计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20180601
58(3):445-457,2021

基于 PPR 模型的稀疏矩阵向量乘及卷积性能优化研究

谢 震[1,2,3] 谭光明[1,2] 孙凝晖[1,2]

[1](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

[2](中国科学院计算技术研究所 北京 100190)

[3](中国科学院大学计算机与控制学院 北京 100049)

(xiezhen@ncic.ac.cn)

Research on Optimal Performance of Sparse Matrix-Vector Multiplication and Convoulution Using the Probability-Process-Ram Model

Xie Zhen[1,2,3], Tan Guangming[1,2], and Sun Ninghui[1,2]

[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

[2](*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)

[3](*School of Computer and Control Engineering*, *University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Performance models provide insightful perspectives to allow us to predict performance and propose optimization guidance. Although there has been much research, pinpointing bottlenecks of various memory access patterns and reaching high performance of both regular and irregular programs on various hardware configurations are still not trivial. In this work, we propose a novel model called probability-process-ram (PPR) to quantify the amount of compute and data transfer time on general-purpose multicore processors. The PPR model predicts the number of instruction for single-core and probability of memory access between each memory hierarchy through a newly designed cache simulator. By using the automatically extracted best optimization method and expectation, we use PPR model for analyzing and optimizing sparse matrix-vector multiplication and 1D convolution as case study for typical irregular and regular computational kernels. Then we obtain best block sizes for sparse matrices with various sparsity structures, as well as optimal optimization guidance for 1D convolution with different instruction sets support and data sizes. Comparison with Roofline model and ECM model, the proposed PPR model greatly improves prediction accuracy by the newly designed cache simulator and achieves comprehensive feedback ability.

Key words performance model; feedback optimization; sparse matrix-vector multiplication; convolution; cache simulator

摘 要 稀疏矩阵向量乘和卷积作为高性能计算的两大计算核心,是非规则和规则访存的典型代表.目前已经做了许多针对性的优化工作,但是对于大量运行着不同指令集和拥有不同计算和访存性能的机器,

收稿日期:2018-09-03;**修回日期**:2020-09-04

基金项目:国家重点研发项目(2018YFB0204400);中国科学院战略性先导科技专项(C 类)(XDC05010100);国家自然科学基金项目(62032023,61972377,61702483)

This work is supported by the National Key Research and Development Program of China (2018YFB0204400), the Strategic Priority Research Program of Chinese Academy of Sciences (C)(XDC05010100), and the National Natural Science Foundation of China (62032023, 61972377, 61702483).

通信作者:孙凝晖(snh@ict.ac.cn)

仍然无法判定在特定的体系结构下导致性能效率无法被完全释放的主要原因及性能瓶颈,同时也很难准确预测出程序在特定机器上可达到的最佳性能.通过使用性能模型方法,建模程序在真实机器上的运行细节,可以得出更加精确的性能预测,并且根据模型输出的反馈信息提出针对性的优化指导.提出了PPR(probability-process-ram)模型,并在一个通用处理器上建模程序内指令执行和数据传输开销,其中包括使用模型预测各种指令数量及内存层次之间的数据传输大小去分析程序各个阶段的性能瓶颈,并且根据模型反馈的信息提出优化方案以及优化后的性能期望.最终使用 PPR 建模和优化 2 个计算核心,同时也比较了与常用的 Roofline 和 ECM 模型的区别.

关键词 性能模型;反馈优化;稀疏矩阵向量乘;卷积;cache 模拟器

中图法分类号 TP391

近些年来,使用性能模型的方法去分析和优化程序已经被广泛地使用.其中以稀疏矩阵向量乘(sparse matrix-vector multiplication, SpMV) $\boldsymbol{y}=\boldsymbol{Ax}$ 为例,作为典型的非规则访存的重要计算核心,该算法被广泛应用在信号处理、图像处理和迭代求解器等科学计算和实际应用中[1].但是在现有的多级存储器层次的体系结构上,稀疏矩阵向量乘的效率一般很低,浮点效率往往低于硬件浮点峰值的10%,其主要原因是复杂的存储器层次结构以及应用数据可重用性差的特征导致 cache 命中率较低,从而凸显了各级存储器之间的访问延迟差异的瓶颈.为了解决这些问题,李肯立等人[2]在 GPU 上使用概率质量函数模型去选择最佳的稀疏矩阵格式,从而构造不同的访存模式去优化数据重用性问题;Li 等人[3]使用建模方法自动调优不同的向量寄存器从而优化矩阵计算开销.但是这些方法都属于粗粒度选择和评判优化方法的优劣,不能细化 SpMV 在特定平台上具体的执行行为.因此如何建模 SpMV 的计算过程以及随机的数据传输特性仍然是性能优化的主要挑战.此外,作为规则访存的典型代表,卷积计算在图像分类、目标检测、图像语义分割和神经网络等领域[4]取得了一系列突破性的研究成果,其强大的特征学习与分类能力引起了广泛的关注.之前的研究表明[5-7],卷积操作在不同的数据规模和体系结构下最优的实现方法差异巨大,从而也给性能模型优化提供了发挥的空间.

此外,得益于性能模型包含的分析和优化的特点,近年来已发展出多种建模方法,其中根据模型是否结合体系结构特征大致可以分为 2 类:

1) 黑盒模型.该方法提取应用特征或者采集运行时数据,拟合或使用机器学习方法对应用程序性能建模.

2) 白盒模型.该方法使用简化的机器模型描述应用程序和硬件的执行关系.

如图 1 所示,最简化的白盒模型是 Konstantinidis 等人[8]提出的 Roofline 模型,该模型描述了应用程序最佳性能与峰值性能、计算访存比和访存带宽之间的关系,预测了不同计算访存比程序可达到的性能上限.同时 Cache-aware Roofline 模型[9]引入的数据局部性规则扩展了 cache 在性能模型中的作用.更细粒度的白盒模型由 Stengel 等人[10]提出的 ECM(execution-cache-memory)模型包含了指令执行和内存层次 2 个部分,该模型把程序运行划分为核内和核外 2 个阶段,对应于程序在 CPU 核内的指令执行和数据在内存层次之间的传输 2 个过程.不过该模型使用 Kerncraft 工具[11]建模程序的指令开销,导致对指令之间数据依赖的建模准确性较低,而且该模型使用的 Pycachesim[11]假定数据在各级 cache 上的缺失数量相同,该假定对于不存在数据复用或者数据完全被复用的程序来说可以达到比较精确的性能预测,然而对于存在一定比例数据重用的非规则访存应用来说,则无法准确预测出性能,更也无法依据模型的输出结果给出具体的优化方案.

