Mixed-precision orthogonalization scheme and its case studies with CA-GMRES on a GPU

Ichitaro Yamazaki, Stanimire Tomov, Tingxing Dong, and Jack Dongarra

University of Tennessee, Knoxville, U.S.A.

Abstract. We propose a mixed-precision orthogonalization scheme that takes the input matrix in a standard 32 or 64-bit floating-point precision, but uses higher-precision arithmetics to accumulate its intermediate results. For the 64-bit precision, our scheme uses software emulation for the higher-precision arithmetics, and requires about 20× more computation but about the same amount of communication as the standard orthogonalization scheme. Since the computation is becoming less expensive compared to the communication on new and emerging architectures, the relative cost of our mixed-precision scheme is decreasing. Our case studies with CA-GMRES on a GPU demonstrate that using mixed-precision for this small but critical segment of CA-GMRES can improve not only its overall numerical stability but also, in some cases, its performance.

1 Introduction

On modern computers, communication (e.g., data movement through memory hierarchies) is becoming expensive compared to computation (i.e., floating point operations), in terms of both required cycle time and energy consumption. It is critical to take this hardware trend into consideration when designing high-performance software for new and emerging computers. For this purpose, we studied a communication-avoiding variant of the Generalized Minimum Residual method [5] (CA-GMRES) on multicore CPUs with multiple GPUs [7]. In this study, we found that to achieve the high performance of CA-GMRES, an efficient and numerically stable orthogonalization scheme is crutial. In this same study, the Cholesky QR (CholQR) orthogonalization scheme [6] showed a superior performance and obtained the performance of optimized BLAS-3 GPU kernels. Unfortunately, CholQR can be numerically unstable, and CA-GMRES may not converge even with reorthogonalization.

To address the aforementioned deficiency, in this paper, we design and study a mixed-precision CholQR that takes the input matrix in a standard precision but accumulates its intermediate results in a higher-precision. Though it has a greater computational cost, our mixed-precision scheme requires about the same

[0] We thank Maho Nakata, Daichi Mukunoki, and the members of DOE "Extreme-scale Algorithms & Solver Resilience (EASIR)" for helpful discussions.

$\widehat{\mathbf{x}} := \mathbf{0}$ and $\mathbf{v}_1 := \mathbf{b}/\|\mathbf{b}\|_2$.
repeat `(restart-loop)`
  `Projection Subspace Generation on GPUs (inner-loop):`
  for $j = 1, s+1, 2s+1, \ldots, m$ **do**
    MPK: Generate new vectors $\mathbf{v}_{k+1} := A\mathbf{v}_k$
      for $k = j, j+1, \ldots, \min(j+s, m)$.
    BOrth: Orthogonalize $V_{j+1:j+s+1}$ against $V_{1:j}$.
    TSQR: Orthogonalize the vectors within $V_{j+1:j+s+1}$.
  end for

  `Projected Subsystem Solution on CPUs (restart):`
  Compute the solution $\widehat{\mathbf{x}}$ in the generated subspace,
    which minimizes its residual norm.
  Set $\mathbf{v}_1 := \mathbf{r}/\|\mathbf{r}\|_2$, where $\mathbf{r} := \mathbf{b} - A\widehat{\mathbf{x}}$.
until solution convergence **do**

Fig. 1. CA-GMRES(s,m).

`Step 1: Gram-matrix formation`
for $d = 1, 2, \ldots, n_g$ **do**
  $B^{(d)} := V_{1:s+1}^{(d)T} V_{1:s+1}^{(d)}$ on GPU
end for
$B := \sum B^{(d)}$ (comm)

`Step 2: Cholesky factorization`
$R :=$ `chol`(B) on CPU

`Step 3: Orthogonalization`
for $d = 1, 2, \ldots, n_g$ **do**
  copy R to d-th GPU
  $V_{1:s+1}^{(d)} := V_{1:s+1}^{(d)} R^{-1}$ on GPU
end for

Fig. 2. CholQR.

amount of communication as the standard scheme does. Since the computation is becoming less expensive compared to the communication, we hope to improve the overall numerical stability of CA-GMRES using the mixed-precision without a significant increase in the orthogonalization time. Case studies on different GPUs demonstrate that this scheme can improve not only the stability of CA-GMRES but also, in some cases, the performance by allowing a larger block size, avoiding the reorthogonalization, and improving the convergence rate.

