Introduction to communication avoiding linear algebra algorithms in high performance computing

Laura Grigori
Inria Rocquencourt/UPMC

Contents

1 Introduction

High performance computing has become omnipresent in an increasing number of industrial and environmental applications. Getting progress in many of such society-relevant issues depends on the usage of forthcoming generation of peta-, and later exa-, scale supercomputers. However these supercomputers have extremely complex hardware architectures, and most of the current algorithms are not able to efficiently exploit them. They suffer a rapid degradation of performance when increasing the number of processors to a large number. This is due to the exponentially growing gap between the time it takes to perform floating-point operations by one of the processors and the time it takes to communicate its result to another. Due to physical constraints, no hardware solution is expected without a technological revolution. Previous investigations have typically looked at this problem as a scheduling or a tuning problem, however the progress achieved by such approaches is not sufficient. A different perspective on the communication problem is required, addressing it directly at the mathematical formulation and the algorithmic design level, i.e., a level higher in the computing stack than previously. This requires a shift in a way the numerical algorithms are devised, which now need to keep the number of communication instances to a minimum. It thus calls for an entire new generation of original numerical and algorithmic solutions.

Communication avoiding algorithms provide such a novel perspective on designing algorithms that provably minimize communication in numerical linear algebra. In this short document we will describe some of the novel numerical schemes employed by those communication avoiding algorithms.

2 The need for avoiding communication

With the development of new and improved simulation techniques, numerical simulations have become an increasingly useful tool to address an entire host of industrial, environmental and scientific issues, e.g., studies of global warming, or understanding the origin of our universe (to cite only a few examples). It is projected that progress in many of such issues will be tied to the usage of current and forthcoming generation of petascale and exascale supercomputers.

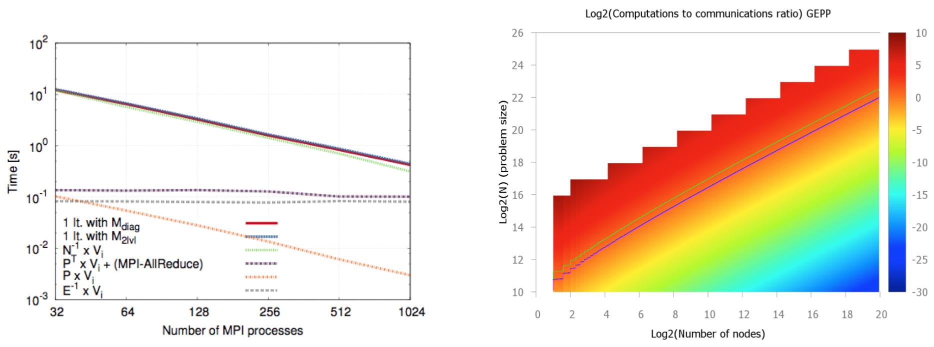
Two challenging trends are discernible here. First, these numerical simulations require progressively larger computing power. This is directly related to the complexity of simulated physical situations, that use large number of time steps, with a 3D simulation volume, which needs to be updated at each step. Though such a problem is encountered very commonly, every specific application has its own particular difficulties. For example, in astrophysics, studying properties of primordial photons produced during the early hot stage of Universe evolution requires the analysis of huge volumes of data. In the so called CMB data analysis, astrophysicists produce and analyze multi-frequency 2D images of the universe when it was 5% of its current age. We describe now a selection of experiments in this area and their expected data volumes, which nearly doubles every year therefore following the Moore's Law. The experiment COBE (1989), collected 10 gigabytes of data, and required 1 Teraflop per image analysis. A more recent experiment, Planck, is a keystone satellite mission who has been developed under auspices of the European Space Agency (ESA). Planck has been surveying the sky since 2010, produces terabytes of data and requires 100 Petaflops per image of the universe. It is predicted that future experiments will collect half petabyte of data, and will require 100 Exaflops per analysis as early as in 2020 [1]. This shows that data analysis in this area, as many other applications, will keep pushing the limit of available supercomputing power for the years to come.

At the same time, the computers are getting faster. Computers that perform 10^{15} floating point operations per second (petascale computers) have become a commonplace in the largest supercomputing centers. The race for faster computers continues, and it is envisaged that exascale performance will be achieved in the following 10-15 years [1]. Unfortunately the architecture of these machines, formed by thousands of multicore processors and accelerators, is becoming progressively extremely complex.