因此,为了改进指令数据依赖和数据复用的建模问题,我们提出了 PPR(probability-process-ram)性能模型,该模型加入了处理器流水线指令建模,也加深了内存层次间数据访存建模能力,以解决预测指令流水线执行和非规则访存中数据传输的建模问题.由此我们的模型扩展了对数据依赖和非规则问题的覆盖范围,同时利用各阶段的建模数据反馈开发者优化性能瓶颈.我们的性能建模由 3 个步骤组成:

1) 构建平台,检测和提取硬件参数,包括计算单元个数、访存单元个数、流水线长度、指令发射宽度、

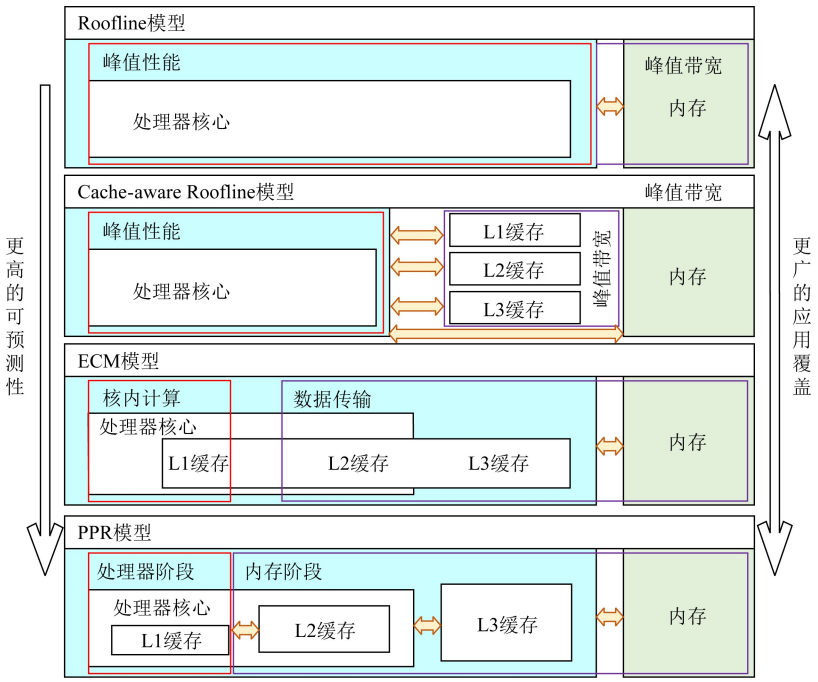

Fig. 1　PPR model and comparison with Roofline, Cache-aware Roofline and ECM

图 1　PPR 模型和 Roofline,Cache-aware Roofline, ECM 模型的对比

指令开销以及各级 cache 大小,cache 分组策略和数据传速率延迟等;

2) 对应用程序构建指令执行有向图,预测指令执行开销,并且标记出规则和非规则数据,通过检测到的配置构建一个全新设计的 cache 模拟器去预测访问数据的传输开销;

3) 分析步骤 2 得到的时间开销,针对所存在的瓶颈提出反馈的优化方案,并且预测出优化后的性能提升.

本文的主要贡献有 3 个方面:

1) 提出了 PPR 性能模型,该模型完整考虑了指令流水、指令执行开销和多级 cache 数据传输.通过细粒度的性能建模得出更精确的性能预测,并且扩展性能建模范围到数据依赖和非规则应用.

2) 为了精确地模拟多级 cache 的数据传输开销,我们设计了一个全新的 cache 模拟器,轻量级地构建于目标机器,并通过模拟应用程序的访存顺序,输出各级 cache miss,进而得到数据的传输开销.

3) 通过计算指令执行和数据传输开销预测出程序性能,并分析建模得到的各阶段时间,找出影响性能的关键瓶颈,反馈开发者对应优化.同时我们也对不同的优化方法建模,预测不同优化策略带来的提升效果,最终指导开发者选择最佳的参数或策略.

1　相关工作

优化稀疏矩阵向量乘方面,前人已经做了很多的工作[12-13],截至目前,已经大量优化技术被提出,如 cache blocking、压缩、重排序以及启发式优化等方法,同时也存在一些问题.具体而言,主要分为 3 个方面:

1) 改变矩阵存储格式优化访存性能,例如 BCSR,ELL,SELL[14]等格式,这些格式存储分块矩阵或对齐数据从而改变访存顺序,减少右端向量的传输次数,优化数据传输开销.所存在的主要问题是格式的选取和转化成本较高,甚至我们也无法准确预测出特定平台上可以获取最佳性能对应的存储格式.

2) 压缩方法,该方法降低矩阵的存储空间,缓解访存带宽的传输压力,但缺点在于压缩和解压的额外开销.

3) 自调优方法,由于不同的矩阵特征往往对应着最佳的矩阵格式和配置参数,通过提取矩阵信息自动选择最优参数以达到最佳性能.但是当前工作[15]提取的矩阵特征有限,主要为行数、列数、非零元个数以及对角数等信息,无法给出十分准确的预测.而所选的格式和参数将决定 SpMV 性能,就 BCSR 格式而言,其主要利用分块技术降低右端向量片段的

数据传输,不过该格式存在多种分块方法,根据不同矩阵的非零元分布情况,多种分块会导致不同的cache miss次数,从而导致完全不同的性能表现.具体而言,过小的分块对数据的重用不够以至于不能显著提升程序性能,而太大的分块则会填充更多零元而大量增加数据集大小,从而导致访存数据增加和性能降低.因此选取合适的分块对SpMV的性能产生至关重要的影响,参数的选取策略也应该是性能模型的主要目的.所有这些都鼓励我们建立一个针对于非规则访存应用的性能模型.

此外近年来卷积神经网络被广泛应用在各个领域(为了加快训练的速度通常选用单精度浮点计算),并取得了一系列突破性的研究成果.而卷积作为卷积神经网络最大的耗时函数,也成为优化的重点.很多工作[16]也使用向量指令或者特定的硬件结构加速计算部分.但是对于开发者而言,如何在不同的指令集支持和数据规模下选择最佳的优化方案仍然是一个亟待解决的问题,这也是我们提出性能模型的主要动机.

2 实验平台

如表1所示,我们的实验基于Haswell微架构的Intel服务器平台,处理器核心支持SSE,AVX,AVX2指令集,每个核心包含一个支持指令乱序执行的端口调度控制器,其中有4个计算端口支持核心每周期2次浮点或者FMA计算,配合使用向量寄存器每个周期最高可达到双精度浮点(DP)16次或单精度浮点(SP)32次的浮点性能,其他4个端口支持访存操作,每个周期支持2个Load和1个Store操作.每个CPU通过4个内存通道与DDR3-1866内存相连.处理器的内存层次由3个片上数据cache组成(32 KB L1D cache, 256 KB L2 cache和30 MB L3 cache).CPU主频被锁定在2.7 GHz.

Table 1 Special Machine Parameters

表1 实验平台及相关参数

参数	细节数据描述
处理器	Intel Xeon E5-2680 V3 12核/24线程 2.5 GHz
缓存	L1缓存: 8路 12×32 KB L2缓存: 8路 12×256 KB L3缓存: 20路 30 MB
缓存行 & 乘加指令	64 B & 256 b
L1缓存带宽	2×32 B/Cycle 读指令 32 B/Cycle 写指令
L2/L3缓存带宽	64 B/32 B/Cycle
内存通道	4通道 DDR3 内存 42.6 GB/s

3 PPR性能模型

我们详细介绍PPR性能模型,该模型分为3个阶段:执行阶段、访存阶段和反馈优化阶段.执行阶段预测计算指令和访存指令在核内的执行开销,访存阶段描述了内存层次之间数据的传输开销,反馈优化阶段则汇总和分析建模信息从而指导瓶颈优化.

