SIMD代码中的向量访存优化研究

徐金龙 赵荣彩 徐晓燕
(信息工程大学数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 向量程序来源于手工编写或由编译器自动生成。受限于编程人员和并行编译器的能力,得到的向量程序都存在一定的优化空间。优化编译器通常关注如何将串行程序向量化,但很少对向量程序进行优化。因此,提出了一种针对SIMD代码的向量访存优化方法。该方法首先分析程序是否需要优化,若存在需求,则对程序同时进行深度冗余优化和对齐优化。实验数据显示,提出的方法可以明显提高程序的运行效率,达到了目标。

关键词 向量化,SIMD,访存冗余,对齐优化
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.004

Memory Access Optimization for Vector Program of SIMD Form

XU Jin-long ZHAO Rong-cai XU Xiao-yan
(State Key Laboratory of Mathematical Engineering and Advanced Computing, PLA Information Engineering University, Zhengzhou 450001, China)

Abstract There are two ways to get vector program, one is handwritten, and the other is generated automatically by the compiler. Limited to programmers and parallel compiler' ability, there always is some optimization space in vector program. The optimizing compiler concernes most about how to transform the serial program into vector form, rarely do further optimization after the vector form generating. We proposed a memory access optimization method for vector program of SIMD form. Firstly it determines that whether the program needs to be optimized. If optimization is needed, redundancy optimization and align optimization will be implemented for the vector form program. Experimental data show that the proposed method can significantly improve the running efficiency of the program, and the goal is achieved.

Keywords Vectorization, Single instruction multiple data, Memory access redundant, Alignment optimization

1 引言

SIMD扩展部件以较低的硬件成本和较少的计算资源实现计算性能的成倍跃升,使它逐渐成为高性能计算机的必备的计算加速部件,常见微处理器大多都提供了SIMD功能扩展部件,并得到迅速发展[1]。SIMD数据宽度已经从最初的128位扩展到512位,例如Intel的Xeon Phi协处理器中具备512bit数据宽度的SIMD单元,所以通过SIMD并行化已经成为提高程序性能的重要途径。

为了避免手工编写向量化程序带来的困难,SIMD自动向量化方法成为编译领域一个重要的研究分支,它是指将串行程序的SIMD并行性自动转换为SIMD指令的编译技术,目前可分为3类:1)面向循环的传统向量化技术[2];2)面向基本块的超字并行技术[3];3)基于模式匹配的SIMD指令生成技术[4,5]。这些技术的目的都是充分发掘程序中的并行性,生成向量化代码,而未考虑生成代码后可能存在的优化空间。

无论是手工编写的向量并行程序,还是并行化编译系统生成的向量并行程序,向量访存操作不可避免地存在冗余。这种冗余是指两条向量访存语句访问的内存空间存在重叠。由于存储性能是计算机性能提高的瓶颈,内存访问在程序运行过程中占用了大量时间,因此对内存访问进行优化,避免冗余访存将能有效提升程序的性能。

2 访存优化相关研究

由于计算机系统中存储器性能是整个系统的瓶颈所在,针对内存的优化成为了提高程序性能的通用手段。硬件方面,设计者在CPU与内存之间添加了容量相对较小但速度相对较快的高速缓存层来缓解存储器与CPU之间的差距,这使得数据局部性成为对程序进行优化的关键因素。当前多数存储方面的优化都是基于这一点。文献[6]提出了一种领域特定语言和编译器,通过一种简洁的方法来自动实现空间和短向量部件的优化;同时,使用一些循环变换方法来解决数据局部性和并行负载均衡的问题。其实现的方式是通过对循环迭代的特殊分块方法来提高局部性和并行性。文献[7]提出了一种通过3个模块来兼顾数据局部性粗粒度并行性以及细粒度并行性的编译架构,在高层程序优化和低层代码生成中间

到稿日期:2014-11-21 返修日期:2015-03-29 本文受国家高技术研究发展计划(863)(2009AA01220),"核高基"重大专项(2009zx10036-001-001)资助。