2 Communication-Avoiding GMRES

At the j-th GMRES iteration, the $(j+1)$-th basis vector $\mathbf{v}_{j+1}$ is generated through a sparse matrix-vector multiply (*SpMV*) followed by its orthonormalization (*Orth*) against the previously-generated basis vectors. In our implementation [7], the coefficient matrix A and the basis vectors are distributed in a block row format among the GPUs on a compute node. We generate these basis vectors on the GPUs, while the projected subsystem is solved on the CPUs.

Even on a single GPU, which is the focus of this paper, both *SpMV* and *Orth* require communication to move the data through the memory hierarchy of the GPU. CA-GMRES aims to reduce this communication by replacing *SpMV* and *Orth* with three new kernels – matrix-powers kernel (*MPK*), block orthogonalization (*BOrth*), and tall-skinny QR (*TSQR*) – that generate and orthogonalize a set of s basis vectors at once. By avoiding the communication, even on a single GPU, CA-GMRES can obtain a speedup of up to two [7]. Figure 1 shows the pseudocode of CA-GMRES(s, m), where $\mathbf{v}_j$ is the j-th column of V, $V_{j:k}$ is the submatrix consisting of the j-th through the k-th columns of V, and the iteration is restarted after $m+1$ basis vectors are computed.

3 Cholesky QR factorization

In this section, we use $V_{1:s+1}^{(d)}$ to denote the local matrix of $V_{1:s+1}$ on the d-th GPU, and n_g is the number of available GPUs. To orthogonalize the $s+1$ vec-

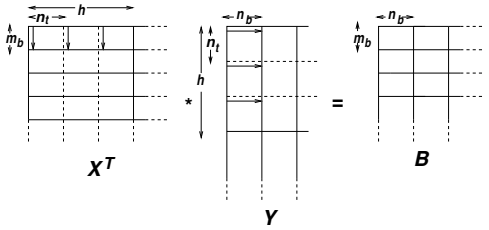

Fig. 3. *InnerProds* implementation (arrow shows data access by a GPU thread).

```
double regC[m_b][n_b], regA[m_b], regB
for ℓ = 1, ... i_b do
    for j = 1 ... n_b
      regA[i] = x_{ℓ*n_t,j}
    for j = 1, ..., n_b do
        regB = y_{ℓ*n_t,j}
        for i = 1 ... m_b
          regC[i][j] += regA[i] * regB
    end for
end for
```

Fig. 4. *InnerProds* pseudocode ($i_b = \frac{h}{n_t}$).

tors $V_{1:s+1}$ (as for *TSQR*), CholQR first forms the Gram matrix $B := V_{1:s+1}^T V_{1:s+1}$ through the matrix-matrix product $B^{(d)} := V_{1:s+1}^{(d)T} V_{1:s+1}^{(d)}$ on the GPU, followed by the reduction $B := \sum_{d=1}^{n_g} B^{(d)}$ on the CPU. Next, the Cholesky factor R of B is computed on the CPU. Finally, the GPU orthogonalizes $V_{1:s+1}$ by a triangular solve $V_{1:s+1}^{(d)} := V_{1:s+1}^{(d)} R^{-1}$. Hence, all the required GPU-GPU communication is aggregated into a pair of messages, while the GPU computation is based on BLAS-3. Hence, both intra and inter GPU communication can be optimized. Figure 2 shows these three steps of CholQR. Unfortunately, the condition number of B, $\kappa(B)$, is the square of $\kappa(V_{1:s+1})$. This often causes numerical problems, especially in CA-GMRES, where even using the Newton basis, the vector $\mathbf{v}_j$ converges to the principal eigenvector of A, and $\kappa(V_{1:s+1})$ can be large.

4 Mixed-Precision Orthogonalization with a GPU

4.1 Implementation

Since the Gram matrix is much smaller in its dimension than the coefficient matrix ($s \ll n$), CholQR typically spent only a small portion of its orthogonalization time computing its Cholesky factor at Step 2. In addition, solving the triangular system with many right-hand-sides at Step 3 exhibits a high parallelism and can be implemented efficiently on a GPU. On the other hand, at Step 1, computing each element of the Gram matrix requires a reduction operation on n-length vectors. These inner-products (*InnerProds*) are communication-intensive and exhibit only limited parallelism. Hence, Step 1 often becomes the bottleneck, where standard implementations fail to obtain high-performance on the GPU.