3.1 执行阶段

在高性能领域,大量被使用的指令可以分为2类:计算指令和访存指令,其中计算指令表示计算数值所需要的计算操作,访存指令表示移动数据到寄存器的访存操作.这2种指令在处理器核内部独立端口调度运行.当程序需要的数据全部都缓存在L1 cache时,寄存器访问数据不需要额外的数据传输,那么完成所有的计算和访存指令则代表程序的结束.由于不同处理器指令执行能力差别很大,为了清晰地描述模型的功能,当前我们基于Intel Haswell微架构E5-2680 V3搭建该模型,该架构1个周期支持2次浮点或者2次FMA计算以及2个Load和一个Store操作.最终2种指令执行所花费的最大时间将会是执行阶段的开销.

我们详细描述指令建模流程,在建模特定应用程序时,首先分析汇编后的代码得知程序包含的加法指令、乘法指令、FMA指令、Load指令和Store指令数量分别为:A,M,I_{FMA},L,S,则A,M,I_{FMA}条计算指令被调度到执行端口执行,而L,S条访存指令被调度到访存端口执行.同时这些指令之间蕴含着先后顺序和数据依赖关系,因此为了充分模拟指令的执行时间,我们构建一个有向图(DAG)分析

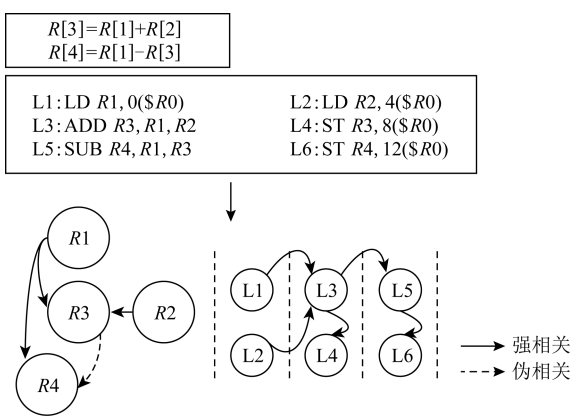

Fig. 2 Schematic for instruction DAG

图2 指令依赖DAG示意图

指令的数据依赖,最终通过模拟 DAG 在硬件上的执行过程建模指令的执行开销.如图 2 所示,以一个简单的程序为例,首先代码被编译为汇编指令,其中包含 2 条浮点计算指令、2 条 Load 指令和 2 条 Store 指令,其次通过分析指令间数据依赖关系构建 S_{DAG},并且以处理器双发射和数据 forwarding 为例,映射出指令执行流水线,最终计算指令花费时间加上等待时间即可得到指令开销.2 类指令执行时间最大值即为执行阶段的时间,通过式(1)得出:

$$T_{process\_phase}=\max(S_{DAG}(A,M,I_{FMA}),S_{DAG}(L,S)). \tag{1}$$

3.2　访存阶段

访存阶段主要建模数据在缓存之间以及主存间的传输开销.我们采用的测试平台是 3 级 cache 和主存的多级内存层次结构,各级 cache 的大小、延迟和传输速率都不尽相同,数据可能出现在缓存或者主存的任何一个内存层次中,当访存指令请求数据时会先查找最近的缓存,如在这一级请求失效则会请求更远的数据层次,最终依次把数据从查询到的缓存传到 L1 cache.与此同时,cache 的数据预取机制可以无阻塞传输可能需要用到的数据到更近的缓存,从而降低数据的访问延迟开销.但是当前 cache 设计只针对规则访存达到很好的预取,因此为了充分模拟 cache 的特性,如图 3 所示,访存阶段首先把数据标记为规则和非规则访存,分 2 种情况建模数据传输开销.

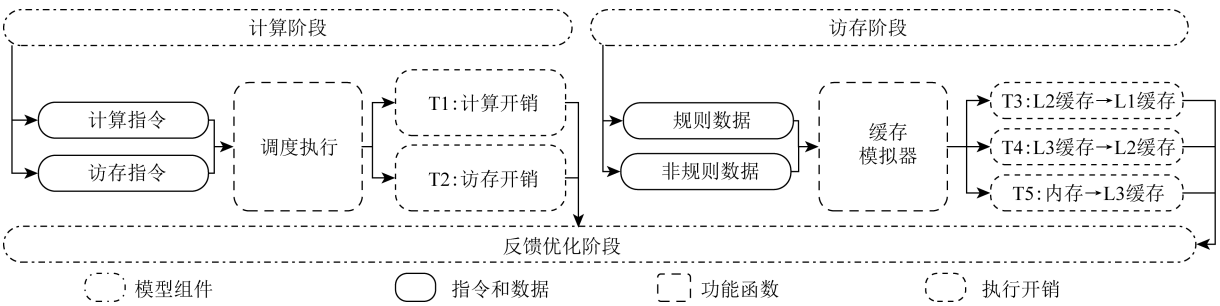

Fig. 3　Three components and execution flow of the PPR model

图 3　PPR 模型的 3 个组件和执行流程

1) 规则访存条件.被访问的数据保存在连续的地址空间或者小跨步的地址空间,且小跨步为不超过一个 cache line 大小的等段式跨越访存.在 CPU 设计中,数据预取占据了很高的地位,主要原因充分利用无阻塞 cache 在多级缓存之间同时传输数据,可以最大化 cache 高带宽特征.为了验证 Haswell 架构的预取策略,我们设计了大量实验得知 L2 cache 和 L3 cache 对规则访存有很高的预取效率,而 L1 cache 为了保障受限的空间不被大量未使用的数据占用,以及避免预测机制去竞争有限的带宽,几乎没有使用预取策略.在访问规则数据时,我们可以假设 L2 cache 以及之后数据访问延迟被预取机制完全隐藏,数据传输路径上的最小带宽即为传输速度.所需要的数据传输时间为

$$T_{RAM\_phase\_for\_regular}=\frac{Regular\_data}{Min\_Bandwidth}, \tag{2}$$

其中,$Regular\_data$ 是规则访存的数据量,$Min\_Bandwidth$ 为数据所在内存层次传输到 L2 cache 的最小带宽.

2) 非规则访存条件.被访问的数据保存在超出跨步大小或者随机的地址空间.但是随机的非规则访存数据可能在同一个 cache line 中,如果假设数据传输大小等于全部访存数据次数个 cache line,预测则会大大高于真实的传输次数,此外,非规则访存通常和规则访存在程序中相伴出现,单纯模拟非规则访存也无法精确预估访存开销.因此为了精确预测访存开销,我们设计一个轻量级的 cache 模拟器建模硬件读取数据时的传输行为.该模拟器的结构如图 4 所示,构建方法为:

① 读取机器的配置.获取机器上各级 cache 的大小和组个数,模拟 cache 间组相联映射构建 cache 表,组内每个 cache line 最初赋值为 invalid,以表明该 cache line 没有缓存任何数据.同时,通过使用修改的 LRU 机制模拟 Intel 的 Smart Cache,并且加入 cache 替换和预取策略.

