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 type, Kaczmarz solver) or write conflicts as they show up in the parallelization of building blocks such as symmetric sparse matrix vector or sparse matrix transpose vector. Scalable, hardware-efficient parallelization of such kernels is known to be a challenge. Most of the typical schemes currently available to parallelize such kernels suffer from performance issues on modern hardware or are are highly matrix specific or require changes in entire matrix storage format.

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

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

SC'18, November 2018, Dallas,TX USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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.

2 Related Work

This is a long-standing and active area of research. 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, which leads to low performance compared with the Roofline model. Here we extend ABMC for distance-2 kernels in order to mitigate these effects.

3 Recursive Algebraic Coloring (RAC) method

The method aims to improve data locality, reduce synchronizations, and generate sufficient parallelism while still retaining simple sparse data formats like Compressed Row Storage (CRS). It is currently a sequential algorithm.

RAC is a 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]. One could also substitute this stage with more complicated algorithm like 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. Thus one could work on $T(a)$ and $T(b)$ in parallel, but serially within each *level group*. 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 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 code, which the library executes in parallel as a callback function, using the thread pool.

5 Performance

In order to compare the performance we use the test setup as mentioned in the poster and AD sheet. In this poster we show performance of two kernels SymmSpMV and KACZ sweeps on two Intel architectures: modern SkyLake and Ivy Bridge. We compare our results against ABMC, MC and MKL implementations. From the performance measurements it is clear that RACE has an upper hand for almost all the matrices, and next comes ABMC followed by MKL and MC versions. Especially for large matrices where the memory traffic and data locality becomes crucial here RACE clearly outperforms others even getting over 2.5 × speed-up in performance. The reason for this can be clearly seen also with data traffic measurements using hardware counters (here we used LIKWID [11]) as shown for an example matrix.

When it comes to iterative solvers like KACZ it is also important to study the convergence behavior of the method since re-ordering the matrix leads to change in convergence. Relative iterations required to achieve the same absolute error as that of exact kernel (serial KACZ) is shown in the spider plot. Here we see RACE is on par with ABMC and in some case has slight advantage in terms of convergence, while MC method follows after that. Combining the performance and iterations we could then finally arrive at the actual benefit one could achieve from RACE which is shown as inverse time to solution.

6 Future Work

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).

[12] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785