徐金龙(1985—),男,博士生,主要研究领域为并行编译,E-mail:longkaizh@126.com;赵荣彩(1957—),男,教授,博士生导师,主要研究领域为体系结构、先进编译等;徐晓燕(1979—),女,博士生,主要研究领域为网络安全。

设计了一种可向量化的代码块，通过高层的程序变换架构来重组程序结构，并以此来开发多维的数据重用性。文献[8]指出，循环合并是并行性和局部性的一种折衷手段，过分地采用循环合并可能会阻碍程序并行，这是因为如果这样做，可能会导致大量的依赖；同时，处理器提供了固定数量的硬件预取流缓冲区，过分合并可能无法有效利用硬件预取，而编程者总是希望能够尽可能多地利用预取缓冲区。如果不采用循环合并，那么并行的机会得到了提升，但是这导致了局部性的缺失，降低了预取的利用率。提出了一种同时实现硬件预取流缓冲区优化利用、局部性优化和开发并行性的循环合并模型，能够很好地适用于包含大段代码的程序。文献[7]同时指出，近些年来，基于多面体模型的优化编译器在程序变换方面以及 cache 数据局部性分块上取得了很好的效果。

另外，SIMD 并行性的发掘也会受到一些特殊访存模式的影响。向量化是一种细粒度的并行化，不仅需要依赖关系满足并行需求，对访存模式也有较苛刻的要求。GCC 和 ICC 的向量化模块都尝试支持某些不太复杂的访存模式[9,10]。GCC 最初向量化模块只支持 loop-based 向量化方法，某些伪跨幅访存不能被正确识别，伪跨幅访存是指：语句被孤立分析时，访存不连续，但考虑迭代内并行其访存是连续的，因此 GCC 引入了 SLP 算法来发掘迭代内并行，解决了伪跨幅访存的问题。ICC 提供了一种高效的向量化方法来专门处理访存跨幅为 2 的幂的情况，并改进了数据重组方式，可以充分利用 Intel 处理器提供的跨幅访存指令。

上述的内存相关的优化技术可从数据局部性方面入手提高程序性能，也可从访存模式入手，深入挖掘程序 SIMD 并行性，从而提高程序性能。这些优化都是在生成向量化代码之前的程序变换过程中实施的，并未考虑已有向量代码可能存在的优化空间，本文针对向量代码中的访存指令进行深度优化，通过删除向量访存冗余，提高向量访存对齐性以提升向量代码的执行效率。

3 问题描述

随着各领域需要处理的数据量越来越大，数据密集型应用也得越来越被重视，以访存为主是此类应用的特点[11]。它们的向量版本一般也存在大量的内存访问，因此对内存访问进行优化存在较大空间。

向量版本的程序的生成有两种途径：1)程序员手工编写，这对程序员要求很高，一般程序员难以胜任，因此手工并行结果不仅容易出错，有时还不能得到高效的程序；2)优化编译器自动生成，这对编译器要求较高。这两种方式都不能保证获得足够优秀的向量程序，尤其在向量访存方面，冗余访存不可避免。

本文仅考虑向量读冗余，如果两个相邻的向量读操作对应的内存有重叠，那么就存在向量读冗余。图 1 展示了一个典型的向量访存冗余语句组，每次向量读可以读入 4 个数据元素，显然，$a[i+1]$和 $a[i+6]$被读了两次，$a[i+2]$和 $a[i+5]$被读了 3 次，$a[i+3]$和 $a[i+4]$被读了 4 次，保证程序正确的情况下消除这些重复的访存就是本文的主要目的。