In our implementation of a matrix-matrix (GEMM) multiply, $B := X^T Y$, each thread block computes a partial *InnerProds*, $B^{(i,j,k)} := X^{(k,i)T} Y^{(k,j)}$, where $X^{(k,i)}$ and $Y^{(k,j)}$ are h-by-m_b and h-by-n_b blocks of X and Y, respectively (see Figure 3 for an illustration). Within a thread block, each of n_t threads computes its partial result in local registries (see Figure 4 for the pseudocode, where $\underline{x}_{\ell,j}$ is the (ℓ, j)-th element of the local $X^{(k,i)}$). Then, each thread block performs the binary reduction of the partial results among its threads, summing n_r columns at a time using the shared memory to store $n_t \times (m_b \times n_r)$ numerical values. The

dd-operation	# of d-instructions Add/Sub	Mul	FMA	Total
Mul (dd-input)	5	3	1	9
Mul (d-input)	3	1	1	5
Add (IEEE-style)	20	0	0	20
Add (Cray-style)	11	0	0	11

Fig. 5. # of d-instructions in dd-operations.

	$\|I - Q^TQ\|$	# flops	GPU comm.
MGS [3]	$O(\epsilon\kappa)$	$2sn^2$, xDOT	$O(s^2)$
CGS [3]	$O(\epsilon\kappa^s)$	$2sn^2$, xGEMV	$O(s)$
CholQR [6]	$O(\epsilon\kappa^2)$	$3sn^2$, xGEMM	$O(1)$
SVQR [6]	$O(\epsilon\kappa^2)$	$3sn^2$, xGEMM	$O(1)$
CAQR [2]	$O(\epsilon)$	$4sn^2$, xGEQF2	$O(1)$

Fig. 6. Properties of standard orthogonalization schemes. More details in [7].

final result is computed by another binary reduction among the thread blocks. Our implementation is designed to reduce the number of synchronizations among the threads while relying on the CUDA runtime and the parameter tuning to exploit the data locality.[1] For the symmetric (SYRK) multiply, $B := V^TV$, the thread blocks compute only a triangular part of B and reads $V^{(k,j)}$ once for computing a diagonal block. We show their performance in the next subsection.

Let us analyze the orthogonality error of CholQR in finite precision. Following the standard error analysis, we denote our finite precision operations at each step of CholQR as follows, where $\widehat{B}$ is the result of computing B in a finite precision, and ϵ_i is the arithmetic precision used at Step i:

$$\begin{array}{lll}
\text{Step 1 } (B := V^TV) & : \widehat{B} = B + \delta B, & \text{where } \|\delta B\| \le c_1\epsilon_1\|V\|^2, \\
\text{Step 2 } (\widehat{B} = R^TR) & : \widehat{R}^T\widehat{R} = \widehat{B} + \delta R^T\delta R, & \text{where } \|\delta R^T\delta R\| \le c_2\epsilon_2\|V\|^2, \\
\text{Step 3 } (Q := V\widehat{R}^{-1}) & : \widehat{Q} = Q + \delta Q, & \text{where } \|\delta Q\| \le c_3\epsilon_3\|\widehat{R}\|.
\end{array}$$

Furthermore, from Steps 1 and 2, we have $\|\widehat{R}\|^2 \le \|V\|^2 + (c_1\epsilon_1 + c_2\epsilon_2)\|B\|$. Hence, the orthogonality error of CholQR is given by

$$\begin{aligned}
I - \widehat{Q}^T\widehat{Q} &= I - (Q+\delta Q)^T(Q+\delta Q) \\
&= I - Q^TQ - Q^T\delta Q - \delta Q^TQ - \delta Q^T\delta Q \\
&= I - \widehat{R}^{-T}V^TV\widehat{R}^{-1} - Q^T\delta Q - \delta Q^TQ - \delta Q^T\delta Q \\
&= \widehat{R}^{-T}(\widehat{R}^T\widehat{R} - \widehat{B} + \delta B)\widehat{R}^{-1} - Q^T\delta Q - \delta Q^TQ - \delta Q^T\delta Q \\
&= \widehat{R}^{-T}(\delta R^T\delta R + \delta B)\widehat{R}^{-1} - Q^T\delta Q - \delta Q^TQ - \delta Q^T\delta Q,
\end{aligned}$$