② 根据应用访存顺序构建访存序列.首先我们把需要访存的数据按照 cache line 大小划分和编号,同时标记需要访问的 cache line 为规则访存或者

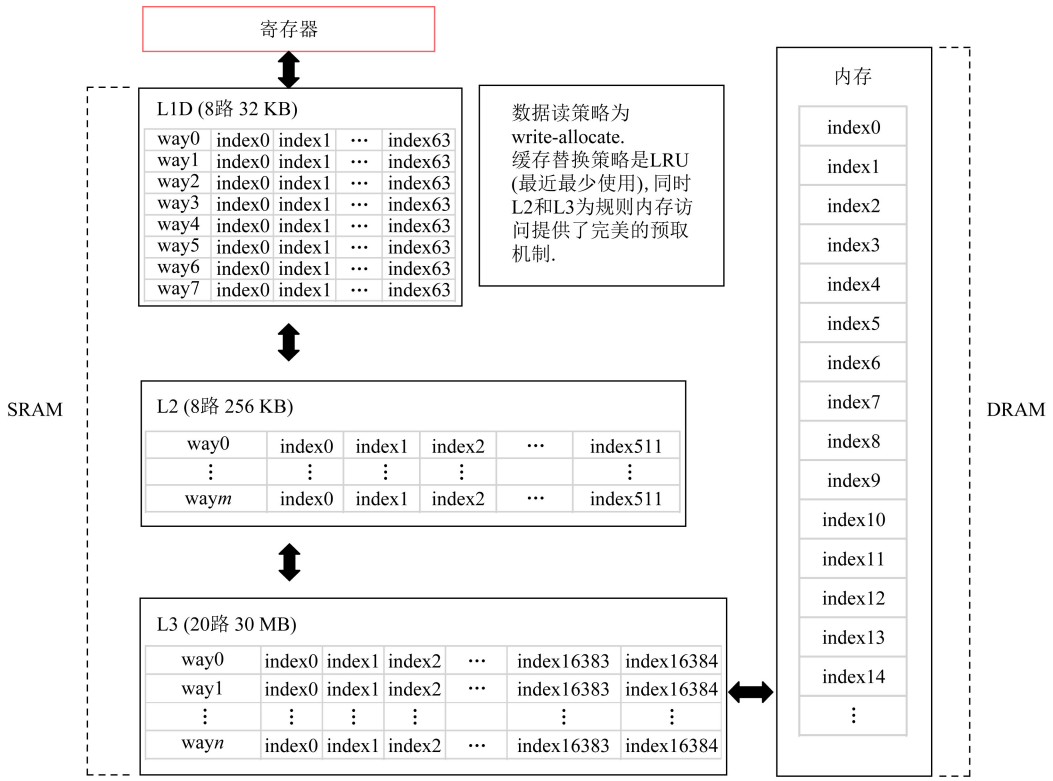

Fig. 4 Multi-level cache simulator

图 4 多层次的 cache 模拟器

非规则访存,例如(index1-0,index3-1,index5-1,index2-0)(0 代表规则,1 代表非规则),该序列主要为了仿真读取 cache line 的访问顺序.

③ 输入访存序列到 cache 模拟器.cache 模拟器依次读取序列的各个 cache 块号.我们标记 cache miss 次数分别是 $L1\_miss$,$L2\_miss$ 和 $L3\_miss$,cache line 的大小为 CL(例如,64 B).模拟器首先如果在 L1 表查找到该序号则使用 LRU 策略标记为最新使用过的块,否则先使 $L1\_miss+1$,然后去 L2 表寻找该 cache 块是否存在,如果存在则把该 cache 块加入到 L1 表中,并且判断该序列是否为规则访存,如果是,则预取 L3 的下一块数据到 L2 表,如果当前数据 L2 表不存在则使 $L2\_miss+1$,然后进一步寻找 L3 表,并且重复上述的查找替换操作.访存开销为

$$T_{\text{RAM\_phase\_for\_irregular}}=\frac{L1\_miss\times CL}{L2\_Bandwidth}+\frac{L2\_miss\times CL}{L3\_Bandwidth}+\frac{L3\_miss\times CL}{Mem\_Bandwidth}. \tag{3}$$

其中 $L2\_Bandwidth$,$L3\_Bandwidth$,$Mem\_Bandwidth$ 分别为 L2 到 L1,L3 到 L2 和主存到 L3 的带宽.

3.3 反馈优化阶段

由于计算阶段和访存阶段可以并发运行:

$$Perf_{\text{PPR}}=((A+M+2I_{\text{FMA}})\times CPU\_freq)/(\max(S_{\text{DAG}}(A,M,I_{\text{FMA}}),S_{\text{DAG}}(L,S)+T(Regular\_data)+T(IrRegular\_data))). \tag{4}$$

其中,$Perf_{\text{PPR}}$ 为基于 PPR 模型的性能预测,CPU_freq 为处理器频率,$Regular\_data$ 为规则访存数据,$IrRegular\_data$ 为非规则访存数据.

两者中的最大值即为程序的执行时间,此式(4)则预测出程序的性能.最后通过分析最大时间作为性能瓶颈,反馈开发者优化相应部分.

4 SpMV 建模

我们挑选了佛罗里达稀疏矩阵库[17]的部分真实矩阵,借助 PPR 模型建模和预测 SpMV 性能,并且找出性能瓶颈,最终针对瓶颈提出针对性的矩阵优化策略.

本节我们首先使用稀疏矩阵库中 13 个矩阵来验证 PPR 模型的正确性,并且与 ECM 模型对比(使用 Kerncraft 和 Pycachesim 工具建模 SpMV),进而在

建模的基础上找到性能瓶颈并提出优化方案.最后,我们比较了优化后模型预测和实际测量的差异.

4.1　测试矩阵

用于建模的稀疏矩阵来自于广泛使用的矩阵集如表 2 所示,这些矩阵具有各种不同的稀疏分布.同时我们的压缩稀疏行(compressed sparse row,CSR)计算内核也使用循环展开和数据对齐等优化,64 b 浮点精度也更符合在实际应用中的效果.

