_1+c_2\big)R+L(c_1+c_2+c_3)/C_s+L(k_1+k_2+k_3)$,可见,$T_2(C_s)$函数是关于 R 的线性递减函数.方案 3 的最短程序运行时间为 $T_3(C_s/2)=-\big((k_1+k_2-k_3)C_s/2+c_1+c_2\big)R+2L(c_1+c_2)/C_s+L(k_1+k_2)$,可见 $T_3(C_s/2)$

© 中国科学院软件研究所 http://www.jos.org.cn

也是关于 R 的线性递减函数.

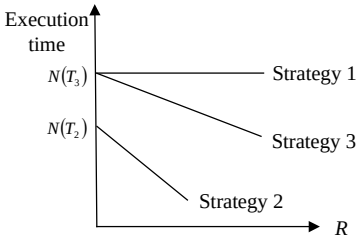


Fig.9 $N(T_3) \geq N(T_2)$

图 9 $N(T_3) \geq N(T_2)$

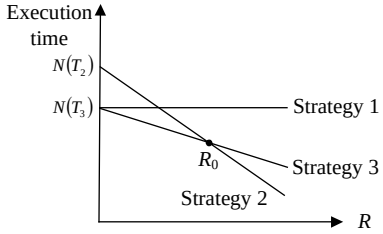


Fig.10 $N(T_3) < N(T_2)$

图 10 $N(T_3) < N(T_2)$

为了便于描述,设 $M(f)$和 $N(f)$表示 f 函数关于重用度 R 的斜率和截距.下面通过分析程序参数来获得不同方案的程序运行时间函数之间的关系.

1) 若 $c_1+c_2-c_3 \geq k_3C_s$,则 $N(T_3)-N(T_2) \geq 0$.由于 $M(T_3) > M(T_2)$,根据以上分析,在重用度 R 的[0,∞]区间上,3种方案的最短程序运行时间函数满足 $T_2 \leq T_3 \leq T_1$,如图 9 所示.对于这种情况,绝对访存密集型程序应当选择方案 2 作为优化策略,即无论程序的重用度多高,我们只考虑采用长流分段方法开发 kernel 重用.

2) 若 $c_1+c_2-c_3 < k_3C_s$,则 $N(T_3)-N(T_2) < 0$,如图 10 所示.由于 $M(T_3) > M(T_2)$,在重用度 R 的[0,∞]区间上,两种方案的最短程序运行时间存在分界重用度 R_0,如公式(6)所示.该重用度是不同方案性能差异的转折点,即我们求解的目标.当 $R>R_0$ 时,采用方案 2(只重用)并取尽可能大的段时程序将取得较好的性能;否则,应采用方案 3.

$$R_0 = \frac{-z_2'}{z_1'} = \frac{2Lk_3 - 2L(c_1+c_2-c_3)/C_s}{(k_1+k_2+k_3)C_s} \tag{6}$$

2.4 任意程序

任意程序的计算密集和访存密集特性可能随段 s 变化而变化.当 s 满足 $k_1s+c_1+k_2s+c_2 < k_3s+c_3$ 时,程序呈计算密集型特点;当 s 满足 $k_1s+c_1+k_2s+c_2 > k_3s+c_3$ 时,程序呈访存密集型特点.经过公式变形,计算密集型和访存密集型程序的分类如下所示,分别标记为①类和②类.

① $(k_1+k_2-k_3)\cdot s < c_3-c_1-c_2$,计算密集型.

② $(k_1+k_2-k_3)\cdot s > c_3-c_1-c_2$,访存密集型.

对于给定的任意程序,根据第 2.2 节和第 2.3 节中的最优段选择策略,通过分析程序的参数特性来获得程序运行时间函数 $T(s)$,从而得到任意程序的最优段选择策略.