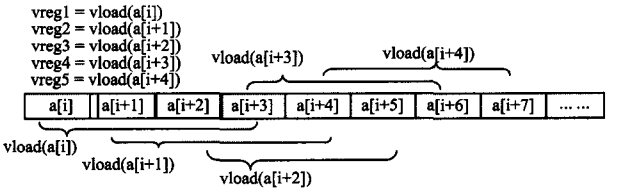


图 1 存在冗余的向量访存语句组

通常，数据重组指令的代价要小于访存指令，对齐访存代价小于非对齐访存，基于上述两点可以进行以下的指令优化。

(1)为了将访存指令数目降到最低，将多条向量访存变换为“最少向量访存＋数据重组”，图 2 是在图 1 的基础上进行优化后的无冗余版本，访存指令条数由 5 条变为 2 条，另有 3 条由数据重组指令实现。

vreg2＝shuffle(vreg1，vreg5，mod1)的语义为：将 vreg1 的后 3 个元素装入 vreg2 的前 3 个槽位，将 vreg5 的第一个元素装入 vreg2 的最后一个槽位。

vreg3＝shuffle(vreg1，vreg5，mod2)的语义为：将 vreg1 的后两个元素装入 vreg3 的前两个槽位，将 vreg5 的前两个元素装入 vreg3 的后两个槽位。

vreg4＝shuffle(vreg1，vreg5，mod3)的语义为：将 vreg1 的最后一个元素装入 vreg4 的第一个槽位，将 vreg5 的前 3 个元素装入 vreg4 的后 3 个槽位。

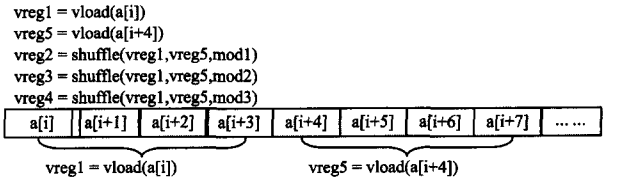


图 2 删除冗余的向量访存

(2)另外，非对齐访存的代价明显高于对齐访存的代价，调整后的“少量的向量访存”尽可能为对齐访存。

4 向量访存指令优化

4.1 向量访存优化需求分析

并不是所有的向量程序都存在访存优化空间，因此首先要确定程序是否存在优化需求。本文涉及的优化包括冗余访存删除和对齐访存优化两方面。

向量访存的冗余分析：一个向量因子(向量操作对应的数据个数)的跨度存在 n 条向量访存语句，若 $n\geq 2$，那么就认为有冗余(n 次向量访存之间有重叠，其中某些访存语句可用其他已经访问过的位置重组来获得)，如{$vload(a[i])$、$vload(a[i+1])$}。

对齐访存优化的目的是尽可能多地生成对齐的访存指令，在对齐信息不可知的情况下，难以实现此优化，此时，可以不考虑对齐访存优化；在对齐信息可知的情况下，非对齐的向量访存都可替换为“对齐访存＋数据重组”。

综上，当存在向量访存冗余或者非对齐向量访存时，向量程序需要访存优化。

4.2 向量访存冗余与对齐优化

本文提供的优化方法主要包括冗余优化和访存优化。通过对访存语句进行分组和确定组内基准访存语句，来获得无冗余的最少访存语句组；同时若对齐信息可知，则在冗余优化的基础上考虑用对齐访存来覆盖非对齐访存空间，实现对齐

访存的最大化，Step1－Step5 是算法的详细描述。

科学计算程序一般具有这样的特点，多个数据进行复杂运算，获得的结果为单个（或极少几个）数据，这就决定了在向量程序中相对于向量读来说，向量写所占比例较小，并且很难存在重叠的写操作，基本不存在优化空间，因此本节所涉及到的访存优化只针对向量读。

Step1　遍历循环中的向量访存操作，收集向量读操作对应的地址范围形成集合 A，如果两次向量读所访问的空间是完全重合的，那么只记录一次。为了方便讨论，我们假设循环中存在的向量读语句组访问了图 3(a)所示的内存空间，后续步骤都基于图 3(a)进行分析及优化。