This document discusses one of the main challenges in high performance computing which is the increased communication cost. Several works have shown that there is a gap between the time it takes to perform floating point operations and the time it takes to communicate. This gap is already seen and felt in the current, highly optimised applications, as illustrated by the left panel of Figure 1a, which displays the performance of a linear solver based on iterative methods used in the CMB data analysis application from astrophysics. This performance result is extracted from [2] where a more detailed description of the algorithms can be found. It shows the cost of a single iteration (solid red line), together with its breakdown into time spent on computation (green line) and communication (blue line). These runs were performed on a Cray XE6 system, each node of the system is composed of two twelve-cores AMD MagnyCours and are based on the Mi-

[1] Personal communication with J. Borrill, LBNL, and R. Stompor, Paris 7 University

dapack library, a recent, state-of-the-art, map-making code [2]. It can be seen that the communication becomes quickly very costly, potentially dominating the runtime of the solver when more than 6000 cores are used (each MPI process uses 6 cores). Moreover, it is predicted that the gap will be increasing exponentially in the foreseeable future! Figure 1b displays the performance estimated on a model of an exascale machine of a dense solver based on Gaussian elimination with partial pivoting (GEPP) factorization [3] (see also [3]). The plot displays the computation to communication ratio as a function of the problem size, vertical axis, and the number of used nodes, horizontal. The plot shows two regimes, at the top left corner this is the computation which dominates the run time, while at the bottom right this is the communication. The white region marks the regime where the problem is too large to fit in memory. We note that the communication-dominated regime is reached very fast, even for such a computationally intensive operation requiring n^3 floating point operations (flops) as shown here (where the matrix to be factored is of size $n \times n$.



(a) CMB (b) Dense LU factorization

Fig. 1: Communication bottleneck of two algorithms, a dense linear solver based on the LU factorization (right) and a sparse iterative solver applied to the map-making problem in astrophysics (left).

Communication refers to both data transferred between processors in a parallel computer and data transferred between different levels of memory hierarchy in a sequential computer. In [4] it is observed that the time per flop is increasing per year at a rate of 59%, while the network bandwidth

[2] Available at http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/software/midapack/team.html

[3] Courtesy of M. Jacquelin

of a parallel machine at a rate of 26% per year, and the latency at a rate of only 15% per year. The increase in the flops rate is obtained now by increasing the number of cores per processor. In a sequential computer, the bandwidth is increasing at a rate of 23% per year, while the latency at only 5% per year. This is the memory wall, a problem predicted already in 95 by Wulf and McKee [5]. However, we are also facing now the inter-processor communication wall, in particular given the novel architectures as accelerators and multicore processors. Because of this, most of the algorithms are not able to efficiently exploit these massively parallel machines. They suffer a rapid degradation of performance when increasing the number of processors to a large number. The slow rate of improvement in the latency is mainly due to physical limitations, and we cannot expect that the hardware research will find a solution to this problem soon. Hence the communication problem needs to be addressed and treated at the software level. Energy consumption is another major concern for exascale. As a by-product, communication avoiding algorithms that minimize communication and hence data movement, will also drastically reduce the energy cost of classic algorithms.

2.1 Different previous approaches for reducing communication

Most of the approaches investigated in the past to address this problem rely on scheduling or tuning techniques that aim at overlapping as much as possible communication with computation. However such an approach can lead to an improvement of at most a factor of two. Ghosting is a different technique for reducing communication, in which a processor stores and computes redundantly data from neighbouring processors for future computations. However the dependency between computations in linear algebra operations prevents a straightforward application of ghosting, that is there are cases in which ghosting would require storing and performing on one processor an important fraction of the entire computation. Cache-oblivious algorithms represent a different approach introduced in 1999 for Fast Fourier Transforms [6], and then extended to graph algorithms, dynamic programming , etc. They were also applied to several operations in linear algebra (see e.g. [7, 8, 9]) as dense LU and QR factorizations. This approach relies on a change in the schedule of the algorithm, which is expressed as a recursive operation. In sequential, the data locality is well exploited in an oblivious way at every level of the memory hierarchy. These algorithms are able to reduce the volume of data transferred between different levels of the memory hierarchy, thus addressing the bandwidth problem on one processor. But for linear algebra, they do not reduce the number of messages (with few exceptions), they perform asymptotically more floating-point operations, and they are difficult to parallelize. The design of architecture specific algorithms is another approach that can be used for reducing the communication in parallel algorithms, and there are many examples in the literature of algorithms that are adapted to a given communication topology. However such an algorithm might become very inefficient on a different architecture, and given the lack of portability, this approach is rarely used nowadays.

3 Lower bounds on communication for dense linear algebra

In this section we review recent results obtained on the communication complexity of dense linear algebra operations. These results assume one level of parallelism and take into account the computation, the volume of communication, and the number of messages exchanged on the critical path of a parallel program.

