

对每个矩阵而言, x_{ssize} 的最大值由 $srow$ 确定,该缓冲区的大小直接影响了 SpMV 的最终性能.对于 x_{dsize} 的选择,我们期望读取一次的开销与访问一次主存的开销相当.经测试,DMA 传递 32 个元素的开销与访问一次主存的开销相当,故 x_{dsize} 设置为 32.

5 实现及优化

5.1 x 访存优化

稀疏矩阵向量乘中, x 是间接访存,访存行为很不规则,在申威众核处理器上, x 的访存是优化的重点,直接对其最终性能起到决定性的影响. x 的访存有几种方式.

- (1) 所有的 x 直接从主存读取;
- (2) 每个从核通过 DMA 预取部分 x ,其余 x 通过访问主存得到,记为 *static-dma*;
- (3) 动静态 buffer 方式,记为 *static-dynamic*.具体见第 4.2 节.

由于从核访问一次主存约需 200 多拍,方案 1 性能明显很差,所以实际中并未使用.方案 2 和方案 3 中静态 buffer 的大小见第 4.2 节.第 6.2.1 节给出了两种方案的性能对比结果.

另外,加载静态 buffer 的初始位置对 SpMV 的性能也有一些影响.初始位置有两种选择.

- 1) 从当前从核计算的行块的起始位置读取,记为 *start-x-row*;
- 2) 从当前从核计算行块所需的第一个 x 处读取,记为 *start-x-current*.

5.2 负载均衡

稀疏矩阵每行的非零元个数不尽相同,且分布不均.按照图 3(a)中的静态任务划分方法,对有些矩阵会导致从核间负载不均衡,这个负载不均衡来自两个方面.

- 每个从核计算的行块的总非零元个数可能差异较大;
- 每个从核计算的行块中 x 的访存行为可能差异较大.

为了解决负载不均衡的问题,本文还采用了动态任务划分的方式,如图 3(b)中所示.该方式中,从核间协同,通过采用我们自己用原子操作实现的锁来完成.

然而,由于目前的锁实现中需要访问主存,这个代价比较高,所以其性能较差,具体见第 6.2.2 节.

为此,本文对这种调度方式进一步进行了优化,只在第 1 次运行 SpMV 时采用动态调度,并记录每个计算核心所分配的任务,在以后的执行过程中,均按照这种方式来进行任务分配,我们将其称为动-静态任务调度.

5.3 自适应优化

由于实际应用中稀疏矩阵千差万别,非零元的分布方式各不相同.对每一个稀疏矩阵而言,任务分配方式、静态 buffer 大小、静态 buffer 加载的起始位置等均对其性能有着很大的影响,有必要针对该稀疏矩阵选择最优的参数组合.可选的参数如下:

- 调度方式,有两种选择:静态调度和动静态结合的调度方式;
- 静态 buffer 读取的起始位置;
- 静态 buffer 的大小,其最大值受 LDM 限制,每个矩阵均不同,初始值选为 128,每 128 递增.

为了减少搜索开销,本文对 Matrix Market 矩阵集中 57 个不同类型的稀疏矩阵选择不同参数的性能结果进行分析,发现任务调度方式、静态 buffer 读取的起始位置均与静态 buffer 的大小关系不大,据此,本文确定了如图 5 的搜索顺序.

该搜索过程需要大约 3~22 个 SpMV 的时间,但是对于实际应用来说,这个过程可以预先进行,以便于在以后的迭代过程中选用性能最高的 SpMV 实现.



Fig.5 The flowchart of search of optimal parameter

图 5 最优参数搜索顺序图

5.4 双缓冲优化

该处理器从核上支持 DMA 访存与计算重叠,为了验证其有效性,本文设置 LDM 上 *val,col,ptr* 的双 buffer. 图 6 中,上图展示了单 buffer 的计算和访存流程,下图展示了双 buffer 的计算和访存流程.但该异构众核 SpMV 算法中主要以 DMA 操作为主,计算所占的比重很小,该优化对整体的性能影响不大.