2.4.1 确定任意程序的 $T(s)$曲线

根据以上分析,可知计算参数(k_3,c_3)和访存参数(k_1,k_2,c_1,c_2)是影响程序特性的关键.对于任意程序,可以通过分析其参数 $k_1+k_2-k_3$ 和 $c_3-c_1-c_2$ 的正负取值,确定程序运行时间 $T(s)$函数的形状.为了形象地表示,我们采用图示函数①|②表示 $T(s)$函数,其中竖线处的值代表分界线,左边取计算密集型函数,右边取访存密集型函数.令 $s'=(c_3-c_1-c_2)/(k_1+k_2-k_3)$,根据①②可知,$s'$ 是计算密集和访存密集类型的分界值.

对于给定程序,首先判断程序参数 $k_1+k_2-k_3$ 和 $c_3-c_1-c_2$ 的正负情况,其次查阅图 11,获得任意程序的程序性质,最后根据前述临界重用度的判断,选择只预取、只重用,还是同时考虑预取和重用的优化策略.最后确定程序运行时间 $T(s)$的函数曲线.

© 中国科学院软件研究所 http://www.jos.org.cn

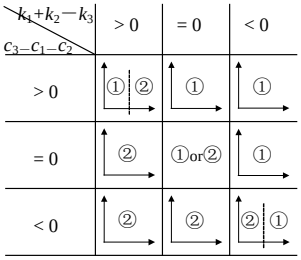


Fig.11 Selecting program property figure

图 11 程序性质选择图

2.4.2 判断曲线是否连续

如图 11 所示,只有两种情形中存在同一程序同时体现计算密集和访存密集的特性.对于这两种情况,根据前文所述,$T(s)$曲线分界两边的具体函数已经确定.为了更加准确地给出分界情况的 $T(s)$函数曲线,我们需要判断该分界处两边的函数是否连续.即判断分界值 s'和两函数交点 s_x 的值是否相等.

下面以分界两边分别为计算密集型只预取和访存密集型只预取的程序为例,介绍 $T(s)$的确定过程.分界两边的 $T(s)$差异记为 $T_{L-R}(s)$,$T_{L-R}(s)=T_L(s)-T_R(s)=(k_1+k_2-\delta k_3)s+(L-s)(c_3-c_1-c_2)+c_1+c_2-\delta c_3+L(k_3-k_1-k_2)$.为了求得在分界值 s'处的两种类型运行时间的函数差异,将 s'代入 $T_{L-R}(s)$并化简,可得:

$$T_{L-R}(s')=(1-\delta)\big((k_1+k_2)c_3-(c_1+c_2)k_3\big)/(k_1+k_2-k_3) \tag{7}$$

若分界线 s'两边的 $T(s)$在 s'处连续,则 $T_{L-R}(s')=0$.根据公式(7),可得 $(1-\delta)\big((k_1+k_2)c_3-(c_1+c_2)k_3\big)=0$.因此,若程序满足以下任意条件,其运行时间函数 $T(s)$在 s'处连续,如图 12 所示.

1) δ=1,即程序的单 kernel 计算时间和访存延迟相等,$k_3=k_1+k_2$,$c_3=c_1+c_2$.

2) $(k_1+k_2)c_3-(c_1+c_2)k_3=0$,即程序参数满足 $(k_1+k_2)c_3=(c_1+c_2)k_3$.

若程序参数不满足以上任意条件,运行时间函数 $T(s)$在 s'处发生间跃,该函数为不连续函数,如图 13 所示.

