Hiding global communication latency and increasing the arithmetic intensity in extreme-scale Krylov solvers

W. Vanroose
P. Ghysels,
D. Roose,
K. Meerbergen

Report 07.2013.2, Aug 2013

This work is funded by Intel and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).

Hiding global communication latency and increasing the arithmetic intensity in extreme-scale Krylov solvers

W. Vanroose[*] P. Ghysels[*†] D. Roose[‡] K. Meerbergen[‡]

For many HPC codes the major cost lies in the solution of large sparse linear systems [1]. For many problems, the methods of choice for such systems are preconditioned Krylov solvers. However, Krylov solvers are hard to scale to a large numbers of cores due to two main bottlenecks: the inter-node latency and the on-node bandwidth. In this position paper, we review recently proposed techniques to overcome each of these bottlenecks and we put forward possible ways to achieve preconditioned Krylov solvers that efficiently use all the resources on many-core chips and are extremely scalable on massively parallel machines. A key ingredient in our approach is the use of stencil compilers.

Future supercomputers will have a large number of nodes, each being a many-core processor. In addition, the cores will feature vector processing units (VPU) with very long vectors. New algorithms and software should exploit these three levels of parallelism.

On massively parallel machines, global communication should be avoided as much as possible. Global communication is very expensive due to the large latency on the wire, unless it can be overlapped with calculations. In Krylov solvers, there are usually at least two such global communication phases per iteration, used for orthogonalization and normalization of the Krylov base vectors. In the standard formulation of most Krylov methods, there is no possibility to overlap this communication with local work, which leads to a bulk synchronous execution pattern that leaves many resources idle. Recently, *pipelined* Krylov methods [6, 7] reorganized the algorithms with only one global reduction per iteration. The reduction's latency can be overlapped with other work such as the (preconditioned) sparse matrix-vector product ((P)SpMV). While the reduction takes place in the background, new Krylov base vectors can be computed using the (P)SpMV. Only when enough (P)SpMVs have been computed to completely hide the global communication latency, an orthogonalization and normalization step is performed. This deferred orthogonalization obviously changes the numerical properties of the Krylov algorithm. However, since only very few (P)SpMVs are required to completely hide the global latency, numerical stability is mildly affected. This can be remediated by introducing shifts in the (P)SpMV that prevents the base vectors from aligning to the dominant eigenvector and results in an improved Krylov basis.

Pipelined methods lift the main bottleneck for scaling Krylov solvers to extreme numbers of cores and the resulting solver scales as well as the (P)SpMV. For many applications, good scalability for the sparse matrix-vector product (SpMV) can be achieved even for 100k cores, if the problem is partitioned such that there is only local communication. For the preconditioner, there is typically a trade-off between parallelism and efficiency. We expect pipelined methods to be better suited for cheap preconditioners with a high degree of parallelism. 2

Although the (P)SpMV may scale well on a distributed memory system, the on-node performance may still be poor. Within a node, the different threads have to share the available

[*]University of Antwerp - Department of Mathematics and Computer Science, Middelheimlaan 1, B-2020 Antwerp - Belgium (wim.vanroose@ua.ac.be)

[†]Intel ExaScience Lab - Kapeldreef 75, B-3001 Leuven - Belgium

[‡]KU Leuven - Department of Computer Science, Celestijnenlaan 200A - bus 2402, B-3001 Leuven - Belgium

resources such as memory bandwidth, VPUs and so on. It is expected that the current trend will continue after the introduction of new memory technologies, such as 3D stacked memory, although they will temporarily diminish the bandwidth bottleneck. Typically, a SpMV has a very low flop per memory transaction rate, the so-called arithmetic intensity. Hence the performance of a SpMV is severely limited by the memory bandwidth. The same statement usually holds for the AXPY and dot-products and many preconditioners.

One way to reduce memory accesses is to reorganize the algorithms so that a sequence of (P)SpMVs is applied together as in s-step methods [9, 5, 8] or polynomial preconditioners. This reduces the number of writes and reads to and from slower memory. Many existing codes are designed around a SpMV provided by the application programmer in a separate routine, but general purpose compilers are unable to merge these operations together when compiling current libraries. However, when the SpMV is written as a repeated stencil operation, stencil compilers such as [10, 2, 4] can merge the sequence of SpMVs together using time-skewing techniques that increase the arithmetic intensity. For general sparse matrix formats, we are not aware of any library or technique that achieve this goal. Different data formats lead to different opportunities for the implementation of computational kernels. When the matrix is stored in a sparse format, the ordering of the sparse matrix is crucial for good performance of a SpMV on multicore machines [12]. It is an open question how to increase the arithmetic intensity of these SpMVs in a power sequence. Similarly, most current preconditioning software is not designed for such operation. Only when the (P)SpMV has sufficiently high arithmetic intensity, the codes can use the full potential of the VPUs on the many-core chip [11]. Exploiting the dense nature of finite element matrices can also lead to an increase of arithmetic intensity [11].