Step2　对冗余向量访存语句实施分组。

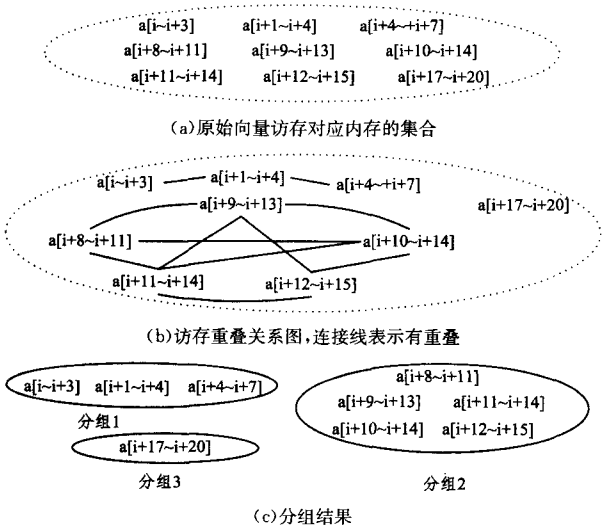
分组目标：

把存在冗余操作的向量访存合成为一组，组间访存不存在冗余。

分组方法：

首先，对收集到的向量访存语句进行两两测试，若两条语句存在内存重叠，并且两条语句之间没有对应内存的写操作，则在它们之间建立一条边。如图 3(a)所示，$a[i\sim i+3]$表示访问到的内存为$(a[i]、a[i+1]、a[i+2]、a[i+3])$，显然 $a[i\sim i+3]$与 $a[i+1\sim i+4]$有重叠。测试完毕后得到如图 3(b)所示的重叠关系图。

其次，根据重叠关系图对访存语句进行分组，图中每个连通分量都对应一个语句子集，这些子集就是本步骤要获得的分组结果，如图 3(c)所示。每个连通分量中边的数目不同，边越多，连线越密，说明访存重叠越多，优化后速度提升的效果越明显。



(a)原始向量访存对应内存的集合

(b)访存重叠关系图，连接线表示有重叠

(c)分组结果

图 3　访存重叠分析及访存分组

Step3　针对每个冗余访存语句组生成基准访存语句组。

访存优化后，多条向量访存将变换为“最少向量访存＋数据重组”的形式，我们将“最少的向量访存”称为基准访存语句组。对基准访存语句组的要求是：1)必须覆盖冗余语句的访存范围；2)尽可能选择对齐的访存语句。通过以下步骤获得基准访存语句组。

Step3.1　获得分组内访存涉及到的内存范围(scope)，如图 3(c)所示，分组 1 的 scope 为 $a[i\sim i+7]$，分组 2 的为 $a[i+8\sim i+15]$，分组 3 的为 $a[i+17\sim i+20]$。

Step3.2　若对齐信息不可知，选取分组内覆盖 scope 的最小向量 load 集合为基准访存语句组(scope_cover_load_set)，针对图 3(c)中的分组 1，这个集合为$\{a[i\sim i+3],a[i+4\sim i+7]\}$；针对图 3(c)中的分组 2，这个集合为$\{a[i+8\sim i+11],a[i+12\sim i+15]\}$，图 3(c)中的分组 3 对应的集合为$\{a[i+17\sim i+20]\}$。若对齐信息可知(假定 $a[0]$对齐)，这个集合中的向量 load 要优先选取对齐访存，若集合中的对齐访存不能覆盖 scope，那么添加其它对齐访存直到满足要求，针对图 3(c)中的分组 3，组内不存在对齐的向量访存，添加覆盖 scope 的对齐访存语句，这个集合最终为$\{a[i+16\sim i+19],a[i+20\sim i+23]\}$。若两个分组 scope_cover_load_set 之间交集不为空，那么将这两个分组合并，获得新的分组。重复执行 Step3.1 和 Step3.2，直至分组不再变化。

