Performance of $s$-step GMRES
to avoid communication on/between GPUs

Ichitaro Yamazaki
with Hartwig Anzt$^1$, Stan Tomov$^1$, Mark Hoemmen$^2$, Jack Dongarra$^1$

$^1$University of Tennesse, Knoxville
$^2$Sandia National Laboratories

International Parallel and Distributed Symposium (IPDPS)
Phoenix, Arizona, 05/02/2014
What is “communication”? 

Inter-processor comm 
between parallel processing units (CPUs and GPUs):

for example, we “explicitly” perform inter-comm 
- using CUDA for CPU ↔ GPU 
- using MPI for CPU ↔ CPU

Intra-processor comm 
through local memory hierarchy (on GPU):

for example, we have different intra-comm 
- using BLAS-3, BLAS-2 or BLAS-1
How do we measure “communication”?

communication cost = \# messages \cdot \text{latency} + \text{communication volume/bandwidth}.

- explicitly reduce inter-comm, while implicitly reduce intra-comm.
- reduce both message count and volume vs. reduce message count but increase volume and comp (i.e., reduce latency).

Why do we care about “communication”?

- gap between arithmetic and comm costs is increasing
  some comm are more expensive on some hardware

\[
\frac{\text{time}}{\text{flop}} \ll \frac{1}{\text{bandwidth}} \ll \text{latency}.
\]

- current compute-bound algorithm may become comm-bound on a next machine.
- avoiding “communication” may also
  - avoid “synchronization,” providing more “parallelism” or reducing “ideling” time
  - improve “energy efficiency” (less frequent comm)
- etc. etc.
What is our goal?

study effectiveness of CA techniques for Krylov iterative methods to solve

\[ Ax = b, \]

where \( A \) is big, on GPUs.

- **direct methods** are robust and stable, but may be too expensive.
- **iterative methods** may be the only feasible alternative.
  - Krylov methods are a class of popular and flexible methods.
  - We focus on GMRES, popular for solving nonsymmetric system.
- **hybrid (direct/iterative) methods** may combine the strength of both.
  - preconditioner.
- **GPUs** are becoming popular for HPC.
  - on a node for this talk.

compare performance of GMRES and CA-GMRES on GPUs
Outline: CA-GMRES on multicores with multiple GPUs

- Algorithm and Implementation
- GPU Kernels
  - Matrix Powers Kernel
  - Orthogonalization Kernels
- Performance
- Final Remarks
Generalized Minimum RESidual (GMRES): from ‘linear algebra’ view

Krylov projection method for solving a nonsymmetric problem,

\[ Ax = b. \]

1. generate a Krylov subspace

\[ \mathcal{K}_j(A, q_1) = \text{span}(q_1, Aq_1, A^2q_1, \ldots, A^j q_1) \]

\[ = \text{span}(q_1, q_2, q_3, \ldots, q_{j+1}) \]

- \( j \)-th step generates ortho-normal basis \( q_{j+1} \).

2. find “best” \( \hat{x} \) in the subspace (with minimum residual norm \( \| b - A\hat{x} \|_2 \))

- solution converges with non-increasing residual norm.
- \( \mathcal{K}_n(A, q_1) \) spans the whole space, where \( n \) is the dimension of \( A \).
- iteration is restarted using the “best” approximation.
Restarted GMRES on GPUs: from ‘algorithmic’ view

1. Generate Krylov Basis on GPUs: \( \sim O(m \cdot \text{nnz}(A) + m^2 n) \) flops.
   
   for \( j = 1, 2, \ldots, m \) do
   
   Sparse Matrix-Vector (\textit{SpMV}) Product:
   
   \[ v_{j+1} := Av_j \]
   
   Orthonormalization (\textit{Orth}):
   
   \[ q_{j+1} := v_{j+1} - Q_{1:j} Q_{1:j}^T v_{j+1} \]

   end for

2. Solve Projected Subsystem on CPUs: \( \sim O(m^2) \) flops.

   GMRES: least-square problem
   
   \( \rightarrow \) restart with “better” starting \( v_1 \).

Performance considerations:

- generating basis vectors dominates computational cost.
  
  - distribute \( A \) and \( Q \) in a 1D block row among GPUs (e.g., \( k \)-way graph vertex cut).

- both \textit{SpMV} and \textit{Orth} require “expensive” communication:
  
  - point-to-point/neighborhood for \textit{SpMV} (inter-GPU).
  - global all-reduces in \textit{Orth} (inter-GPU).
  - data movements between local memory hierarchy (intra-GPU).
“Communication-Avoiding” GMRES

Improve performance by avoiding some comm:

▶ replacing \( SpMV \) with
  
  ▶ Matrix Powers Kernel (\( MPK \)):
    
    apply matrix powers \( Av_j, A^2v_j, \ldots, A^sv_j \)
    
    by \( v_{j+k} := Av_{j+k-1} \) for \( k = 1, 2, \ldots, s \).