Table 2　Selected 13 Sparse Matrices and Scales

表 2　选择的 13 个稀疏矩阵及其规模

矩阵	非零元个数	行数
12month1	2.26×10^7	1.2×10^4
bone010	7.17×10^7	9.87×10^5
cage15	9.92×10^7	5.2×10^6
circuit5M	5.95×10^7	5.6×10^6
crankseg_2	1.41×10^7	6.3×10^4
Ldoor	4.65×10^7	9.52×10^5
mip1	1.04×10^7	6.6×10^4
rail4284	1.13×10^7	4×10^3
Si41Ge41H72	1.5×10^7	1.86×10^5
torso1	8.5×10^6	1.16×10^5
coPapersCiteseer	3.21×10^7	4.34×10^5
pwtk	1.16×10^7	2.17×10^5
bar	7.4×10^6	1.72×10^5

4.2　性能预测和瓶颈分析

基于 CSR 格式的 SpMV 伪代码如图 5 给出.在"执行阶段",我们首先基于伪代码上构建指令数据依赖图,其中向量 $\boldsymbol{X}$ 的数据读入依赖 $A.col\_ptr$ 的结果,累加 tmp 则依赖于 $A.vals$ 和 $\boldsymbol{X}$ 的结果.由于 Haswell 的 1 个周期可发射 4 条指令,编译器通过使用多个寄存器可以生成相互独立的累加指令,但

```
for (k=0; k<A.numRows; k++){
  double tmp=0.0;
  for (j=A.row_ptr[k]; j<A.row_ptr[k+1];
j++) {
      tmp+=A.vals[j]×X[A.col_ptr[j]];
    }
  B[k]=tmp;
}
```

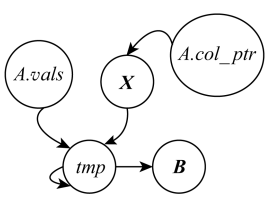

Fig. 5　Pseudo-code for CSR-based SpMV

图 5　基于 CSR 格式的 SpMV 伪代码

是 Load 指令需要的数据依赖无法避免.而对于一个稀疏矩阵,令其行、列和非零元分别为 R,C,NNZ,则整型数组 $A.row\_ptr$ 的大小为 $(R+1)\times4$ B,整型数组 $A.col\_index$ 的大小为 $NNZ\times4$ B,双精度数组 $A.vals$ 的大小为 $NNZ\times8$ B,双精度向量 $\boldsymbol{X}$ 的大小是 $C\times8$ B.伪码中的乘法和加法分别含有 NNZ 条加法和 NNZ 条乘法指令,此外访问 $A.row\_ptr$ 需要 $R+1$ 条 Load 指令,访问 $A.col\_index$ 需要 NNZ 条 Load 指令,访问 $A.value$ 需要 NNZ 条 Load 指令,访问向量 $\boldsymbol{X}$ 需要 NNZ 条 Load 指令,输出向量 $\boldsymbol{B}$ 需要 R 条 Store 指令.对于"执行阶段",通过式(1),在执行计算指令上花费的时间是 NNZ 个周期,由于访存指令的数据依赖,导致访问 $\boldsymbol{X}$ 有 3 个周期的指令延迟,则执行访存指令上花费的时间是 $5\times NNZ$ 个周期.通过循环展开和乱序执行可以使计算指令和访存指令分开执行,则通过式(1),最终花费在执行阶段的时间为 $5\times NNZ$ 个周期.

对于"访存阶段",模型主要建模的非规则访存开销是向量 $\boldsymbol{X}$ 的数据传输.如图 6 所示,我们将矩阵和向量的数据以 cache line 为大小划分为单个小分块,分别标记出规则和非规则访存,并构建访问序列.根据 3.2 节中提到的建模方法,通过读取目标机器的硬件参数来构建 cache 模拟器.模拟器依次读取访问序列并记录缓存未命中次数.图 6 给出了模拟器计算矩阵 1 行的示意流程图.首先,按照访问的顺序依次读取数据到 cache 中,其中模拟器对规则访存的数据实现了充分的预取,对非规则访问的数据则逐级读入.具体来说,模拟器标记行偏移、列偏移和值数组为规则访存数据,标记向量为非规则访存数据,在读取规则访存数据 1 个分片的同时,远端的 cache 也同时传输需要访问的下一个分片到上一级 cache,而对于读取的非规则访存数据,则按照传输顺序,逐步传输到最上层的 cache.同时,该模拟器记录了传输所带来的各级 cache miss,使用式(3)得出访存时间,最终使用式(4)得到了各个矩阵对应的性能. 如图 7 所示,横坐标表示我们挑选的 13 个稀疏矩阵,左侧纵坐标为矩阵的浮点性能,单位 GFLOPS.右侧纵坐标表示各级内存层次中的 cache miss 次数.图 7 的顶部显示了不同形状代表的不同含义.其中测量的 L1,L2,L3 缓存未命中使用了 PAPI[18] 工具统计,以及使用 cache 模拟器模拟的各级 cache miss 在图 7 中用不同形状的图标标记.通过使用 PPR 模型及式(4)计算的性能用圆圈标记,ECM 模型预测