Step4　冗余访存语句组的代码生成。

Step3 执行完毕后，得到了新的冗余访存语句组以及对应的基准访存语句组，对于每一组语句进行如下处理：遍历组内的所有访存语句，若该语句属于 scope_cover_load_set，跳过此语句；若不属于 scope_cover_load_set，那么所要读取的数据已经被基准访存语句装载到两个向量寄存器中，可采用向量重组的方式获得，例如图 3(c)中的分组 1，其中 $a[i+1\sim i+4]$可由基准访存语句组中的其他语句合成。上述处理结束后，得到的向量重组语句组记为 vshuffle_stmt_set。scope_cover_load_set 和 vshuffle_stmt_set 就是优化后最终得到的语句组。

Step5　语句调度及合法性检查。

优化后的语句组需要调度，vshuffle_stmt_set 中的语句执行前，对应的基准语句必须已经执行，通常将 scope_cover_load_set 中的语句提升至语句组的最前端，vshuffle_stmt_set 紧随其后，这其实是一种语句重排序，需要进行合法性分析以保证程序语义的正确性。若合法性检测通过，那么调度成功；若不通过，则将程序还原为优化前的形式。

5　测试及分析

本文的访存优化基于开源编译器 Open64[12] 框架实现的向量化编译器 SW-VEC，编译环境为 Linux 操作系统。实验平台 CPU 主频为 2.0GHz，内存为 2GB，SIMD 扩展部件的向量寄存器宽度为 256 位，可以同时处理 8 个整形数据或者 4 个浮点型数据。

本文的实验方法：本文算法的输入为向量形式的源代码或中间表示，通过 SW-VEC 将其进行访存优化并生成优化后的向量程序，再通过基础编译器的二次编译生成二进制代码并在国产服务器上运行，性能指标为加速比(即串行程序的运行时间除以向量化程序的运行时间)。

本文选取了潜水波程序 SWE 和流体力学程序 OPENCFD 进行测试，图 4 是潜水波程序 SW 核心循环的简化版，对应的向量版本如图 5 所示(open64 自动生成)，分析图 5 的向量访存可知，该用例特别适用于本文提出的优化方法。向量版本经过优化后如图 6 所示。流体力学程序 OPENCFD 的核心循环也有类似的访存特征。

```
for(i;i<N;i++){
  vw=X1 * fin[c-1] +X2 * fin[c-2] +X3 * fin[c]+X4 * fin[c+3999]+X5 * fin[c-4001];
  ve=X1 * fin[c+1] +X2 * fin[c+2] +X3 * fin[c]+X4 * fin[c+4001]+X5 * fin[c-3999];
  vn=X1 * fin[c-4000]+ X2 * fin[c-8000]+ X3 * fin[c]+X4 *
```

```
fin[c-4001]+X5 * fin[c-3999];
vs=X1 * fin[c+4000]+ X2 * fin[c+8000]+ X3 * fin[c]+X4 * fin[c+3999]+X5 * fin[c+4001];
vc=X1 * fin[c] +X2 * fin[c-1] +X3 * fin[c+1]+ X4 * fin[c-4000]+X5 * fin[c+4000];
fout[c]=vw+ve+vn+vs+vc; c++;}
```

图 4 潜水波程序(SWE)核心循环的简化版

```
for(i;i<N;i+=4){
vreg1=vld(fin[c-2]); vreg2=vld(fin[c-1]); vreg3=vld(fin[c]);
vreg4=vld(fin[c+1]); vreg5=vld(fin[c+2]);
vreg6=vld(fin[c-4001]); vreg7=vld(fin[c-4000]); vreg8=vld(fin[c-3999]);
vreg9=vld(fin[c+3999]); vreg10=vld(fin[c+4000]); vreg11=vld(fin[c+4001];
vreg12=vld(fin[c+8000]); vreg13=vld(fin[c-8000]);
v_vw=X1 * vreg2+X2 * vreg1+X3 * vreg3+X4 * vreg9+X5 * vreg6;
v_ve=X1 * vreg4+X2 * vreg5+X3 * vreg3+X4 * vreg11+X5 * vreg8;
v_vn=X1 * vreg7+X2 * vreg13+X3 * vreg3+X4 * vreg6+X5 * vreg8;
v_vs=X1 * vreg10+ X2 * vreg12+X3 * vreg3+X4 * vreg9+X5 * vreg11;
v_vc=X1 * vreg3+X2 * vreg2+X3 * vreg4+X4 * vreg7+X5 * vreg10;
v_fout=vw+ve+vn+vs+vc;
vst(v_fout,fout[c]); c+=4;}
```