Direct methods find a solution to a given problem in a determined number of steps. In the case of dense factorizations, the number of flops performed is on the order of n^3 for matrices having n columns and n rows. Direct methods (as LU factorization for solving linear systems and QR factorization for solving least squares problems) are known to be very robust and they are used on problems that are difficult to solve by iterative methods.

One notable previous theoretical result on communication complexity is a result derived by Hong and Kung [10] providing lower bounds on the volume of communication of dense matrix multiplication for sequential machines. These bounds are extended to dense parallel matrix multiplication in [11] (with a different approach used for the proofs). Recently we have shown in [12] that these bounds hold for LU and QR factorizations (under certain assumptions) and that they can be used to also identify lower bounds on the number of messages. General proofs that hold for almost all direct dense linear algebra are given in [13]. Consider a matrix of size $m \times n$ and a direct dense linear algebra algorithm as LU, QR, or rank revealing QR factorization. On a sequential machine with fast memory of size M, a lower bound on the number of words and on the number of messages communicated between fast and slow memory during the considered direct linear algebra method is:

$$\# \text{ words} \geq \Omega\left(\frac{mn^2}{\sqrt{M}}\right), \;\; \# \text{ messages} \geq \Omega\left(\frac{mn^2}{M^{3/2}}\right). \qquad (1)$$

where $\#words\_moved$ represents the volume of communication and $\#messages\_send$ represents the number of messages sent by at least one processor. The bounds can be obtained by using the Loomis-Whitney inequality, as proven in [11, 13], which allows to bound the number of flops performed given an amount of data available in a memory of size M. Equation 1 can be used to derive bounds for a parallel program performing n^3 operations and running on P processors. Given a dense matrix of size m-by-n, and assuming that at least one processor does mn^2/P floating point operations, and that the size of the memory of each processor is on the order of mn/P, the lower bounds become:

$$\# \text{ words} \geq \Omega\left(\sqrt{\frac{mn^3}{P}}\right), \;\; \# \text{ messages} \geq \Omega\left(\sqrt{\frac{nP}{m}}\right). \qquad (2)$$

These lower bounds allow us to identify that most of the existing algorithms as implemented in well-known numerical libraries as ScaLAPACK and LAPACK do not minimize communication.

New communication avoiding algorithms have been introduced in the recent years that attain the lower bounds on communication (sometimes only up to polylogarithmic factors) and are as stable as classic algorithms. The communication avoiding LU factorization is presented in [14, 15] while the communication avoiding QR factorization is introduced in [12, 16] and further improved in [17]. A communication avoiding rank revealing QR factorization is presented in [18] while an LU factorization more stable than Gaussian elimination with partial pivoting is presented in [19]. These algorithms have significantly lower latency cost in the parallel case, and significantly lower latency and bandwidth costs in the sequential case, than conventional algorithms as for example implemented in LAPACK and ScaLAPACK. We note that although these new algorithms perform slightly more floating point operations than LAPACK and ScaLAPACK, they have the same highest order terms in their floating point operation counts (except for rank revealing QR). In practice, when used with advanced scheduling techniques, the new algorithms lead to important speedups over existing algorithms [20, 21].

Let's now give an example of classic algorithms that do not attain the lower bounds on communication. Several direct methods of factorization require some form of pivoting to avoid division by small numbers, or preserve stability. The classic pivoting schemes, as partial pivoting in LU factorization or column pivoting in rank revealing QR, imply that the subsequent algorithm communicates asymptotically more than the lower bounds require. For a machine with one level of parallelism, the number of messages exchanged is on the order of n, where n is the number of columns of the matrix, while the lower bound on number of messages in equation 2 is on the order of $\sqrt{P}$ (for square matrices), where P is the number of processors used in the algorithm. Hence in this case minimizing communication requires to invent novel pivoting schemes. There are examples in the literature of pivoting schemes, as for example proposed by Barron and Swinnerton-Dyer in their notable work [22] that minimize communication on sequential machines. At that time the matrices were of dimension 100-by-100 and the pivoting scheme was stable. But as shown in [15], this method can become unstable for sizes of the matrices we encounter nowadays.

The solution that we have developed for the LU factorization is the following. A typical factorization algorithm progressively decomposes the input matrix A into a lower triangular matrix L and an upper triangular matrix U, by traversing blocks of columns (referred to as panels) of the input matrix. At each step of the factorization, a panel is factored, and then the trailing matrix is updated. The classic LU factorization based on partial pivoting (GEPP) factors the panel column by column, and for each column the element of maximum magnitude is permuted to the diagonal position to ensure stability of this method, and this leads to a number of messages exchanged at least equal to n, the number of columns of the input matrix. CALU is based on a novel pivoting strategy, tournament pivoting, which performs the panel factorization as following. A preprocessing step plays a tournament to find at low communication cost b pivots that can be used to factor the entire panel, where b is the panel width. Then the b rows are permuted into the first positions and the LU factorization with no pivoting of the entire panel is performed. The preprocessing step is performed as a reduction operation where the reduction operator is the selection of b pivot rows using GEPP at each node of the reduction tree. This strategy has the property that the communication for computing the panel factorization does not depend on the number of columns, but depends only on the number of processors. The overall number of messages of CALU is $\sqrt{P}log(P)$, hence this algorithm attains the lower bounds on communication modulo a logarithmic factor. In practice, the size of the matrix n is larger by order of magnitudes than the number of processors P.