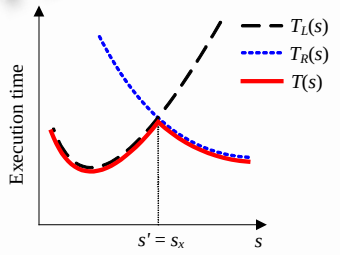


Fig.12 Continuous program execution function

图 12 连续的程序运行时间函数

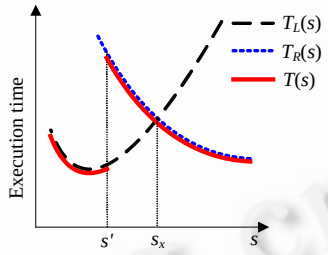


Fig.13 Discontinuous program execution function

图 13 不连续的程序运行时间函数

3 编译实现

为了实现我们所设计的最优段选择算法,需确定该模型所需的访存参数(k_1,k_2,c_1,c_2)和计算参数(k_3,c_3).kernel 的访存时间($T_m(s)=(k_1+k_2)s+(c_1+c_2)$)取决于访存系统指令运行时间、等待时间和传输时间.对于给定程序,根据单 AG 读/写的时间和指令发射方式确定单位数据访存耗费的时间 k_1 和 k_2,根据数据读/写之前访存系统所需的固有时间(包括指令发射开销等)确定访存时不随 s 变化的固有时间 c_1 和 c_2.kernel 计算参数 k_3 和 c_3 可以通过综合分析核级编译生成的 uc 文件和 KernelC 文件获得.

图 14 给出了 Imagine 上流编译器 IStream 的编译流程,本文的工作集中于灰色图框部分.图 14 下部给出了分段优化的具体步骤.首先,对于给定程序,确定程序的计算参数和访存参数,根据图 11 所示的程序性质选择表,判断程序随段大小变化而呈现的程序特性.其次,基于确定的程序特性和程序中的 kernel 重用度,根据第 2.2.4 节和第 2.3.4 节给出的优化策略选择方案,选择给定程序所需的优化策略.然后,若是计算密集和访存密集混合性

© 中国科学院软件研究所 http://www.jos.org.cn

质的程序,则需要根据公式(7)确定该函数是否在分界线 s'处连续,以获得准确的 $T(s)$函数.最后,按照第 2.2 节和第 2.3 节介绍的方法确定该程序的最小程序运行时间处的段大小 s_0.

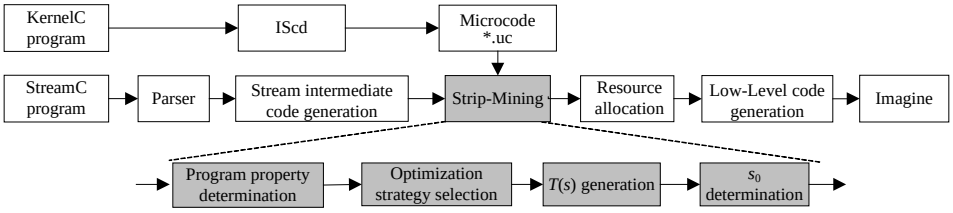


Fig.14 Strip-Mining implementation in compiler

图 14 长流分段编译实现

4 实验评测

为了评测长流分段策略的有效性,我们选取科学计算领域中 9 个典型的科学计算核心程序,见表 1.其中,NLAG 是一个二维非线性扩散流体力学方程组的代数求解器,Transp 是光学应用 Capao 中耗时的子函数.我们的实验分别测试了所有程序的 FORTRAN 版本(Seri)、未分段流版本(Orig)和分段优化流版本(OSM)的性能.

FORTRAN 版本的程序由 Intel 提供的 ifort 编译,编译开关为-*O*3,然后运行于一个单核的 Itanium 2 服务器上.Itanium 2 的工作频率为 1.6GHz,其一级 cache 大小为 16KB,二级 cache 大小为 256KB,三级 cache 大小为 6MB.Itanium 2 服务器上有一个 4GB 的片外存储器,其访存带宽为 6.4GB/s.后两个版本的流程序由 IStream 和 IScd 编译,运行于 Imagine 的 500MHz 时钟精确模拟器——Isim[12].该模拟器设置为一个地址发生器.

表 2 列出了分段优化流版本分别与串行版本、未分段流版本相比,这 9 个测试用例所能获得的性能加速比.与运行于 Itanium 2 的串行程序相比,采用长流分段优化的大多数程序(EP,DFFT,Laplace,Jacobi,GEMM 和 Transp)能够获得较高的加速比,只有 3 个程序(Swim,MG,NLAG-5)几乎没有加速,说明我们提出的长流分段优化能够有效地开发流处理器的强大计算能力.与未分段流程序相比,所有的长流分段版本都能获得较高的性能,这是因为长流分段优化可以有效地避免和隐藏访存延迟,提高访存系统的供数能力.