图 5 潜水波程序(SWE)的向量版本

1)原向量语句组

```
vreg1=vld(fin[c-1]);vreg2=vld(fin[c-1]);
vreg3=vld(fin[c]);vreg4=vld(fin[c+1]);
vreg5=vld(fin[c+2]);
```

2)优化后

```
vrega=vld(fin[c-4]);vreg3=vld(fin[c]);
vregb=vld(fin[c+4]);
vreg1=shuffle+from(vrega vreg3);
vreg2=shuffle+from(vrega vreg3);
vreg4=shuffle+from(vrega3 vregb);
vreg5=shuffle+from(vrega3 vregb);
```

图 6 图 5 第一个向量访存语句组的优化结果

分别测试其串行源程序版本运行时间、普通向量版本运行时间、经过本文算法优化后的向量版本运行时间，得到如表 1 所列的测试结果。

表 1 测试结果——运行时间和加速比(与串行程序对比)

程序名称	串行版本	普通向量版本		访存优化版本	
	运行时间(s)	运行时间(s)	加速比	运行时间(s)	加速比
潜水波(SWE)	213	130	1.63	122	1.73
流体力学(OPENCFD)	7.4	4.5	1.64	3.9	1.90

为了更直观地表述本算法的优势，图 7 给出了加速比的对比图。显然程序进行访存优化后，与普通向量版本相比有了明显的效率提升，潜水波程序优化后相对普通向量版本有 1.07 的加速比，相对串行程序则达到了 1.73；流体力学程序相对普通向量版本有 1.16 的加速比，相对于串行版本则达到了 1.90。数据显示，本文算法对 OPENCFD 程序的效果要好于 SWE，其原因在于 OPENCFD 对应向量版本程序所含的重叠访存更加密集，存在更大的优化空间。

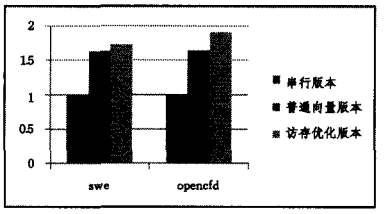


图 7 向量程序相对于串行程序的加速比

综合上述的测试与分析，本文提出的优化方法是有效的，该方法较适用于流体力学、浅水波等类型的科学计算程序。此类程序的特点是：经常同时存在近距离地址空间的访问，一般向量化后经常存在访存的重叠，本文的方法将对其进行进一步优化，消除这些重叠，同时提供对齐优化。

结束语 本文研究了 SIMD 向量代码的访存相关优化，需要指出的是，本文优化的实施阶段是向量代码生成以后。通过向量访存的重叠性(即冗余性)和访存的对齐性来确定是否需要进行相关优化，优化包括冗余优化和访存优化。针对冗余优化，通过对访存语句进行分组和确定组内基准访存语句，来获得无冗余的最少访存语句组；针对对齐优化，在对齐信息可知的条件下，在冗余优化的基础上考虑用对齐访存来覆盖非对齐访存空间，实现对齐访存的最大化。实验数据证明了本方法的可行性和有效性。

由于本文涉及的优化是以减少访存数量和增强访存质量(尽可能为对齐访存)为基本手段的，将生成大量的数据重组，最终语句条数不一定减少，甚至会增多，因此，优化并不是一定能提升程序性能，最终的优化结果是否有效还需要考虑数据重组操作的效率和优化后的语句条数等。面向本文优化的收益分析是下一步需要展开的研究内容。