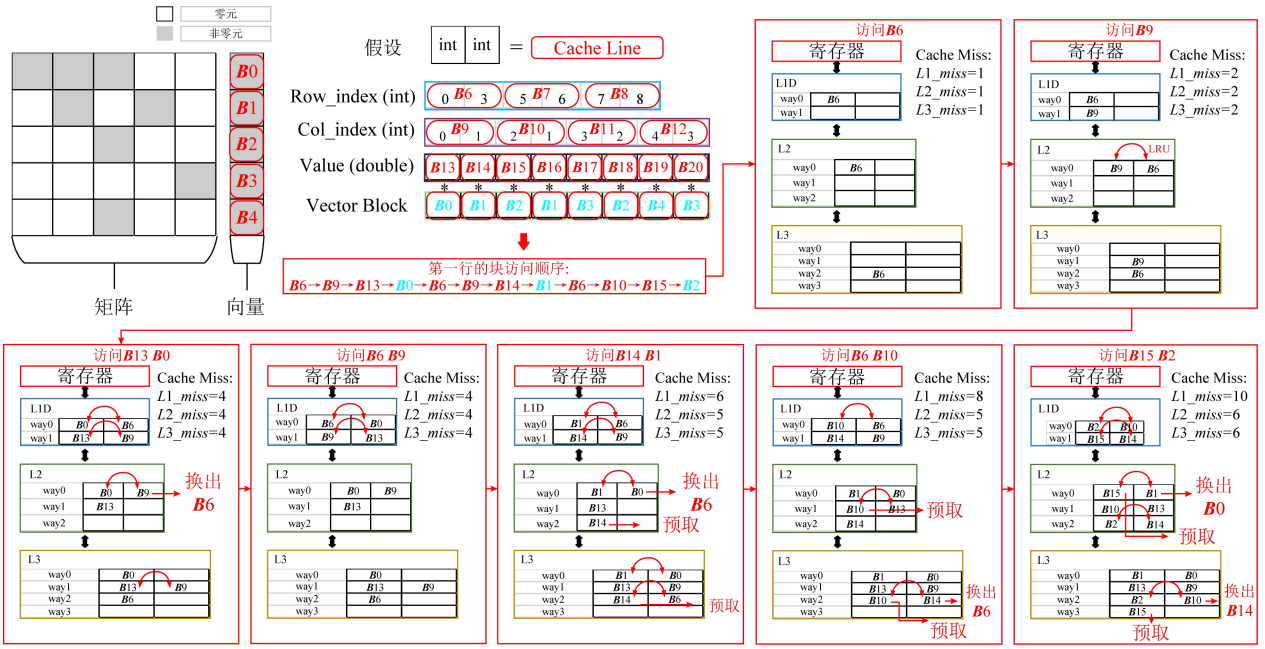

Fig. 6 Schematic work-flow of cache simulator for a fragment of sparse matrix

图 6 用于稀疏矩阵片段仿真 cache 模拟器的执行示意图

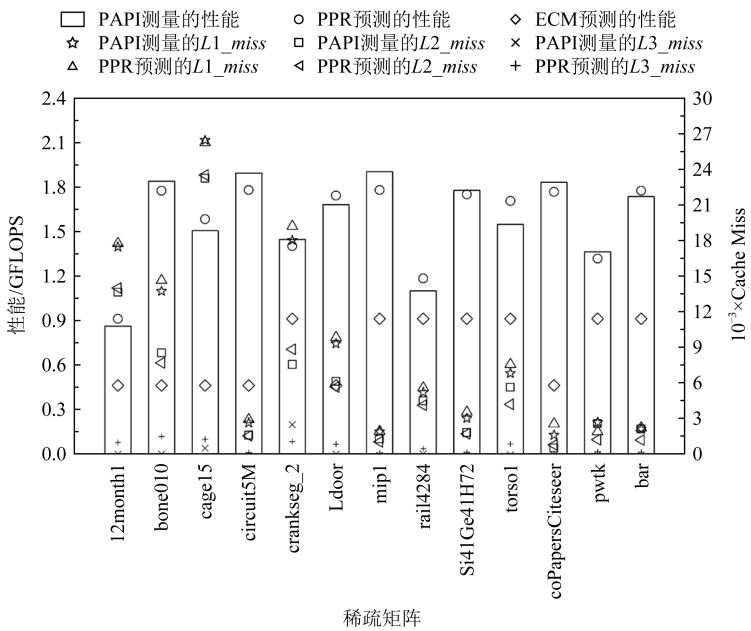

Fig. 7 Comparison of PPR model with actual measurement and ECM model

图 7 PPR 模型与实际测量和 ECM 模型的对比

的性能则用菱形标记.观察发现,使用 cache 模拟器模拟的 cache miss 次数十分接近于 PAPI 测量的 cache miss.相比之下,使用 PPR 计算出的性能相比 ECM 模型预测更接近于真实测量,可以大大提升预测精度.结果也间接地反映了模型和 cache 模拟器的准确性.

从图 7 中明显看出 L2 cache miss 在部分矩阵计算时数量较高,原因是访问 CSR 格式矩阵对应列的向量 $\boldsymbol{X}$ 导致重复的数据传输,并且结合分析 SpMV 的建模结果,我们发现主要的时间花费和瓶颈是非规则数据的传输开销.为了解决这个问题,可以通过改变稀疏矩阵访问模式去增加向量 $\boldsymbol{X}$ 的重用性以及减少数据分片的冗余传输,进而降低数据传输开销,增加浮点运算效率.

4.3　反馈优化

对于降低数据传输开销,矩阵分块是一种用于优化数据重用的典型技术.一个 $m\times n$ 的稀疏矩阵可以在逻辑上被划分为 $r\times c$ 块,并且每个块通常包含至少 1 个非零元素.在处理 SpMV 计算时,每个块可以把向量的一部分保存在寄存器中,用来重用向量 $\boldsymbol{X}$ 中相应的元素,以增加向量数据的局部性.这种优化方法叫作分块的 CSR,是压缩稀疏行存储格式的一种变种,它也简称为 BCSR.该格式连续存储同一块的所有元素,块之间则以行主序存储.BCSR 也降低稀疏矩阵的列索引(每个块 1 个而不是每个非零元 1 个),也减少了内存传输数量.但是,统一的分块大小需要填充显式的零元素,从而导致额外的计算和数据传输,也成为使用 BCSR 格式的挑战.

基于以上原理,我们选择 BCSR 格式并且使用 PPR 模型解决挑选最优分块问题.传统方法一般采用机器学习或统计方法来预测最优分块形状,其预测在一定条件下是有效的,但预测精度经常不稳定.因此我们使用 PPR 模型预测计算指令开销并构建访存序列以建模 BCSR 算法访存开销.通过读取输入矩阵,我们可以得到各种分块下的指令开销和各级 cache miss 次数,预测出每种分块的性能,最终给出了最优的分块方案.如表 3 所示,我们为 13 个矩阵构建 BCSR 模型并给出预测的最佳分块.通过和实际测量的最佳分块对比,模型的预测十分接近于最佳结果,并且带来了平均 124% 的性能加速比.同时我们也考虑了建模开销,通过对 64 种分块大小性能建模和预测,该方法平均花费 12 个 SpMV 时间,低于直接执行的 64 个 SpMV 时间,相比于直接运行得到的性能,我们的方法大大节约了选择最优参数的开销.

Table 3　Optimal Block Sizes and Speedup

表 3　最佳的分块大小及加速比

矩阵	预测的分块大小	最佳分块大小	加速比/%
12month1	1×1	1×1	100
bone010	3×1	3×3	155
cage15	1×1	1×1	100
circuit5M	2×1	2×1	108
crankseg_2	2×1	2×1	121
Ldoor	7×7	7×7	166
mip1	2×2	2×2	138
rail4284	1×2	1×2	102
Si41Ge41H72	2×1	2×1	105
torso1	3×1	3×3	126
coPapersCiteseer	2×1	2×1	107
pwtk	3×1	3×2	145
bar	3×1	3×1	141

5　卷积建模

我们使用 PPR 模型建模规则访存应用-卷积.根据我们的调研发现,直接卷积计算(不变换为矩阵乘法)在不同的数据规模和不同的机器结构下最有效的优化方法差异很大.以一维卷积为例(其他卷积方法都是建立在一维卷积基础上),使用 SSE,AVX 或者 AVX2 指令集进行 4 或者 8 个单精度浮点计算对程序的影响很大,同时对于当前建模机器中向量寄存器 Load 指令而言,对齐数据花费时间是非对齐数据时间的一半,但是使用对齐的数据则会增加数据集的大小,而增加数据集大小在不同规模下会影响 cache miss 数量.所以,我们使用 PPR 模型定量执行指令、访存指令、cache 和内存传输时间,给出模型预期的性能,并且和真实测试到的性能作对比.最后给出在多个指令集支持和不同数据大小上的最佳优化方案.

5.1　原始代码的性能预测和瓶颈分析

从算法 1 可以看出,该卷积 1 次读取 16 个 kernel 数据和 16 个输入数据去执行 16 个加法和 16 个乘法操作.我们可以看出数据具有极好的连续性,第 2 次卷积操作会重用上一次的 15 个数据.16 次卷积计算则导致 16 个单精度浮点数传输,占据 2 个 cache line.

算法 1. Naïve 计算.

输入:输入数据 IN、长度 $length$、核数据 $KERNEL$、核长度 $kernel\_length$;

输出:输出数据 OUT.