References

[1] M. Snir, W. Gropp, and P. Kogge, "Exascale research: Preparing for the post–moore era," tech. rep., University of Illinois at Urbana-Champaign, https://www.ideals.illinois.edu/handle/2142/25468, 2011.

[2] L. Grigori, R. Stompor, and M. Szydlarski, "A parallel two-level preconditioner for cosmic microwave background map-making," Proceedings of the ACM/IEEE Supercomputing SC12 Conference, 2012.

[3] L. Grigori, M. Jacquelin, and A. Khabou, "Performance predictions of multilevel communication optimal lu and qr factorizations on hierarchical platforms," in In Proceedings of International Supercomputing Conference, LNCS, 2014.

[4] S. L. Graham, M. Snir, and C. A. Patterson, eds., Getting Up To Speed: The Future Of Supercomputing. Washington, D.C., USA: National Academies Press, 2005.

[5] W. Wulf and S. McKee, "Hitting the wall: Implications of the obvious," ACM SIGArch Computer Architecture News, vol. 23, no. 1, pp. 20–24, 1995.

[6] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, "Cache-oblivious algorithms," In FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999. IEEE Computer Society.

[7] S. Toledo, "Locality of reference in LU Decomposition with partial pivoting," SIAM J. Matrix Anal. Appl., vol. 18, no. 4, 1997.

[8] F. Gustavson, "Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms," IBM Journal of Research and Development, vol. 41, no. 6, pp. 737–755, 1997.

[9] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kagstrom, "Recursive blocked algorithms and hybrid data structures for dense matrix library software," SIAM Review, vol. 46, no. 1, pp. 3–45, 2004.

[10] J.-W. Hong and H. T. Kung, "I/O complexity: The Red-Blue Pebble Game," in STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, (New York, NY, USA), pp. 326–333, ACM, 1981.

[11] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," J. Parallel Distrib. Comput., vol. 64, no. 9, pp. 1017–1026, 2004.

[12] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," Tech. Rep. UCB/EECS-2008-89, UC Berkeley, 2008. LAPACK Working Note 204.

[13] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in linear algebra," SIAM J. Matrix Anal. Appl., 2011.

[14] L. Grigori, J. W. Demmel, and H. Xiang, "Communication avoiding Gaussian elimination," Proceedings of the ACM/IEEE SC08 Conference, 2008.

[15] L. Grigori, J. Demmel, and H. Xiang, "CALU: a communication optimal LU factorization algorithm," SIAM Journal on Matrix Analysis and Applications, vol. 32, pp. 1317–1350, 2011.

[16] J. W. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," SIAM Journal on Scientific Computing, no. 1, pp. 206–239, 2012. short version of technical report UCB/EECS-2008-89 from 2008.

[17] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H. D. Nguyen, and E. Solomonik, "Reconstructing householder vectors from tall-skinny qr," in In Proceedings of IEEE International Parallel and Distributed Processing Symposium IPDPS, 2014.

[18] J. Demmel, L. Grigori, M. Gu, and H. Xiang, "Communication-avoiding rank-revealing qr decomposition," SIAM Journal on Matrix Analysis and its Applications, 2014. In press.

[19] A. Khabou, J. Demmel, L. Grigori, and M. Gu, "Communication avoiding lu factorization with panel rank revealing pivoting," SIAM Journal on Matrix Analysis and Applications, vol. 34, no. 3, pp. 1401–1429, 2013. preliminary version published as INRIA TR 7867.

[20] S. Donfack, L. Grigori, and A. K. Gupta, "Adapting communication-avoiding LU and QR factorizations to multicore architectures," Proceedings of IPDPS, 2010.

[21] S. Donfack, L. Grigori, W. D. Gropp, and V. Kale, “Hybrid static/dynamic scheduling for already optimized dense matrix factorization,” IEEE International Parallel and Distributed Processing Symposium IPDPS, 2012.

[22] D. W. Barron and H. P. F. Swinnerton-Dyer, “Solution of Simultaneous Linear Equations using a Magnetic-Tape Store,” Computer Journal, vol. 3, no. 1, pp. 28–33, 1960.