and the error norm is bounded by

$$\begin{aligned}
\|I - \widehat{Q}^T\widehat{Q}\| &\le (c_1\epsilon_1 + c_2\epsilon_2)\|V\|^2\|\widehat{R}^{-1}\|^2 + 2c_3\epsilon_3\|Q\|\|\widehat{R}\| + c_3^2\epsilon_3^2\|\widehat{R}\|^2 \\
&\le (c_1\epsilon_1 + c_2\epsilon_2)\|V\|^2\|\widehat{R}^{-1}\|^2 + O(\epsilon_3\|\widehat{R}\|) \qquad (\text{assuming } \|\widehat{R}\| \le \epsilon_3^{-1}) \\
&\le (c_1\epsilon_1 + c_2\epsilon_2)\|V\|^2\|V^{-1}\|^2 + O(\epsilon_3\|V\|^2).
\end{aligned}$$

Based on the above analysis, our mixed-precision scheme uses a higher-precision at the first two steps of CholQR, while the standard precision is used at the last step, Hence, if the condition number of V is bounded by the reciprocal of the machine precision (i.e., $\|V\|\|V^{-1}\| = \kappa(V) \le \epsilon_d^{-1}$), we have

$$\|I - \widehat{Q}^T\widehat{Q}\| = O\left(\frac{\epsilon_{dd}}{\epsilon_d^2}(\epsilon_d\kappa(V))^2\right) \le O\left(\epsilon_d\kappa(V)\right),$$

[1] In the current implementation, the numbers of rows and columns in X and Y are a multiple of h, and multiples of m_b and n_b, respectively, where n_b is a multiple of n_r.

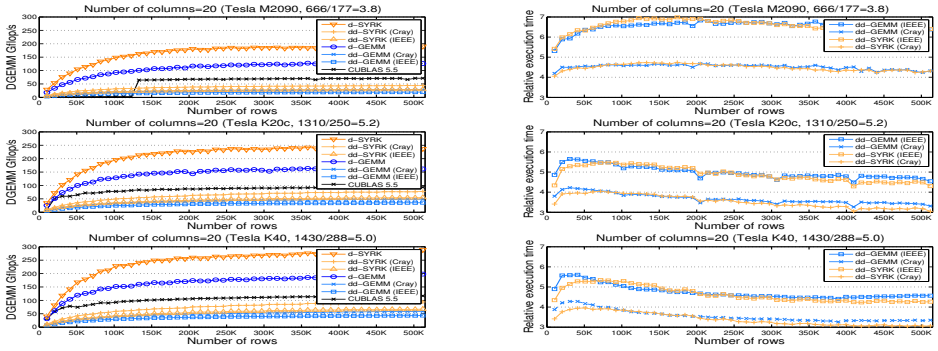

Fig. 7. Performance of *InnerProds* in double precision.

where ϵ_d and ϵ_{dd} are the machine epsilons in the standard and higher precisions, respectively. The modified Gram-Schmidt (MGS) [1] is another popular orthogonalization scheme which has the same norm-wise orthogonality error bound as dd-CholQR and was stable in our numerical studies [7]. However, MGS is based on BLAS-1 and obtains only a fraction of the d-CholQR performance [7].

When the target hardware does not support a desired higher precision, software emulation is needed. For instance, double-double (dd) arithmetic emulates the quadruple precision arithmetic by representing each floating-point value by an unevaluated sum of two double precision numbers, and is capable of representing the 106 bits precision, while the standard double value is 53 bits precision (i.e., $\frac{\epsilon_{dd}}{\epsilon_d^2} = 1$). There are two standard implementations [4] of dd-add, $a + b = \widehat{c} + e$, where one satisfies the IEEE-style error bound ($e = \delta(a + b)$ with $\delta \leq 2\epsilon_{dd}$), and the other satisfies the weaker Cray-style error bound ($e = \delta_1 a + \delta_2 b$ with $|\delta_1|, |\delta_2| \leq \epsilon_{dd}$). Table 5 summarizes the computational costs of the dd-arithmetics required for the mixed-precision dd-CholQR. Using this software emulation, dd-CholQR performs much more computation than the standard d-CholQR. On the other hand, d-CholQR and dd-CholQR communicate about the same amount since dd-CholQR only writes the s-by-s output matrix in the dd-precision, while reading the n-by-s input matrix in the d-precision ($s \ll n$).