① for ($i=0;i<(length-kernel\_length);i=i+1$) do

②　$tmp\leftarrow 0$;

③　for ($k=0,k<kernel\_length;i=i+1$) do

④　$tmp\leftarrow tmp+IN[i+k]\times KERNEL[kernel\_length-k-1]$;

⑤　end for

⑥　$OUT[i]\leftarrow tmp$;

⑦ end for

在16次卷积操作的过程中,分别读取数据 IN 中16个元素更新数据 OUT 的值,其中包含16×16个乘加指令,并且还产生16×16个Load和16个Store指令.通过使用多个寄存器和指令重排,可以完全消除指令之间的数据依赖.对于当前架构,1个周期可以执行2个乘加指令,因此计算指令开销为16×16/2=128个周期,并且1个周期可以执行2个Load指令和1个Store指令,因此访问指令开销为16×16/2=128个周期.访存阶段,由于计算的数据都是规则地址空间,访问数据的延迟可以被预取机制覆盖,因此2个cache line需要消耗L2 cache到L1 cache的2个周期,L3 cache到L2 cache的4个周期,主存到L3 cache的10个周期,最后我们可以推断出不同数据大小的GFLOPS性能.当数据都在L1 cache时,CPU主频被锁定在2.7 GHz,通过式(4)可计算出性能为(16×16×2)/(max(128,128)/2.7)=10.8 GFLOPS.当数据在L2 cache时,通过式(4)可计算出性能为(16×16×2)/(max(128,128+2)/2.7)=10.63 GFLOPS.当数据在L3 cache时,通过式(4)可计算出性能为(16×16×2)/(max(128,128+4)/2.7)=10.47 GFLOPS.当数据在主存时,通过式(4)可计算出性能为(16×16×2)/(max(128,128+10)/2.7)=10.02 GFLOPS.具体指令数量和性能如表4所示:

Table 4 Measured and Predicted Performance Using Algorithm 1

表4 算法1测量及预测的性能

操作	L1容量	L2容量	L3容量	内存容量
加法和乘法/周期	128	128	128	128
Load指令&Store指令&传输开销/周期	128	128+2=130	128+4=132	128+10=138
预测的性能/GFLOPS	10.80	10.63	10.47	10.02
测量的性能/GFLOPS	10.52	10.38	10.17	9.81

从表4可以看出,无论数据在任何一层的内存层次中,卷积的主要瓶颈为执行计算指令和访存执行的开销.我们将介绍2种不同的优化方法,并分析优化所能带来性能.

5.2 优化方法和建模分析

SIMD(single instruction multiple data)指单指令多数据技术,从奔腾II处理器系列引入IA-32架构,并且扩展了128 b SSE指令和256 b AVX,AVX2指令的支持.使用SIMD可以向量化计算和访问多个连续数据,从而大大降低指令执行时间.此外,对于向量化Load和Store指令需要确保被访问的首地址按照16 B对齐,访问未对齐数据所花费的时间是对齐数据的2倍.接下来,我们将使用AVX2指令实现数据非对齐和对齐2个版本,最终给出优化建议.

算法2设计了AVX2指令的非对齐算法,一次计算使用AVX2指令同时操作8个单精度浮点数,但是由于数据填充在连续的空间中,访问数据的间隔为4 B,不能保证16 B的访存对齐,因此必须选择 _mm256_loadu_ps 接口使用非对齐向量访问指令读取非对齐数据.

算法2. AVX展开不对齐计算.

输入:输入数据 IN、长度 $length$、核数据 $KERNEL$、核长度 $kernel\_length$;

输出:输出数据 OUT.

① _m256 kernel_reverse[kernel_length];
② for (i=0;i<kernel_length;i=i+1) do
③ kernel_reverse[i] ← _mm256_broadcast_ss (KERNEL[kernel_length−i−1]);
④ end for
⑤ for (i=0;i<(length−kernel_length/16); i=i+1) do
⑥ acc0,acc1← _mm256_setzero_ps();
⑦ for (k=0;k<kernel_length/16;k= k+1) do
⑧ data_offset←i×16+k×16;
⑨ for (l=0;l<4;l=l+1) do
⑩ for (m=0;m<4;m=m+1) do
⑪ data_block← _mm256_loadu_ps (IN[0]+data_offset+l+ m×4);
⑫ acc0← _mm256_fmadd_ps (kernel_reverse[k×16+l+ m×4],data_block,acc0);
⑬ data_block← _mm256_loadu_ps (IN[0]+data_offset+l+ m×4+8);

⑭　　$acc1 \leftarrow$ *_mm256_fmadd_ps*(*kernel_reverse*[$k\times16+l+m\times4$], *data_block*, *acc1*);

⑮　　end for

⑯　end for

⑰　end for

⑱　*_mm_storeu_ps*(*OUT*[0]+$i\times16$, *acc0*, *acc1*);

⑲ end for

由此可以算出,当前架构 AVX2 指令 1 个周期可以执行 2 个乘加指令,每个乘加指令操作 8 个单精度浮点数,因此计算指令开销为(16×16)/(8×2)=16 个周期.此外由于使用非对齐的访存指令,1 个周期可以执行一个非对齐的 Load 指令和半个 Store 指令,因此访问指令开销为(16×16)/8=32 个周期.数据在内存层次上的传输和算法 1 类似.当数据都在 L1 cache 时,性能则为(16×16×2)/(max(16,32)/2.7)=43.2 GFLOPS.当数据在 L2 cache 时,通过式(4)可计算出性能为(16×16×2)/(max(16,32+2)/2.7)=40.66 GFLOPS.当数据在 L3 cache 时,通过式(4)可计算出性能为(16×16×2)/(max(16,32+4)/2.7)=38.40 GFLOPS.当数据在主存时,通过式(4)可计算出性能为(16×16×2)/(max(16, 32+10)/2.7)=32.91 GFLOPS.性能如表 5 所示:

Table 5　Measured and Predicted Performance Using Algorithm 2

表 5　算法 2 测量及预测的性能

操作	L1 容量	L2 容量	L3 容量	内存容量
加法和乘法/周期	16	16	16	16
Load 指令 &Store 指令 & 传输开销/周期	32	32+2=34	32+4=36	32+10=42
预测的性能/GFLOPS	43.20	40.66	38.40	32.91
测量的性能/GFLOPS	42.63	40.27	38.18	30.53

从表 5 分析发现,算法 2 访存指令成为了性能瓶颈,因此我们优化非对齐的访存指令.算法 3 扩充数据到原有的 4 倍,使得每一次访存地址都按照 16 B 地址对齐,不过也带来了额外的数据传输.

算法 3. AVX 展开对齐计算.

输入:输入数据 *IN*、长度 *length*、核数据 *KERNEL*、核长度 *kernel_length*;

输出:输出数据 *OUT*.