The challenge is now to combine algorithmic advances for multi-node and many-core systems. Indeed, it is necessary to hide, at the same time, the latencies of the global reductions and increase arithmetic intensity of the (P)SpMV. We believe that for preconditioned Krylov methods this can be achieved by a combination of s-step methods (or polynomial preconditioning) and pipelining. But this requires a redesign of the Krylov subspace algorithms and preconditioning techniques around stencil compilers instead of a user written SpMV routine. The pipelining approach allows for scalable preconditioned Krylov solvers on massively parallel machines whereas the s-step approach can increase arithmetic intensity and hence on-node performance of the (P)SpMV. In the pipelined outer Krylov solve, stability is only mildly affected since the pipelining depth will be small.

Furthermore, multilevel methods can exploit the fact that the cost (both in execution time as in energy) of intra-node data access is significantly lower than between nodes. This implies that there is more freedom in choosing preconditioners on a node, while this is not so for preconditioning the data on different nodes. In multilevel approaches the preconditioner or the entire solver can be reorganized in a tree that is mapped on the hardware. The operations of the solver related to the vertices of the tree near the root require data from nodes that are far away, while the vertices related to a node only use local data. A major challenge is to design multilevel solvers that exploit such structure and that require only few iterations on the root vertex of the tree. In multigrid, the coarse grid solve could be performed by another (multilevel) method. The root level could be related to a Krylov solve over the entire system, or it could be a direct solver on a *small* Schur complement that is solved on each node independently.

Summary: We see an opportunity to leverage the potential of state-of-the-art compiler optimizations to increase the arithmetic intensity and hence the performance of Krylov solvers. Merging as many individual low arithmetic intensity steps as possible can be achieved by using tools such as [2, 3]. Additional opportunities are repeated stencil applications, for instance in polynomial preconditioning, that could also benefit from these stencil compilers. The combination of these techniques with a pipelined Krylov solver, which in theory could scale as well as the SpMV, will result in algorithms that are scalable both within the node and over massively parallel distributed memory machines.

References

[1] *Report on the workshop on extreme-scale solvers: Transition to future architectures*, tech. rep., U.S. Department of Energy, Office of Advanced Scientific Computing Research, 2012.

[2] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, *A practical automatic polyhedral parallelizer and locality optimizer*, in ACM SIGPLAN Notices, vol. 43, ACM, 2008, pp. 101–113.

[3] C. Chen, J. Chame, and M. Hall, *CHiLL: A framework for composing high-level loop transformations*, U. of Southern California, Tech. Rep, (2008), pp. 08–897.

[4] M. Christen, O. Schenk, and H. Burkhart, *Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures*, in Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, IEEE, 2011, pp. 676–687.

[5] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding communication in sparse matrix computations*, 2008 IEEE International Symposium on Parallel and Distributed Processing, (2008), pp. 1–12.

[6] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose, *Hiding global communication latency in the GMRES algorithm on massively parallel machines*, SIAM Journal on Scientific Computing, 35 (2013), pp. C48–C71.

[7] P. Ghysels and W. Vanroose, *Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm*, tech. rep., ExaScience.com 12.2012.1, submitted, 2013.

[8] L. Grigori and S. Moufawad, *Communication avoiding ILU(0) preconditioner*, Rapport de recherche RR-8266, INRIA, Mar. 2013.

[9] M. Hoemmen, *Communication-avoiding Krylov subspace methods*, PhD thesis, University of California, 2010.

[10] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson, *The pochoir stencil compiler*, in Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, ACM, 2011, pp. 117–128.

[11] N. Vannieuwenhoven and K. Meerbergen, *Imf: An incomplete multifrontal lu-factorization for element-structured sparse linear systems*, 35 (2013), pp. A270–A293.

[12] A. N. Yzelman and D. Roose, *High-level strategies for parallel shared-memory sparse matrix–vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, (2013). in press.