参 考 文 献

[1] 李春江，黄娟娟，徐颖，等. 典型编译器自动向量化效果评估与分析[J]. 计算机科学，2013，40(4)：41-46
Li Chun-jiang，Huang Juan-juan，Xu Ying，et al. Evaluation and Analysis of Effects of Auto-vectorization in Typical Complier [J]. Computer Science，2013，40(4)：41-46

[2] Allen R，Kennedy K. 现代体系结构的优化编译器[M]. 张兆庆，乔如良，冯晓兵，等译. 北京：机械工业出版社，2004
Allen R，Kennedy K. Optimizing compilers modern architectures [M]. Zhang Zhao-qin，Qiao Ru-liang，Feng Xiao-bing，et al，eds. Beijing：China Machine Press，2004

[3] Larsen S，Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets[C]// Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. 2000：145-156

[4] Boekhold M，Karkowski I，Corporaal H. Transforming and parallelizing ANSI C programs using pattern recognition[C]// Lecture Notes in Computer Science. 1999

[5] Manniesing R，Karkowski I，Corporaal H. Automatic SIMD parallelization of embedded applications based on pattern recognition[C] // Proceedings of 6th International Euro-Par Confe-

rence. 2000:349-356

[6] Henretty T, Veras R, Franchetti F, et al. A stencil compiler for short-vector simd architectures[C]// Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. ACM, 2013:13-24

[7] Kong M, Veras R, Stock K, et al. When polyhedral transformations meet SIMD code generation[J]. ACM SIGPLAN Notices, 2013, 48(6):127-138

[8] Bondhugula U, Gunluk O, Dash S, et al. A model for fusion and code motion in an automatic parallelizing compiler[C]// Proceedings of the 19th international conference on Parallel architectures and compilation techniques. ACM, 2010:343-352

[9] Rosen I, Nuzman D, Zaks A. Loop-aware SLP in GCC[C]// GCC summit. 2007:131-142

[10] Nuzman D, Rosen I, Zaks A. Auto-vectorization of interleaved data for SIMD[J]// ACM SIGPLAN Notices, 2006, 41(6):132-143

[11] 何颂颂,顾乃杰,任开新. 一种面向数据密集型应用的并行程序执行模型[J]. 小型微型计算机系统,2013,34(7):1457-1461
He Song-song, Gu Nai-jie, Ren Kai-xin. Parallel Program Execution Model for Data-intensive Applications[J]. Journal of Chinese Computer Systems, 2013, 34(7):1457-1461

[12] Open64. Overview of the open64 Compiler Infrastructure[EB/OL]. http://open64.sourceforge.net, 2006

(上接第 17 页)

探索了 GPU 程序性能优化技术,对在 GPU 上进行高性能程序设计的经验进行了总结。本文利用 GPGPU-Sim 对 GPU 的存储层次结构进行了模拟,找出了 SM 数量与存储控制器数量之间的最佳配置关系,发现要使系统达到最佳性能,当二者的工作频率相同时,这一比例值为 4 时可以实现。建立了一个性能模型,通过对指令流水线、共享存储器访存、全局存储器访存的定量分析,找出了性能瓶颈,提高了执行速度。指令流水线中,指令类型是根据功能单元数量来划分的,进而对每种类型的存储器进行优化,提高指令吞吐量。对于共享存储器,主要考虑 bank conflicts 对程序性能可能产生的影响,测试其在不同 active warps 并行数量下访存的带宽。而在全局存储器访存的建模过程中,主要考虑合并内存访问对性能造成的影响,通过分析 thread 数量、block 数量、每个 thread 对全局存储器访存事务次数来设置访存带宽。最后,实验部分比较了在 CPU 和 GPU 上不同维度的矩阵乘法执行速度,并对本文的代码的性能和 CUBLAS 代码进行了比较,证明了该模型的实用性,并有效地实现了矩阵乘法的优化。

