Recursive Algebraic Coloring Engine

Poster Summary

Christie Louis Alappat
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
christie.alappat@fau.de

Georg Hager
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
georg.hager@fau.de

Gerhard Wellein
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
gerhard.wellein@fau.de

Olaf Schenk
Institute of Computational Science
Lugano, Switzerland
olaf.schenk@usi.ch

Abstract

Many iterative numerical methods for sparse systems and important building blocks of sparse linear algebra feature strong data dependencies. They may be loop-carried dependencies as they occur in many iterative solvers or preconditioners (e.g., Gauss-Seidel, Kaczmarz) or write conflicts as in the parallelization of building blocks such as symmetric sparse matrix vector or sparse matrix transpose vector multiplication. Scalable, hardware-efficient parallelization of such kernels is known to be a challenge. Most of the typical solutions suffer from low performance on modern hardware, are highly matrix specific, or require tailored matrix storage formats.

In this poster we show a novel method called Recursive Algebraic Coloring (RAC), which achieves high hardware efficiency on modern multi-core architectures and works with simple data formats like compressed row storage (CRS). RAC uses a recursive level-based method that aims at finding optimal permutations while preserving data locality. It is implemented and consolidated into a user-friendly library called Recursive Algebraic Coloring Engine (RACE). A thorough performance analysis shows that RACE outperforms traditional Multicoloring methods and Intel MKL implementations with a factor of 2–2.5×. We are on par with Algebraic Block Multicoloring (ABMC) for small matrices, while for large matrices we gain almost a factor of 1.5–2×.

Keywords Shared memory parallel, Distance-k dependency, Graph algorithm, Sparse matrices, Sparse linear algebra, Performance

1 Motivation

Many sparse linear algebra kernels, such as symmetric sparse matrix-vector multiplication (SymmSpMV) or the Gauss-Seidel iteration, are hard to parallelize due to write-after-write or read-after-write dependencies. In this poster we concentrate on distance-2 dependencies like in SymmSpMV.

SC'18, November 2018, Dallas,TX USA
2018.

2 Related Work

Many solutions to the distance-2 problem have been proposed, such as locking methods, thread-private target arrays [5], special storage formats [1, 9], and matrix reordering, on which we focus here. Multicoloring (MC) [4, 7] and Algebraic Block Multicoloring (ABMC) [6] are two widely used solutions in this area. In [3], MC was applied to the CARP-CG algorithm. However, reordering can impact data access locality, increase the need for synchronization, and effect false sharing, leading to low performance. Here we extend ABMC for distance-2 kernels in order to mitigate these effects.

3 Recursive Algebraic Coloring (RAC) method

The method aims at improving data locality, reduce synchronization, and generate sufficient parallelism while still retaining simple sparse data formats like Compressed Row Storage (CRS).

RAC is a sequential, recursive, level-based algorithm that is applicable to general distance-k dependencies. It is currently limited to matrices with symmetric structure (undirected graph), but possibly nonsymmetric entries. In the following we describe the four steps of the algorithm, which operate on the matrix graph.

1. *Level construction.* A breadth-first search (BFS) [8] is done on the graph to improve data locality [10]. In our experiments we substituted this stage with the slightly more complicated Reverse Cuthill-McKee algorithm (RCM) [2].
2. *Permutation.* The matrix is permuted in the order of levels. We additionally store an array containing the index of to the first element in each level (`level_ptr`).
3. *Distance-k coloring.* Using the levels $L(i)$ one can show that $L(i)$ and $L(i + (k + j))$ are distance-k independent for all $j \geq 1$. In case of $k = 2$ this would mean if we leave at least a gap of two levels between any two groups of levels ($T(a)$ and $T(b)$) they are distance-2 independent. $T()$ is called *level groups* and are formed

by aggregating nearby levels $L()$. One possible configuration can be seen in the figure for the case of two colors; each red *level group* is separated by at least two levels of blue and vice-versa. Obviously there is now a significant load imbalance because of the differently sized level groups.

4. *Load balancing*. The main idea is to resolve the distance-k dependency as required by the algorithm at hand, but also distribute nonzeros evenly across the desired number of parallel threads. This is done by assigning more levels in areas where the levels are small, but fewer levels where they are large, observing the minimum requirement of two levels for maintaining the distance-2 dependency. The algorithm tries to reduce the variance in the number of nonzeros in the two colors by acquiring or giving levels from or to the corresponding *level group*.

If above steps do not lead to sufficient parallelism, recursion is applied. A sub-graph is chosen (typically a *level group*) based on a global load balancing algorithm, which decides that splitting a *level group* into multiple subgroups will be beneficial. Then the four steps above are applied on this sub-graph. The thread that was assigned to the parent sub-graph must spawn two or more subthreads to work on the parts.

