Mixed Precision Methods for Convergent Iterative Schemes

Robert Strzodka
Stanford University, Max Planck Center
strzodka@stanford.edu

Dominik Göddeke
University of Dortmund
dominik.goeddeke@math.uni-dortmund.de

1. INTRODUCTION

Most error estimates of numerical schemes are derived in the field of real or complex numbers. From a computational point of view this assumes infinite precision. For the implementation on a computer, the infinite number fields are quantized into a finite set of values. Numerical stability analysis of the schemes then reveals how sensitive they react to distortions of the data, introduced by the data quantization and rounding. However, precise quantitative analysis of complex schemes with a clear dependence on the quantization parameters is very difficult. In particular, for iterative schemes which feed the results of one iterative step as input into the next step, and possibly execute hundreds or even thousands of iterations, there is no easy way to obtain reliable error bounds in the long run.

These difficulties often lead to the demand for high precision arithmetic in hardware processors, in the hope that the finer quantization will result in higher accuracy of the final result. While this is true in many cases, in general, there is no monotonic relation between the precision of the quantization and the final accuracy, i.e. increasing the computational precision can lower the accuracy of the result. In practice, however, increasing the precision often is successful, so that implementations tend to favor the highest precision format supported by hardware, without much consideration if it is really necessary. Accordingly, high precision formats are used extensively even in the most performance critical pieces of code, where several parallel low precision computations could lead to significant speedup instead. We suggest to use mixed precision methods to overcome this problem.

2. CONVERGENT ITERATIVE SCHEMES

Given the series of vectors $\mathbf{x}^k \in \mathbb{R}^{N_x}, \mathbf{p}^k \in \mathbb{R}^{N_p}$ we define the corresponding series of matrices $\mathbf{X}^k \in \mathbb{R}^{N_x \times k+1}, \mathbf{P}^k \in \mathbb{R}^{N_p \times k+1}$ containing the vectors from 0 to k. With multi-dimensional functions F_x, F_p we may write down the general iterative scheme:

$$\begin{aligned}(\mathbf{x}^0, \mathbf{p}^0) &:= \text{initial values},\\ \mathbf{x}^{k+1} &:= F_x(\mathbf{X}^k, \mathbf{P}^k),\\ \mathbf{p}^{k+1} &:= F_p(\mathbf{X}^{k+1}, \mathbf{P}^k).\end{aligned}$$

We assume that the series $(\mathbf{x}^k)_k$ converges to a unknown limit $\mathbf{x}^*$. However, we continue the iterations only for $k_{\max}$ steps, which depends upon some error estimator. Thus, we obtain the approximate solution $\mathbf{x}^{k_{\max}}$. The $\mathbf{p}^k$ are auxiliary vectors used by F_x and F_p.

With a new function G, the iteration over $\mathbf{x}^k$ can be re-written as

$$\begin{aligned}\mathbf{x}^{k+1} &:= \mathbf{x}^k + G(\mathbf{X}^k, \mathbf{P}^k), \qquad (1)\\ G(\mathbf{X}^k, \mathbf{P}^k) &:= F_x(\mathbf{X}^k, \mathbf{P}^k) - \mathbf{x}^k.\end{aligned}$$

Then the approximate solution can be expressed explicitly:

$$\mathbf{x}^{k_{\max}} = \mathbf{x}^0 + \sum_{k=0}^{k_{\max}-1} G(\mathbf{X}^k, \mathbf{P}^k).$$

Since $(\mathbf{x}^k)_k$ converges, $\left(G(\mathbf{X}^k, \mathbf{P}^k)\right)_k$ must converge to 0. Thus, the approximate solution $\mathbf{x}^{k_{\max}}$ is obtained by summing up addends that are decreasing in magnitude (in the limit). We cannot change the order of the summation because we need the results $\mathbf{x}^k$ in the given order to compute the next addend $G(\mathbf{X}^k, \mathbf{P}^k)$. So our problem here is that we start with large addends and add up smaller and smaller contributions (in terms of absolute values) to the sum.