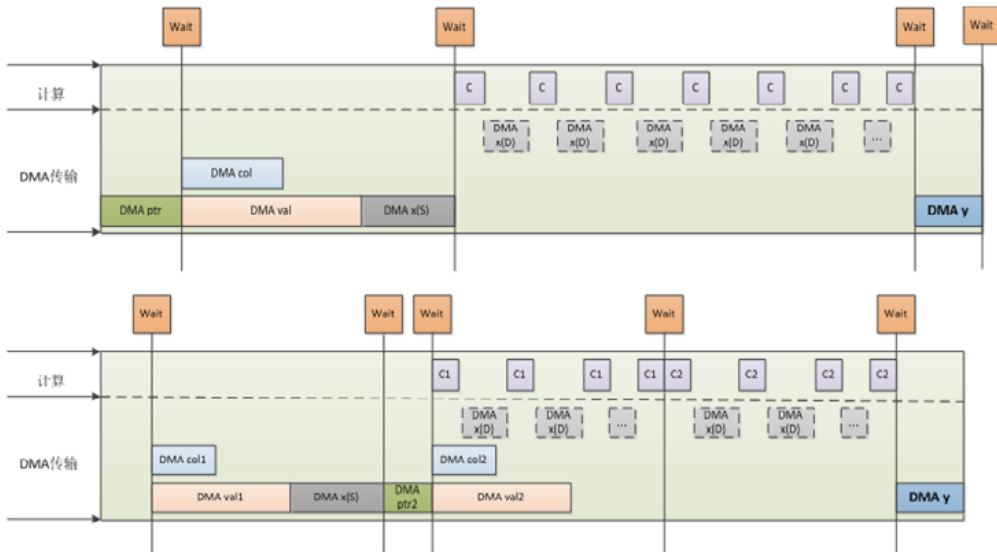


Fig.6 SpMV with CSR format in timeline

图 6 CSR 格式 SpMV 时序图

6 实验结果

6.1 实验平台

我们采用神威“太湖之光”的一个核组作为测试平台,借助高性能线程库 *Athread* 将计算任务异步加载到从核执行.测试矩阵选用了 *Matrix Market* 矩阵集中的矩阵进行测试,矩阵规模从数千到百万,矩阵非零元个数从数万到 100 多万.表 1 中给出了测试矩阵的基本信息.

Table 1 The information of test matrices

表 1 测试矩阵信息表

编号	矩阵名字	矩阵维数(m)	非零元数(nz)
1	bcsstk17	10 974	428 650
2	bcsstk28	4 410	219 024
3	raefsky2	3 242	294 276
4	Linverse	11 999	95 977
5	Cant	62 451	4 007 383
6	s3dkq4m2	90 449	4 820 891
7	fv2	9 801	87 025
8	nemeth01	9 506	725 054
9	LF10000	19 998	99 982
10	af_0_k101	503 625	17 550 675
11	cavity20	4 562	138 187
12	ecology1	1 000 000	4 996 000
13	epb3	84 617	463 625
14	qa8fk	66 127	1 660 579
15	Obstclae	40 000	197 608
16	af_shell3	504 855	17 588 875

注:下节给出的所有测试结果均为计算 50 次 SpMV 的时间.

6.2 测试结果

6.2.1 x 访存优化的对比结果

图 7 中比较了第 5.1 节中提到的方案 2 和方案 3 的性能,从图中可以看出,方案 3 明显优于方案 2,最高加速比可达 21 倍.这是因为方案 3 利用了稀疏矩阵的局部性,动态缓冲区的 x 数据得以重复利用.

图 8 中给出了部分矩阵选用两种加载静态 buffer 的起始位置的性能对比.矩阵 qa8fk,raefsky2,cavity20 选用方案 2 性能较好,而 cant 和 s3dkq4m2 是选用方案 1 性能较好,性能差最大的有 55%,最小的也有 11%.