Table 1 Testing programs

表 1 测试程序

Program	Swim	EP	MG	FFT	Laplace	Jacobi	GEMM	NLAG-5	Transp
Source	SPEC2000	NPB	NPB	–	NCSA	–	BLAS	–	–
Arrays	14	1	3	1	2	4	3	2	5
Size	513×513	131 072	64×64×64	4 096	256×256	128×128	256×256	256×256	512×512

Table 2 Performance speedup

表 2 性能加速比

Program	Swim	EP	MG	FFT	Laplace	Jacobi	GEMM	NLAG-5	Transp
OSM vs. Seri	1.36	4.55	2.35	7.54	3.41	2.04	3.17	1.92	2.15
OSM vs. Orig	3.56	1.08	3.23	2.22	2.65	3.57	5.39	1.65	2.53

SRF 中的流重用是长流分段的主要目的之一.流重用能够减少片外访存,因此访存次数是反映长流分段有效性的关键要素.图 15 给出了未分段流版本和分段优化流版本的片外访存次数.所有的数值都基于未分段流版本进行了归一化.显然,所有分段优化流版本的访存次数较之未分段流版本都有所减少,这是因为我们的长流分段技术能够有效地开发 kernel 重用以提高 SRF 局部性.特别是,由于 GEMM 程序的 kernel 重用度高,因此可以通过重用小的矩阵段来达到避免访存的目的,分段优化后,其访存次数具有较大幅度的减少.而未分段的 EP 流程序具有较少的访存次数,因此分段优化所能避免的访存数量范围有限,优化幅度较小.

© 中国科学院软件研究所 http://www.jos.org.cn

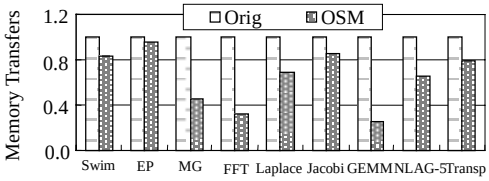


Fig.15 Normalized memory transfers

图 15 归一化的访存次数

计算访存之间的重叠度是衡量流处理器性能的一个重要因素.图 16 列出了 9 个测试用例的流版本在 Imagine 上的访存时间(mem)和计算核心运行时间(com)占程序总运行时间的比例.当访存延迟和计算时间重叠时,访存比例和计算比例总和将超过 100%.对于大多数未分段流程序,访存时间比例近似于 100%,意味着程序运行总时间几乎都被访存时间所占据.对于长流分段后的流程序,大多数程序的访存比例和计算比例之和都超过 100%,也就是说,采用我们所提出的长流分段优化能够有效地开发访存延迟和计算时间的重叠执行.其中,EP 程序分段优化后其访存比例变化较小,这是因为 EP 是一个计算密集型程序,其计算时间是程序总时间的主要部分,无法有效地开发计算时间和访存时间的重叠.

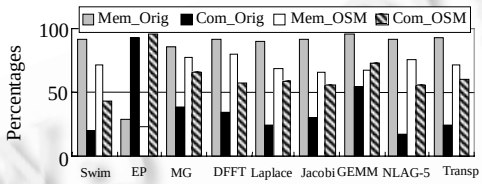


Fig.16 Percentages of memory access time and computation time to total execution time

图 16 访存时间和计算时间占总运行时间的百分比

为了评测长流分段优化的可扩展性,我们测试采用长流分段优化后不同数据规模的程序的性能.以 Jacobi 程序为例,图 17 给出了 Jacobi 程序分段优化版本较之未分段版本的加速比.显然,随着数据规模的扩大,长流分段版本在 Imagine 上的加速比也随之增大.实际上,当 Jacobi 规模小于 256×256 时,即使不采用长流分段,Jacobi 程序也不会产生双缓冲,因此程序的数据重用已被开发.此时采用分段技术只能促进小段流的计算访存重叠,但也引入了更多的 kernel 切换开销和流分配开销,分段优化的优势不明显.当 Jacobi 的数据规模超过 256×256 时,对长流进行合适的分段能够保证大规模的 Jacobi 程序不出现双缓冲,并且可以更加有效地隐藏访存延迟.结果显示,我们提出的长流分段优化具有良好的可扩展性.