4.2 Performance

Figure 7 compares the *InnerProds* performance in d- or dd-precision on different GPUs. Each GPU has a different relative cost of communication to computation, and on top of each plot, we show the ratio of the double-precision peak performance (Gflop/s) over the shared memory bandwidth (GB/s) (i.e., flop/B to obtain the peak). This ratio tends to increase on a newer architecture, indicating a greater relative communication cost. We tuned our kernel for each matrix dimension in each precision on each GPU (see the five tunable parameters h, m_b, n_b, n_r, and n_t in Section 4.1), and the figure shows the optimal performance.

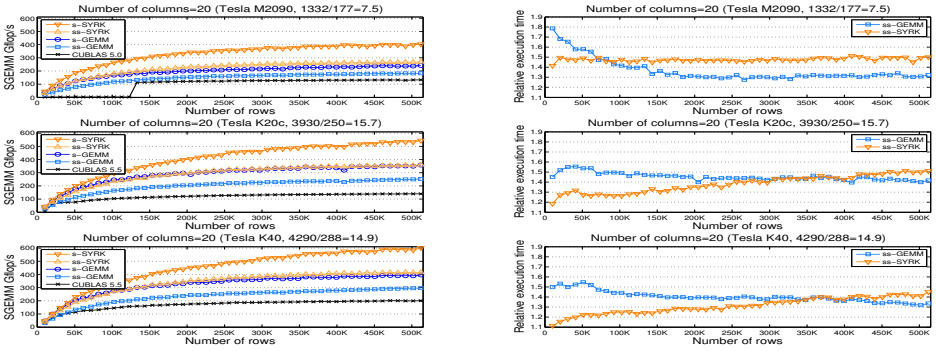

Fig. 8. Performance of *InnerProds* in single precision.

Based on the shared memory bandwidth in the figure, the respective peak performances of d-GEMM are 442, 625, and 720Gflop/s on M2090, K20c, and K40. Our d-GEMM obtained 29, 26, 28% of these peak performances and speedups of about 1.8, 1.7, and 1.7 over CUBLAS on these GPUs. In addition, though it performs twenty times more operations, the gap between *dd-InnerProds* and *d-InnerProds* tends to decrease on a newer architecture with a lower computational cost, and *dd-InnerProds* is only about three times slower on K20.

Figure 7 shows the performance of the mixed-precision *ss-InnerProds* in single precision, where the input matrix is in single precision, but the intermediate results are computed in double precision. Since the higher-precision is now supported by the hardware, even on an older GPU, the relative cost of the mixed-precision *ss-InnerProds* over the standard *s-InnerProds* is much lower than that of *dd-InnerProds* over *d-InnerProds*, where the software emulation is needed. In addition, *ss-InnerProds* is significantly more efficient obtaining over 300Gflop/s, in comparison to *d-InnerProds* that obtains just over 150Gflop/s.

Figure 9 shows the breakdown of d-CholQR orthogonalization time. Because of our efficient implementation of *InnerProds*, only about 30% of the orthogonalization time is now spent in *d-InnerProds*. As a result, while *dd-InnerProds* was about three times more expensive than *d-InnerProds*, Figure 10 shows that dd-CholQR is only about 1.7 or 1.4 times more expensive than d-CholQR when GEMM or SYRK is used for *InnerProds*, respectively. For dd-CholQR, the double-double Cholesky factorization on the CPU is computed using MPACK[2].

5 Case Studies with CA-GMRES

Figure 11 shows the orthogonality error and its upper-bound at each orthogonalization step of CA-GMRES$(15, 60)$ for the `cant` matrix.[3] Next, in Figure 12,

[2] `http://mplapack.sourceforge.net`

[3] Our sparse test matrices are from `http://www.cise.ufl.edu/research/sparse/matrices/`

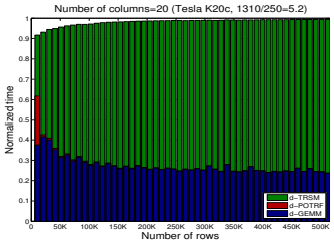

Fig. 9. d-CholQR time breakdown.

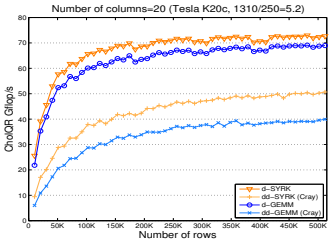

Fig. 10. d/dd-CholQR performance.