4 RACE library

We have implemented the RAC method in a library, the Recursive Algebraic Coloring Engine (RACE). Using RACE implies a pre-processing and a processing phase. In pre-processing the user supplies the matrix, the kernel requirements (e.g., distance-1 or distance-2) and hardware settings (number of threads, affinity strategy). The library generates a permutation and stores the recursive coloring information in a `level_tree`. It also creates a pool of pinned threads to be used later. In the processing phase, the user provides a sequential kernel function, which the library executes in parallel as a callback, using the thread pool.

5 Performance

The test setup for the performance measurements is available in the artifacts description. We show the performance of SymmSpMV for a range of matrices on Intel Ivy Bridge EP and Skylake SP, comparing RACE against ABMC, MC, and MKL implementations. RACE is faster than the alternatives (by up to 2.5×) for almost all matrices, followed by ABMC, MKL, and MC. The advantage of RACE is especially pronounced with large matrices, where data traffic and locality of access is pivotal. For one matrix (nlpkkt-200) we show performance of SymmSpMV together with Roofline limits and data traffic measurements (via LIKWID [11]).

With iterative solvers (like Kaczmarz), matrix reordering causes a change in convergence behavior. The Kiviat graph shows the change in iterations to convergence relative to the serial version (lower is better). Here RACE is on par with ABMC, followed by MC. Hence, we show for Kaczmarz the GFlop/s rate as well as the actual inverse time to solution.

References

[1] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. 2009. Parallel Sparse Matrix-vector and Matrix-transpose-vector Multiplication Using Compressed Sparse Blocks. In *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures (SPAA '09)*. ACM, New York, NY, USA, 233–244. https://doi.org/10.1145/1583991.1584053

[2] Elizabeth Cuthill. 1972. *Several Strategies for Reducing the Bandwidth of Matrices*. Springer US, Boston, MA, 157–166. https://doi.org/10.1007/978-1-4615-8675-3_14

[3] Martin Galgon, Lukas Krämer, Jonas Thies, Achim Basermann, and Bruno Lang. 2015. On the Parallel Iterative Solution of Linear Systems Arising in the FEAST Algorithm for Computing Inner Eigenvalues. *Parallel Comput.* 49, C (Nov. 2015), 153–163. https://doi.org/10.1016/j.parco.2015.06.005

[4] Assefaw H. Gebremedhin, Duc Nguyen, Md. Mostofa Ali Patwary, and Alex Pothen. 2013. ColPack: Software for Graph Coloring and Related Problems in Scientific Computing. *ACM Trans. Math. Softw.* 40, 1, Article 1 (Oct. 2013), 31 pages. https://doi.org/10.1145/2513109.2513110

[5] T. Gkountouvas, V. Karakasis, K. Kourtis, G. Goumas, and N. Koziris. 2013. Improving the Performance of the Symmetric Sparse Matrix-Vector Multiplication in Multicore. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 273–283. https://doi.org/10.1109/IPDPS.2013.43

[6] Takeshi Iwashita, Hiroshi Nakashima, and Yasuhito Takahashi. 2012. Algebraic Block Multi-Color Ordering Method for Parallel Multi-Threaded Sparse Triangular Solver in ICCG Method. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12)*. IEEE Computer Society, Washington, DC, USA, 474–483. https://doi.org/10.1109/IPDPS.2012.51

[7] Mark T. Jones and Paul E. Plassmann. 1994. Scalable Iterative Solution of Sparse Linear Systems. *Parallel Comput.* 20, 5 (May 1994), 753–773. https://doi.org/10.1016/0167-8191(94)90004-3

[8] C. Y. Lee. 1961. An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers* EC-10, 3 (Sept 1961), 346–365. https://doi.org/10.1109/TEC.1961.5219222

[9] Michele Martone. 2014. Efficient Multithreaded Untransposed, Transposed or Symmetric Sparse Matrix-vector Multiplication with the Recursive Sparse Blocks Format. *Parallel Comput.* 40, 7 (July 2014), 251–270. https://doi.org/10.1016/j.parco.2014.03.008

[10] L. Oliker, X. Li, P. Husbands, and R. Biswas. 2002. Effects of Ordering Strategies and Programming Paradigms on Sparse Matrix Computations. *SIAM Rev.* 44, 3 (2002), 373–393. https://doi.org/10.1137/S00361445003820

[11] J. Treibig, G. Hager, and G. Wellein. 2010. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. (2010).