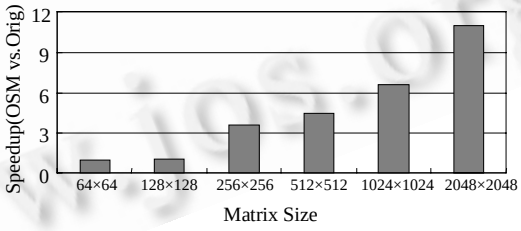


Fig.17 Scalability

图 17 可扩展性

5 相关工作

循环分块(tiling)是提高传统处理器的 Cache 重用的有效方法.针对 Cache 的分块技术已有深入的研究[17−19]等.其中,文献[18]提出了相对简单的启发式段选择方法,文献[19]给出了确定块大小的量化方法.这些方法通过

© 中国科学院软件研究所 http://www.jos.org.cn

分析程序的行为,确定适合不同程序的块大小.这种面向程序的块选择方法准确、灵活.循环分块不仅适用于基于 Cache 的系统,也是开发基于 Scratch-Pad Memory 的系统性能的有效手段[20].

流处理器的 SRF 和传统处理器的 cache 都是为缓解存储墙问题而引入的片上存储中间层次.流处理器上数据流的组织是影响 SRF 重用和预取的关键.当单 kernel 的所有流长度超过 SRF 容量时,我们需要对长流进行分段.流的分段技术类似于循环分块技术,是实施于数据流的流程序变换方法.Stanford 大学的流编译组研究了一种二叉树搜索最优段大小的方法[11],着重于开发 kernel 重用.该算法是一种启发式方法,没有建立详尽的参数模型以准确指导最优段选择,也没有同时考虑预取和重用优化的折衷.

借鉴传统 Cache 分块大小和已有的 SRF 中长流分段方案,可知综合考虑系统参数和程序行为两方面是决定 SRF 分段大小的关键.我们从模型指导的角度,详细分析并量化了影响流处理器上最优段大小的关键因素,建立了准确的最优段选择模型.并基于该模型分析,提出了有效的预取和重用指导的最优段选择策略.

6 结束语

为了提高 SRF 的利用率,本文提出了一种基于参数模型的最优段大小选择策略.该策略能够获得理论上的最优分段.首先建立了流体系结构上程序运行时间的参数模型.该模型能够有效地反映段大小对程序计算时间、访存时间以及两者的重叠程度的影响.基于该模型的分析,我们设计了流处理器上最优段大小选择策略,旨在通过获得最优分段来最大化流预取和流重用的收益,从而最小化程序运行时间.本文的研究结论能够在理论上对长流分段技术起到有益的指导作用.实验结果表明,所提出的最优段大小选择策略能够有效地避免和隐藏访存延迟,从而保证了 Imagine 上大量运算单元的持续运转.

未来的工作主要包括:首先,测试更多的科学计算应用来评测我们提出的最优段选择策略;其次,研究模型和经验方法共同指导的段选择策略,从而可以简单、快速地获得近优段大小;最后,我们还会继续研究适用于流处理器的更多编程和编译优化方法.

References:

[1] Khailany B. The VLSI implementation and evaluation of area-and energy-efficient streaming media processors [Ph.D. Thesis]. Stanford: Stanford University, 2003.

[2] Jayasena NS. Memory hierarchy design for stream computing [Ph.D. Thesis]. Stanford: Stanford University, 2005.

[3] Khailany B, Dally WJ, Rixner S, Kapasi UJ, Mattson P, Namkoong J, Owens JD, Towles B, Chang A. Imagine: Media processing with streams. IEEE Micro, 2001,21(2):35−46.

[4] Gordon MI, Thies W, Amarasinghe S. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In: Proc. of the ASPLOS 2006. New York: ACM Press, 2006. 151−162.

[5] Du J, Yang X, Wang G, Ao F. Scientific computing applications on the Imagine stream processor. In: Jesshope C, Egan C, eds. Proc. of the 11th Asia-Pacific Computer Systems Architecture Conf. LNCS 4186, Springer-Verlag, 2006. 38−51.