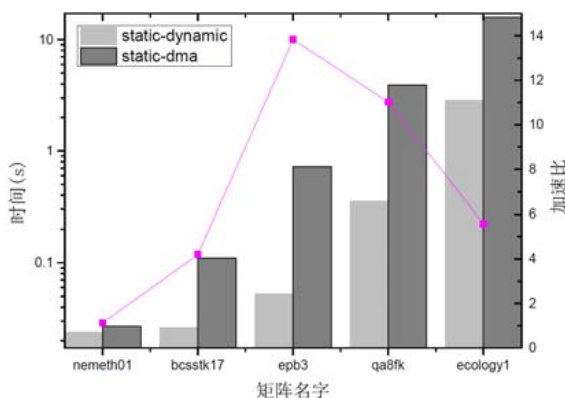


Fig.7 The optimized performance of x loading

图 7 x 访存优化效果对比图

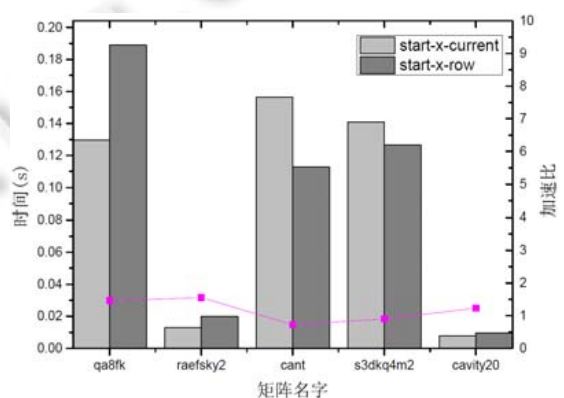


Fig.8 The impact of the start position of loading dynamic/static buffer on performance

图 8 加载动静态 buffer 的初始位置对性能的影响

6.2.2 任务调度

由于稀疏矩阵千差万别,不同的任务调度方式对其优化的效果也不尽相同,图 9 中给出了动-静态调度方式

性能较好的测试矩阵的结果,并将其与静态调度方式进行了对比.其中,两种调度方式均采用从当前所需的第 1 个 x 作为起点加载静态 **buffer**,并且选用了 x_{size} 可选范围内的最优性能.从图中可以看出,最大加速比可以达到 6 倍多,说明不同任务调度方式对某些矩阵的性能有着很大的影响.

第 6.2 节中提到:动态调度时,由于加锁引入了额外的开销.图 10 中比较了采用动态调度进行计算的时间(记为 *dynamic*)与利用动态调度的任务划分方式进行静态分配的计算时间(记为 *static(dynamic)*),其性能约有 10%~40% 的差异.

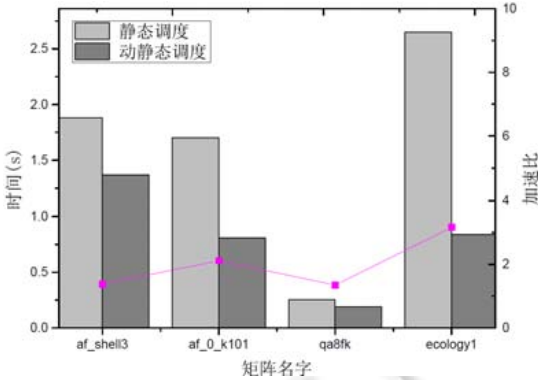


Fig.9 The impact of different scheduling method on performance

图 9 不同的任务调度方式的优化效果

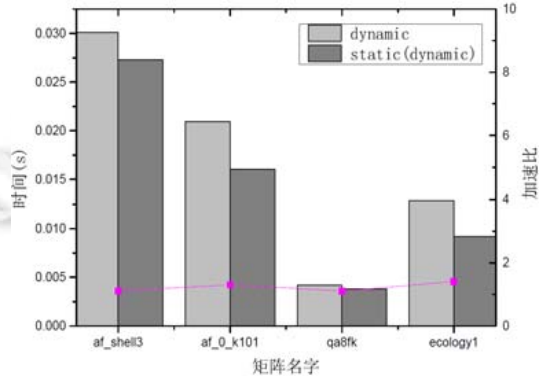


Fig.10 Comparison of the first two calculations using the static (dynamic) method

图 10 动静态调度方式前两次性能对比

6.2.3 自适应优化

图 11 给出了对第 5.3 节中提到的 3 个参数进行自适应优化的性能结果(不含调优时间),并与采用静态调度方式、 $x_{size}=1536$ 、从当前所需的第 1 个 x 进行加载静态 **buffer** 的方法进行了对比.从图中可以看出:自适应优化取得了比较明显的加速效果,平均性能提升为 44%,最大的 *ecology1* 矩阵可达到 6 倍多,这主要是动静态任务调度带来的加速.