参 考 文 献

[1] Liu Jie, Chi Li-hua, Jiang Jie, et al. Performance evaluation methodology for massively parallel computer systems[J]. Computer Engineering&Science, 2013, 35(3):25-30

[2] Dongarra J J, Luszczek P, Petitet A. The LINPACK benchmark: Past, present, and future [J]. Concurrency and Computation: Practice and Experience, 2003, 15(9):803-820

[3] SPEC benchmarks [OL]. http://www.spec.org/benchmarks.html

[4] Luszczek P, Dongarra J, Koester D, et al. Introduction the HPC challenge benchmark suite[OL]. http://icl.cs.utk.edu/hpcc/pubs March

[5] Yuan Nan, Zhou Yong-bin, Tan Guang-ming, et al. High Performance Matrix Multiplication on Many Cores[OL]. http://asg.ict.ac.cn/tgm/europar09.pdf

[6] Gunnels J A, Henry G M, Van De Geijn R A. A family of high-performance matrix multiplication algorithms: Lecture Notes in Computer Science, 2001[C]// Proceedings of the International Conference on Computational Science(ICCS'01). Springer-Verlag, 2001:51-60

[7] Long Guo-ping, Fan Dong-rui, Zhang Jun-chao, et al. A performance model of dense matrix operations on many-core architectures: Lecture Notes in Computer Science, 2008[C]// Euro-Par 2008-Parallel Processing. Las Palmas de Gran Canaria, Spain: Springer Berlin Heidelberg, 2008:120-129

[8] Liang Juan-juan. Design and implementation based on GPU BLAS library [D]. Hefei: University of Science and Technology of China, 2010

[9] Baghsorkhi S S, Delahaye M, Patel S J, et al. An adaptive performance modeling tool for GPU architectures[C]// Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2010). ACM, 2010:105-114

[10] Hong S, Kim H. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness[C]// Proceedings of the 36th International Symposium on Computer Architecture(ISCA 2009). 2009:152-163

[11] Yu Zhi-bin, Jin Hai, Zou Nan-hai. Computer architecture software-based simulation[J]. Journal of Software, 2008, 19(4):1051-1068

[12] Cai Jing. Analysis Key Technologies of GPGPU Architecture and Research, Extend on Simulator[D]. Changsha: National University of Defense Technology, 2009

[13] Zhang Shu, Chu Yan-li. GPU High-performance computing [M]. Beijing: China Water & Power Press, 2009

[14] Wang Zhuo-wei, Cheng Liang-lun, Zhao Wu-qing. Parallel Computation Performance Analysis Model Based on GPU[J]. Computer Science, 2014, 41(1):31-38

[15] Wang Zhuo-wei. Research on Performance Optimization for Numerical Computation based on GPU [D]. Wuhan: Wuhan University, 2012

[16] Cheng Si-yuan. Research on Performance Evaluation and Optimization for CPU-GPU Heterogeneous System [D]. Changsha: National University of Defense Technology, 2011

[17] Wai Lun-fung. Dynamic Warp Formation: Exploiting Thread Scheduling for Efficient MIMD Control Flow on SIMD Graphics Hardware [D]. University of British Columbia, 2008

[18] Volkov V, Demmel J W. Benchmarking GPUs to tune dense linear algebra, 2008[C]// 2008 SC International Conference for High Performance Computing, Networking, Storage and Analysis(SC 2008). United States: IEEE Computer Society, 2008

[19] 邹航,王华秋,黄勇. 基于 GPU 加速的彩虹表分析 MD5 哈希密码[J]. 重庆理工大学学报(自然科学版),2013,27(7):61-66
Zou Hang, Wang Hua-qiu, Huang Yong. GPU Accelerated Rainbow Tables Analysis of MD5 Has Password[J]. Journal of Chongqing University of Technology(Natural Science), 2013, 27(7):61-66