[6] Yang X, Du J, Yan X, Deng Y. Matrix-Based programming optimization for improving memory hierarchy performance on Imagine. In: Proc. of the 4th Int'l Symposium on Parallel and Distributed Processing and Applications. LNCS 4330, Springer-Verlag, 2006. 782−793.

[7] Erez M. Merrimac-High-Performance, highly-efficient scientific computing with streams [Ph.D. Thesis]. Stanford: Stanford University, 2007.

[8] Yang X, Yan X, Xing Z, Deng Y, Jiang J, Zhang Y. A 64-bit stream processor architecture for scientific applications. In: Proc. of the 34th Annual Int'l Symp. on Computer Architecture. New York: ACM Press, 2007. 210−219.

[9] Yan X, Tang T, Deng Y, Du J, Yang X. Evaluation of transcendental functions on Imagine architecture. In: Proc. of the Int'l Conf. on Parallel Proc. Washington: IEEE Computer Society, 2007. 53−59.

[10] Kapasi UJ, Mattson P, Dally WJ, Owens JD, Towles B. Stream scheduling. In: Proc. of the 3rd Workshop on Media and Streaming Processors. 2001. 101−106.

© 中国科学院软件研究所 http://www.jos.org.cn

[11] Das A, Dally WJ, Mattson P. Compiling for stream processing. In: Proc. of the 15th Int'l Conf. on Parallel Architectures and Compilation Techniques. New York: ACM Press, 2006. 33−42.

[12] Mattson P. A programming system for the Imagine media processor [Ph.D. Thesis]. Stanford: Stanford University, 2002.

[13] Owens JD, Rixner S, Kapasi UJ, Mattson P, Towles B. Media processing applications on the Imagine stream processor. In: Proc. of the 2002 Int'l Conf. on Computer Design. Washington: IEEE Computer Society, 2002. 295−302.

[14] Kapasi UJ, Rixner S, Dally WJ, Kbailany B, Ahn JH, Mattson P, Owens JD. Programmable stream processors. IEEE Computer, 2003,36(8):54−62.

[15] Kapasi UJ, Dally WJ, Rixner S, Owens JD, Khailany B. The imagine stream processor. In: Proc. of the 2002 Int'l Conf. on Computer Design. Washington: IEEE Computer Society, 2002. 295−302.

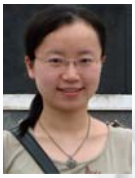
[16] Ahn JH, Dally WJ, Khailany B, Kapasi UJ, Das A. Evaluating the imagine stream architecture. In: Proc. of the 31th Int'l Symposium on Computer Architecture. Washington: IEEE Computer Society, 2004. 14−25.

[17] Song YH, Li Z. New tiling techniques to improve cache temporal locality. ACM SIGPLAN Notices, 1999, 34(5):215−228.

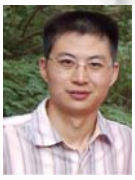
[18] Coleman S, McKinley KS. Tile size selection using cache organization and data layout. In: Proc. of the SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 1995. 279−290.

[19] Hsu CH, Kremer U. A quantitative analysis of tile size selection algorithms. Journal of Supercomputing, 2004,27(3):279−294.

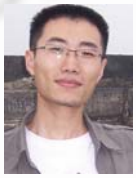
[20] Zhang C, Kurdahi F. On combining iteration space tiling with data space tiling for scratch-pad memory systems. In: Proc. of the 2005 Conf. on Asia South Pacific Design Automation. New York: ACM Press, 2005. 973−976.



杜静(1979－),女,山西晋中人,博士生,主要研究领域为高性能编译,并行体系结构.



敖富江(1975－),男,博士生,主要研究领域为数据流挖掘,并行计算.



唐滔(1984－),男,博士生,主要研究领域为高性能编译,并行体系结构.

杨学军(1963－),男,博士,教授,主要研究领域为并行体系结构,并行编译,并行操作系统.

© 中国科学院软件研究所 http://www.jos.org.cn