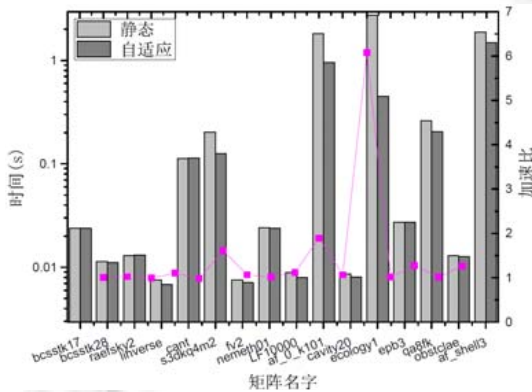


Fig.11 The performance of adaptive optimization

图 11 自适应优化效果图

6.2.4 性能结果

本文对选取的 16 个矩阵进行了测试,图 12(a)展示了分别在主核和从核运行的结果,主核版采用最原始的 CSR 格式 SpMV 实现.可以看出:测试的矩阵相对主核版均有不同程度的性能提升,最高可达 10 倍多,最低也有 4 倍多,平均加速比为 6.51 倍.另外,本文还测试了带宽利用率,总访问量采用公式 $nz \times 12 + (nrow + 1) \times 4 + nrow \times 8 \times 2$ 来计算.图 12(b)给出了测试矩阵的带宽利用率(总带宽按照实测带宽 27.5Gb/s 计算),最高可达 86.09%,最低可达

31.76%,平均带宽利用率为 47%.

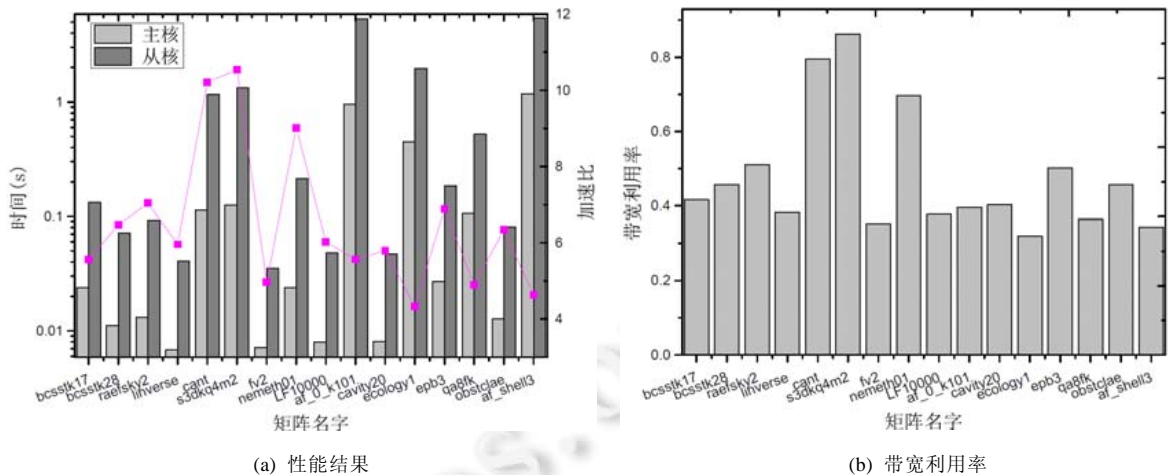


Fig.12 The performance of bandwidth efficiency of test matrices

图 12 测试矩阵的性能及带宽利用率

6.3 测试结果分析

从测试结果来看,从核上 SpMV 的性能与其非零元的分布有很大关系.如果非零元分布的局部性特征比较明显,那么本算法中动静态 buffer 的命中率较高,从而整体性能较好.