① *_m256kernel_reverse*[*kernel_length*];

② for ($i=0;i<kernel\_length;i=i+1$) do

③　*kernel_reverse*[i]$\leftarrow$*_mm256_broadcast_ss*(*KERNEL*[*kernel_length*$-i-1$]);

④ end for

⑤ *floatin_aligned*[4][*length*];

⑥ for ($i=0;i<4;i=i+1$) do

⑦　*memcpy*(*in_aligned*[i],(*IN*[0]+i),(*length*$-i$)×*sizeof*(*float*));

⑧ end for

⑨ for ($i=0;i<(length-kernel\_length/16);i=i+1$) do

⑩　$acc \leftarrow$ *_mm256_setzero_ps*();

⑪　for ($k=0;k<kernel\_length/16;k=k+1$) do

⑫　*data_offset*$\leftarrow i\times16+k\times16$;

⑬　for ($l=0,m=0;l<4\&\&m<4;l=l+1,m=m+1$) do

⑭　*data_block*$\leftarrow$*_mm256_load_ps*(*in_aligned*[l]+*data_offset*+$l+m\times4$);

⑮　*acc*$\leftarrow$*_mm256_fmadd_ps* (*kernel_reverse*[$k\times16+l+m\times4$], *data_block*, *acc*);

⑯　end for

⑰　end for

⑱　*_mm_storeu_ps*(*OUT*[0]+$i\times16$,*acc*);

⑲ end for

算法 3 改进了访存指令开销,1 个周期可以执行 2 个对齐的 Load 指令和 1 个 Store 指令,因此访问指令开销为(16×16)/(8×2)=16 个周期.为了保证每次访存数据按照 16 B 对齐,我们扩展数据为原始数据的 4 倍,使之 16 次计算数据量增加到了 8 个 cache line.在不同数据规模的性能如表 6 所示:

Table 6 Measured and Predicted Performance Using Algorithm 3

表 6 算法 3 测量及预测的性能

操作	L1 容量	L2 容量	L3 容量	内存容量
加法和乘法/周期	16	16	16	16
Load 指令 &Store 指令 & 传输开销/周期	16	16+2×4=24	16+4×4=32	16+10×4=56
预测的性能/GFLOPS	86.40	57.60	43.20	24.69
测量的性能/GFLOPS	83.56	55.27	43.15	20.86

5.3 优化指导

对比多种优化方案后,我们发现最佳优化方案取决于数据集的大小.在 Haswell 架构上,我们发现,当数据小于 L3 cache 大小时,选择数据对齐的优化算法(算法 3)能够得到较高性能,而随着数据逐渐增大.非对齐算法(算法 2)则更加合适.

6 总 结

我们详细介绍了 PPR 性能模型,并且详细描述了执行阶段、访存阶段及反馈优化阶段.然后我们将该模型应用到 SpMV 和一维卷积上,其中这 2 种算法是非规则访存和规则访存的典型代表.在建模 SpMV 时,我们实例化了 cache 模拟器的工作流程,输出各级 cache miss 次数,进而帮助反馈优化阶段分析各开销的时间占比.在优化时,我们选择了增加数据重用的 BCSR 格式,建模目标矩阵在各种分块大小上的指令和数据传输开销,进而得到最优的分块选择.此外,我们针对一维卷积的原始代码和 2 种优化代码分别建模,详细了解各种优化方法在不同数据量下的性能表现,给出优化建议.该工作现阶段主要是建模单核性能,在此基础上可以进一步提高 PPR 模型针对多核应用的性能建模和预测能力,揭示出多核的性能瓶颈,最终指导并行程序的性能优化.

参 考 文 献

[1] Press W H, Teukolsky S A. Biconjugate gradient method for sparse linear systems [J]. Computers in Physics, 1992, 6(4): 400-410

[2] Li Kenli, Yang Wangdong, Li Keqin. Performance analysis and optimization for SpMV on GPU using probabilistic modeling [J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(1): 196-205

[3] Li Shigang, Hu Changjun, Zhang Junchao, et al. Automatic tuning of sparse matrix-vector multiplication on multicore clusters [J]. Science China Information Sciences, 2015, 58(9): 1-14

[4] Zitová B, Flusser J. Image registration methods: A survey [J]. Image and Vision Computing, 2003, 21(11): 977-1000

[5] Qadeer W, Hameed R, Shacham O, et al. Convolution engine: Balancing efficiency and flexibility in specialized computing [J]. Communications of the ACM, 2013, 41(3): 24-35

[6] Uhl A. Wavelet packet best basis selection on moderate parallel MIMD architectures [J]. Parallel Computing, 1996, 22(1): 149-158

[7] Chakrabarti C, Vishwanath M. Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers [J]. IEEE Transactions on Signal Processing, 1995, 43(3): 759-771

[8] Konstantinidis E, Cotronis Y. A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling [J]. Journal of Parallel and Distributed Computing, 2017, 107(1): 37-56

[9] Ilic A, Pratas F, Sousa L. Cache-aware roofline model: Upgrading the loft [J]. IEEE Computer Architecture Letters, 2013, 13(1): 21-24

[10] Stengel H, Treibig J, Hager G, et al. Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model[C] //Proc of the 29th ACM on Int Conf on Supercomputing. New York: AMC, 2015: 207-216

[11] Hammer J, Eitzinger J, Hager G, et al. Kerncraft: A tool for analytic performance modeling of loop Kernels [J]. Tools for High Performance Computing, 2017, 28(7): 1-22

[12] Buttari A, Eijkhout V, Langou J, et al. Performance optimization and modeling of blocked sparse kernels [J]. International Journal of High Performance Computing Applications, 2007, 21(4): 467-484

[13] Vuduc R, Moon H. Fastsparse matrix-vector multiplication by exploiting variable block structure [J]. High Performance Computing and Communications, 2005, 25(2): 807-816

[14] Kreutzer M, Hager G, Wellein G, et al. A unified sparse matrix data format for efficient general sparse matrix-vector multiply on modern processors with wide SIMD units [J]. SIAM Journal on Scientific Computing, 2014, 36(5): 401-423

[15] Choi J, Singh A, Vuduc R. Model-driven autotuning of sparse matrix-vector multiply on GPUs [J]. ACM SIGPLAN Notices, 2010, 45(5): 115-126

[16] Freeman J, Troyer A. Collaborative textual improvisation in a laptop ensemble [J]. Computer Music Journal, 2011, 35(2): 8-21

[17] Davis T, Hu Yifan. The University of Florida sparse matrix collection [J]. ACM Transactions on Mathematical Software, 2011, 38(1): 1-25

[18] Browne S, Dongarra J, Garner N, et al. A portable programming interface for performance evaluation on modern processors [J]. International Journal of High Performance Computing Applications, 2000, 14(3): 189-204

Xie Zhen, born in 1991. PhD. His main research interests include parallel algorithms, high performance computing and machine learning.

谢　震,1991 年生.博士.主要研究方向为并行算法、高性能计算与机器学习.

Tan Guangming, born in 1980. PhD, professor and PhD supervisor. Senior member of CCF. His main research interests include high performance computing and parallel computing.

谭光明,1980 年生.博士,教授,博士生导师,CCF 高级会员.主要研究方向为高性能计算与并行算法.

Sun Ninghui, born in 1968. PhD, professor and PhD supervisor. Academician of the Chinese Academy of Engineering. His main research interests include high performance computing, parallel algorithm, and file system.

孙凝晖,1968 年生.博士,教授,博士生导师,中国工程院院士.主要研究方向为高性能计算、并行算法及文件系统.