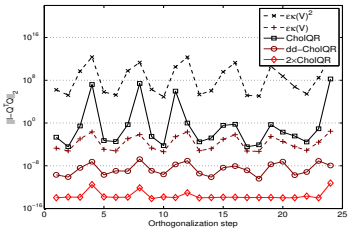

Fig. 11. Orthogonality error bound.

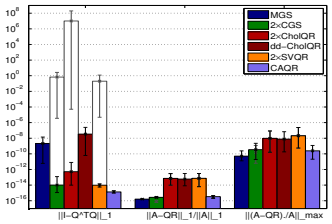

Fig. 12. Average error of *TSQR*.

we show the average error norms over the CA-GMRES iterations using different orthogonalization schemes (see Figure 6) for the same matrix. For this particular matrix, CGS, CholQR, and SVQR require reorthogonalization, and the white bars show the error norms after the first orthogonalization. Though the orthogonalization error of dd-CholQR is slightly greater than that of MGS, CA-GMRES converges with the same number of iterations without the reorthogonalization.

Finally, Figure 13 shows the normalized solution time of CA-GMRES with K20c. Using dd-CholQR, in these particular cases, the solution time was reduced not only because the reorthogonalization was avoided but also because CA-GMRES converged in fewer iterations in the first two cases, where the software emulation is needed.[4] In addition, dd-CholQR sometimes allowed a larger value of s, potentially leading to more efficient performance of the GPU kernels.

6 Conclusion

We proposed a mixed-precision orthogonalization scheme to improve the numerical stability of CA-GMRES. Our case studies demonstrated that though it requires 20× more computation, the use of mixed-precision for this small but critical segment of CA-GMRES can improve not only its overall stability but also, in some cases, its performance. We also showed that the cost of the mixed-precision scheme is decreasing on architectures where the relative cost of the

[4] In contrast to Figure 12 ($s = 15$), the orthogonality error norms of d-CholQR were significantly greater than those of dd-CholQR when $s = 30$.

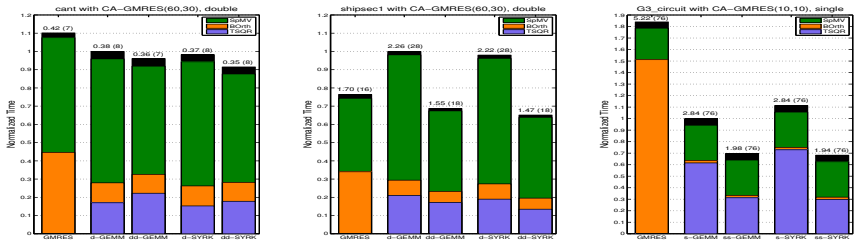

Fig. 13. CA-GMRES Performance: On top of each bar shows total time in sec. and restart count. CA-GMRES with non-optimal s got speedups over GMRES using CGS.

computation is decreasing. In this paper, we studied the performance of CA-GMRES on a single GPU. Our previous studies [7] demonstrated that on multiple GPUs of a compute node, the performance of CA-GMRES depends more on the performance of the GPU kernels than on the CPU-GPU communication. Hence, similar benefits of using mixed-precision are expected on the multiple GPUs. We plan to study the effects of mixed-precision on systems where the communication becomes more expensive (e.g., distributed GPUs, or CPUs), and where the scheme, therefore, may lead to a greater performance improvement. Furthermore, we will study the use of mixed-preision in eigensolvers where the orthogonality can be more crutial. Finally, it is of interest to apply or extend some recent mixed precision efforts (e.g., reproducible BLAS[5] and precision tuning[6]) for our studies. All the higher-precision BLAS developed for this study will be released through the MAGMA library.

References

1. A. Björck. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT Numerical Mathematics*, 7:1–21, 1967.
2. J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Sci. Comput.*, 34(1):A206–A239, 2012.
3. G. Golub and C. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.
4. Y. Hida, X. Li, and D. Bailey. Quad-double arithmetic: Algorithms, implementation, and application. Technical Report LBNL-46996, 2000.
5. Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.
6. A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23:2165–2182, 2002.
7. I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra. Improving the performance of CA-GMRES on multicores with multiple GPUs. To appear in the proceedings of 2014 IEEE IPDPS. Also, available as a UT-EECS technical report.

[5] `http://www.eecs.berkeley.edu/~hdnguyen/rblas`

[6] `http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/corvette/precimonious`