目前,整体的带宽偏低,这是因为计算时采用了主核计算 CSR 格式 SpMV 的访存量,而实际在从核计算时,由于 x 的间接访问,必然会引入 x 的额外访存.未来将进一步改进 x 的访存策略,以提升整体性能.

对于一个特定的矩阵,可通过观察分析其非零元的分布规律,设计出特定的 x 的传输方案,这样能尽可能地减少 x 的冗余访存,进而提升带宽利用率和整体性能.

7 结论及下一步工作

SpMV 是众多科学与工程应用中经常调用的核心函数之一,其性能至关重要.而 CSR 格式是使用最广泛的一种稀疏矩阵存储格式.本文针对申威处理器提出了一种 CSR 存储格式 SpMV 操作的通用异构众核并行算法,该算法首先从任务划分、LDM 空间划分方面进行精细设计.为了提升向量 x 的访存命中率,本文提出了一套动静态 buffer 的缓存机制,并分析了加载静态 buffer 起始位置对性能的影响;对某些稀疏矩阵从核间负载不均衡的原因,提出了一套动静态的任务调度方法以实现负载均衡.另外,还分析了该算法中影响 SpMV 性能的几个关键因素,并开展了自适应优化,进一步提升了性能.

对于 CSR 格式的 SpMV,未来还需进一步考虑提升 x 访存命中率的方法,比如利用该处理器的寄存器通信.还可以考虑根据稀疏矩阵的 x 访存特征对其进行分类,对每一类的矩阵采用更加适合的访存方法,以提升自适应性.另外还需考虑新的整体访存量更少的存储格式,以提升整体性能.

References:

- [1] Kourtis K, Karakasis V, Goumas G, Koziris N. CSX: An extended compression format for SpMV on shared memory systems. In: Proc. of the 16th ACM Symp. on Principles and Practice of Parallel Programming. San Antonio, 2011.
- [2] Sun XZ, Zhang YQ, Wang T, Long GP, Zhang XY, Li Y. Crsd: Application specific auto-tuning of SpMV for diagonal sparse matrices. In: Proc. of the 17th Int'l Conf. on Parallel Processing—Vol. Part II (Euro-Par 2011). 2011. 316–327.
- [3] Im EJ, Yelick K. Optimizing sparse matrix computations for register reuse in SPARSITY. In: Proc. of the Int'l Conf. on Computational Science. LNCS 2073, 2001. 127–136.