Summations of the above form are fairly inaccurate on the standard floating point representation, because once the exponent of the sum has reached a high value, the small addends contribute with only very few digits to the mantissa of the sum, and at a certain stage do not contribute at all. Therefore, we should accumulate the results in a high precision format, possibly even higher than the final accuracy we aim for. But since we target parallel hardware, we want to compute G itself and multiple iterations of the scheme in low precision. For this purpose we split the sum along some indexes $k^0_{\max} = 0, k^1_{\max}, \ldots, k^L_{\max} = k_{\max}$ into several parts

$$\mathbf{x}^{k_{\max}} = \mathbf{x}^0 + \sum_{l=0}^{L-1} \sum_{k=k^l_{\max}}^{k^{l+1}_{\max}-1} G(\mathbf{X}^k, \mathbf{P}^k). \qquad (2)$$

The computation in the inner sum uses low precision, whereas the accumulation in the outer sum happens in high precision. To exploit the full range of the low precision format, the inner iterations execute on differences rather than absolute values, i.e. instead of Eq. 1, for $k = k^l_{\max}, \ldots, k^{l+1}_{\max} - 1$ we use

$$\left(\mathbf{x}^{k+1} - \mathbf{x}^{k^l_{\max}}\right) := \left(\mathbf{x}^k - \mathbf{x}^{k^l_{\max}}\right) + G(\mathbf{X}^k, \mathbf{P}^k). \qquad (3)$$

Now, we must also compute $G(\mathbf{X}^k, \mathbf{P}^k)$ accurately in low precision. There are two scenarios.

G depends on $\mathbf{X}^k$. In this case we express $G(\mathbf{X}^k, \mathbf{P}^k)$ in terms of $G(\mathbf{X}^k - \tilde{\mathbf{X}}^k, \mathbf{P}^k)$, where the matrix $\tilde{\mathbf{X}}^k$ has the same size as $\mathbf{X}^k$ and contains in columns $k = k^l_{\max}, \ldots, k^{l+1}_{\max} - 1$ the vector $\mathbf{x}^{k^l_{\max}}$, i.e. the result from the beginning of the corresponding partial sum. Then the corresponding columns $\mathbf{x}^k$ and $\mathbf{x}^{k^l_{\max}}$ are of similar magnitude, their difference can be easily represented in the low precision format, and the computation of $G(\mathbf{X}^k - \tilde{\mathbf{X}}^k, \mathbf{P}^k)$ is quite accurate. However, the required reformulation can be tricky in general, especially for non-linear G. Let us give an example for the simple case when G depends only on $\mathbf{x}^k$ and is affine in this

argument. Then we can replace $G(\mathbf{X}^k, \mathbf{P}^k)$ in Eq. 3 with

$$G(\mathbf{x}^k, \mathbf{P}^k) = G(\mathbf{x}^k - \mathbf{x}^{k^l_{\max}}, \mathbf{P}^k) + \left(G(\mathbf{x}^{k^l_{\max}}, \mathbf{P}^k) - G(\mathbf{0}, \mathbf{P}^k)\right),$$

where $\left(G(\mathbf{x}^{k^l_{\max}}, \mathbf{P}^k) - G(\mathbf{0}, \mathbf{P}^k)\right)$ is computed once for the entire partial sum in high precision and down-casted to low precision.

The above case already covers many of the iterative solvers used for linear equation systems $\mathbf{Ax} = \mathbf{b}$. For example, for a stationary iterative method (e.g. Jacobi, Gauss-Seidel, SOR) we have

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{B}\mathbf{x}^k + \mathbf{c}, \qquad G(\mathbf{x}^k) := (\mathbf{B} - \mathbb{1})\mathbf{x}^k + \mathbf{c}, \\ G(\mathbf{x}^k) &= (\mathbf{B} - \mathbb{1})(\mathbf{x}^k - \mathbf{x}^{k^l_{\max}}) + \mathbf{c} + (\mathbf{B} - \mathbb{1})\mathbf{x}^{k^l_{\max}}, \end{aligned}$$

and this reformulation is equivalent to the solution of the residual system $\mathbf{Ax}' = \mathbf{b}'$ with $\mathbf{x}' = \mathbf{x} - \mathbf{x}^{k^l_{\max}}$ and $\mathbf{b}' = \mathbf{b} - \mathbf{Ax}^{k^l_{\max}}$.