- [4] Nishtala R, Vuduc R, Demmel, J, Yelick K. When cache blocking sparse matrix vector multiply works and why. In: Proc. of the Applicable Algebra in Engineering, Communication, and Computing. 2007.
- [5] Mellor-Crummey J, Garvin J. Optimizing sparse matrix-vector product computations using unroll and JAM. *Int'l Journal of High Performance Computing Applications*, 2004,18:225–236.
- [6] Willcock J, Lumsdaine A. Accelerating sparse matrix computations via data compression. In: Proc. of the 20th Annual Int'l Conf. on Supercomputing (ICS 2006). New York, 2006. 307–316.
- [7] Kourtis K, Goumas G, Koziris N. Optimizing sparse matrix-vector multiplication using index and value compression. In: Proc. of the 5th Conf. on Computing Frontiers. Ischia, 2008.
- [8] D'Azevedo E, Fahey M, Mills R. Vectorized sparse matrix multiply for compressed row storage format. In: Proc. of the Int'l Conf. on Computational Science. 2005.
- [9] Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In: Proc. of the 2007 ACM IEEE Conf. on Supercom-Putting. Reno, 2007.
- [10] Vuduc R, Demmel J, Yelick K. OSKI: A library of automatically tuned sparse matrix kernels. In: Proc. of the SciDAC 2005, *Journal of Physics: Conf. Series*, 2005.
- [11] Li JJ, Tan GM, Chen M, Sun NH. SMAT: An input adaptive auto-tuner for sparse matrix-vector multiplication. In: Proc. of the 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2013). 2013. 117–226.
- [12] Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis. ACM Press, 2009. 18.
- [13] Vazquez F, Fernandez J, Garzon E. A new approach for sparse matrix vector product on NVIDIA GPUs. *Concurrency and Computation: Practice and Experience*, 2011,23(8):815–826.
- [14] Monakov A, Lokhmotov A, Avetisyan A. Automatically tuning sparse matrix-vector multiplication for GPU architectures. In: Proc. of the Int'l Conf. on High-Performance Embedded Architectures and Compilers. Berlin, Heidelberg: Springer-Verlag, 2010. 111–125.
- [15] Choi JW, Singh A, Vuduc RW. Model-Driven autotuning of sparse matrix-vector multiply on GPUs. *ACM Sigplan Notices*, 2010, 45(5):115–126.
- [16] Ashari A, Sedaghati N, Eisenlohr J, *et al.* An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on GPUs. In: Proc. of the 28th ACM Int'l Conf. on Supercomputing. ACM Press, 2014. 273–282.
- [17] Yan S, Li C, Zhang Y, *et al.* yaSpMV: Yet another SpMV framework on GPUs. *ACM SIGPLAN Notices*, 2014,49(8):107–118.
- [18] Pichel JC, Rivera FF, Fernández M, *et al.* Optimization of sparse matrix-vector multiplication using reordering techniques on GPUs. *Microprocessors and Microsystems*, 2012,36(2):65–77.
- [19] Tang WT, Tan WJ, Ray R, *et al.* Accelerating sparse matrix-vector multiplication on GPUs using bit-representation-optimized schemes. In: Proc. of the Int'l Conf. on High Performance Computing, Networking, Storage and Analysis. ACM Press, 2013. 26.
- [20] Sedaghati N, Mu T, Pouchet LN, *et al.* Automatic selection of sparse matrix representation on GPUs. In: Proc. of the 29th ACM Int'l Conf. on Supercomputing. ACM Press, 2015. 99–108.
- [21] Ashari A, Sedaghati N, Eisenlohr J, *et al.* Fast sparse matrix-vector multiplication on GPUs for graph applications. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2014). IEEE, 2014. 781–792.
- [22] Guo D, Gropp W. Adaptive thread distributions for SpMV on a GPU. In: Proc. of the Extreme Scaling Workshop. University of Illinois at Urbana-Champaign, 2012. 2.
- [23] Liu X, Smelyanskiy M, Chow E, *et al.* Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: Proc. of the ACM Int'l Conf. on Supercomputing. 2013. 273–282.
- [24] Tang WT, Zhao R, Lu M, *et al.* Optimizing and auto-tuning scale-free sparse matrix-vector multiplication on Intel Xeon Phi. In: Proc. of the 2015 IEEE ACM Int'l Symp. on Code Generation and Optimization (CGO). 2015. 136–145.
- [25] Kreutzer M, Hager G, Wellein G, *et al.* A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. *SIAM Journal on Scientific Computing*, 2014,36(5):C401–C423.
- [26] Liu W, Vinter B. CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication. In: Proc. of the ACM Int'l Conf. on Supercomputing. 2015. 339–350.



刘芳芳(1982—),女,山西霍州人,高级工程师,CCF 专业会员,主要研究领域为高性能扩展数学库,稀疏迭代解法器,异构众核并行.



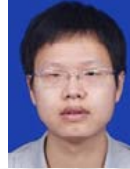
杨超(1979—),男,博士,研究员,博士生导师,主要研究领域为高性能计算,科学与工程计算.



袁欣辉(1989—),男,研究实习员,主要研究领域为国产高性能计算机的性能优化, MPI 库实现与优化.



吴长茂(1974—),男,博士,助理研究员,CCF 专业会员,主要研究领域为并行软件,并行算法,并行卫星成像仿真.



敖玉龙(1990—),男,博士,CCF 专业会员,主要研究领域为大规模并行计算和程序优化.

www.jos.org.cn

www.jos.org.cn