G does not depend on $\mathbf{X}^k$. Some schemes are already constructed in such a way that they only accumulate the results as in Eq. 1 and do not use $\mathbf{X}^k$ in other computations, e.g. the Conjugate Gradient solver. In this case the auxiliary variables $\mathbf{P}^k$ implicitly contain the information about the global descent direction towards the limit. Because they accumulate errors over time while being computed in low precision, we use the transition from one partial sum to the other in Eq. 2 to recompute them in high precision and thus ensure that the following evaluations of G are more accurate again. In case of the Conjugate Gradient solver, the residual ($\mathbf{r}$) can be computed directly form the current high precision solution and the descent direction ($\mathbf{p}$) can be orthogonalized against it [2].

Above we have discussed how to handle the accumulation of the low precision results from G. We may also ask how the less precise computation of G itself effects the overall convergence rate. In general, the additional rounding errors incurred by the use of a lower precision format are still very small in comparison to the magnitude of G, because floating point numbers are designed to minimize the relative error. However, even these small effects will have a large impact if the exact scheme converges very quickly. In the other extreme of a very slowly converging scheme we may also run into problems, because the additional error due to the low precision can then make the difference between slow convergence and divergence. Often the flexibility of the sum splitting in Eq. 2 can alleviate convergence problems by simply producing a finer division and thus more computation in high precision, i.e. we may take advantage of the mixed precision method for all but very extreme cases. However, we cannot reduce the precision for the computation of G arbitrarily, because below a certain threshold the resulting vector will not point in the direction of the limit anymore.

3. PARALLEL CO-PROCESSORS

Mixed precision methods offer two main advantages on the hardware level.

Computation. The number of transistors needed for a multiplier grows quadratically with the operand size. Thus, if we half the operand size we can implement four low precision instead of one high precision multiplier. This point is less relevant for architectures which spend most of their transistors on caches and control logic, because then the enlarging of the ALUs has only a small impact on the overall transistor count. But for co-processors with a high number of parallel ALUs, doubling the operand size would quadruple the number of required transistors.

Table 1: Speedup of the CPU-GPU mixed precision solver over a CPU double precision solver.

domain size (l)	Conjugate Gradient	Multigrid
$66,049(\ 8)$	2.4	2.2
$263,169(\ 9)$	3.7	2.9
$1,050,625(10)$	4.2	3.7

Bandwidth. Many computations are limited by the bandwidth rather than processing requirements. By reducing the required bandwidth for most operations of a mixed precision algorithm we can also gain speedup for these memory bound algorithms. This point applies to all architectures as data movement has become much more expensive than computation, even within the same processor.

Several parallel architectures have evolved in recent years that are particularly suitable for mixed precision iterative schemes. Some of them do not offer double precision computations, like GPUs and allegedly the AGEIA PhysX processor. With these schemes they can deliver double precision results while executing in parallel on the single float format and having the CPU do very few double precision corrections. Others, like FPGAs and the Cell processor, can compute in double arithmetic but lose a factor of four or more in performance. This loss can be recovered by executing most of the computations in single and only very few corrections in double precision.

4. RESULTS

We have examined the applicability of mixed precision methods for Finite Element simulations on GPUs and FPGAs with different variants of plain Conjugate Gradient (CG) and sophisticated Multigrid (MG--MG) solvers [1, 2]. Although actual implementations have to exploit different functionality of the underlying processing elements and interconnects, the general mixed precision method as outlined above applies to both architectures. In summary, we can always obtain the same accuracy as a full double precision solver (compared against an analytically given reference solution), compute less than 1% of the operations in double precision and 99% in parallel in single precision, and gain overall a factor of 3-5 speedup for sufficiently large domains, compared to a highly optimized direct CPU double precision solver. For small problem sizes that fit into the L2 cache, the CPU can outperform even highly parallel architectures because of the enormous bandwidth to the cache.

Table 1 shows exemplary acceleration factors for combined CPU-GPU mixed precision solvers. The CPU performs the few double precision corrections, while the GPU executes the inner CG or MG solver as a low precision preconditioner. In this test we solve the Poisson problem on a regular discretization of the unit square with $(2^l + 1)^2$, $l = 8, 9, 10$ grid points (4x more for the MG--MG), using conforming bilinear Finite Elements. Timings include the data transfers to and from the GPU over the PCIe bus.

5. REFERENCES

[1] D. Göddeke, R. Strzodka, and S. Turek. Accelerating double precision FEM simulations with GPUs. In *Proceedings of ASIM 2005 - 18th Symposium on Simulation Technique*, 2005.

[2] R. Strzodka and D. Göddeke. Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In *IEEE Proceedings on Field-Programmable Custom Computing Machines*, 2006.