▶ replacing \( Orth \) with
  
  ▶ Block Orthogonalization (\( BOrth \)):
    
    orthogonalize \( V_{j+1:j+s} \) against \( Q_{1:j} \).

▶ Tall-Skinny QR (\( TSQR \)):
  
  orthogonalize \( V_{j+1:j+s} \) against each other.

→ generate \( s \) vectors “at once” to reduce communication.
Matrix Powers Kernel for a tridiagonal matrix

1. communicate required nonlocal elements for $s$-step between GPUs

2. apply $s$ matrix powers with extra computation on shrinking overlap
   - local submatrix is expanded with $s$-level overlap

→ reduce inter-GPU latency by $s$ (with redundant computation).
Matrix Powers Kernel for a general matrix

1. communicate required nonlocal elements for $s$-step between GPUs

2. apply $s$ matrix powers with extra computation on shrinking overlap
   - local submatrix is expanded with $s$-level overlap

→ reduce inter-GPU latency by $s$ (with redundant computation).
Matrix Powers Kernel Performance

Our MPK reduces inter-GPU latency but trades off

- additional memory to store “ghost boundary.”
- addition computation for SpMV with “ghost boundary.”
- potentially, increasing total inter-GPU communication volume.

- MPK performance with G3_Circuit -
TSQR Algorithms

Many ways to orthogonalize columns of $V$ with each other:

- **Modified Gram-Schmidt** (with $O(s^2)$ reductions)
  - ortho each column against each column based on BLAS-1 $\times$DOT and $\times$AXPY

- **Classical Gram-Schmidt** (with $O(s)$ reductions)
  - ortho each column against prev columns based on BLAS-2 $\times$GEMV

- **Cholesky (or SVD) QR** (with $O(1)$ reductions)
  - ortho all columns against prev columns based on BLAS-3 $\times$GEMM, $\times$TRSM

- **CAQR** (with $O(1)$ reductions)
  - ortho all columns against prev columns based on tree-reduction BLAS-1,2 $\times$GEQR2

- trade-off exists between performance and accuracy
- BOOrth is based on MGS or CGS.
TSQR Implementations

Standard algorithms for TSQR: consider both inter, and intra-GPU communication using optimized BLAS kernels on each GPU.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( | I - Q^T Q | )</th>
<th># flops, GPU kernel</th>
<th># GPU-CPU comm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGS</td>
<td>( O(\epsilon \kappa) )</td>
<td>2( ns^2 ), BLAS-1 xDOT</td>
<td>( O(s^2) )</td>
</tr>
<tr>
<td>CGS</td>
<td>( O(\epsilon \kappa^2) )</td>
<td>2( ns^2 ), BLAS-2 xGEMV</td>
<td>( O(s) )</td>
</tr>
<tr>
<td>CholQR</td>
<td>( O(\epsilon \kappa^2) )</td>
<td>2( ns^2 ), BLAS-3 xGEMM</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>SVQR</td>
<td>( O(\epsilon \kappa^2) )</td>
<td>2( ns^2 ), BLAS-3 xGEMM</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>CAQR</td>
<td>( O(\epsilon) )</td>
<td>4( ns^2 ), BLAS-1,2 xGEQR2</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

multiple dot–products with DGEMV (s=30)

\[ V_{1:k-1} := \begin{bmatrix} r_{1:k-1, k} \\ V_{1:k-1}^T \end{bmatrix} \]
TSQR Implementations:

Standard algorithms for **TSQR**: consider both inter, and intra-GPU communication using optimized BLAS kernels on each GPU.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FLOP Complexity</th>
<th>GPU Kernel</th>
<th>GPU-CPU Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGS</td>
<td>$O(\epsilon \kappa)$</td>
<td>$2ns^2$, BLAS-1 xDOT</td>
<td>$O(s^2)$</td>
</tr>
<tr>
<td>CGS</td>
<td>$O(\epsilon \kappa^2)$</td>
<td>$2ns^2$, BLAS-2 xGEMV</td>
<td>$O(s)$</td>
</tr>
<tr>
<td>CholQR</td>
<td>$O(\epsilon \kappa^2)$</td>
<td>$2ns^2$, BLAS-3 xGEMM</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>SVQR</td>
<td>$O(\epsilon \kappa^2)$</td>
<td>$2ns^2$, BLAS-3 xGEMM</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>CAQR</td>
<td>$O(\epsilon)$</td>
<td>$4ns^2$, BLAS-1,2 xGEQR2</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

![Graph showing block inner-products with DGEMM (s=30)](image)

**CA-GMRES on GPUs**
TSQR Performance (16-core SandyBridge with three M2090 Fermi, $s = 30$)

- performance depends more on intra-comm (BLAS performance) than on inter-comm.
- it scales well over 3 GPUs.
Experiment Setups for Performance of GMRES/CA-GMRES

- matrix equilbration for numerical stability
- reordering/partitioning for performance
  - local reordering if needed
- CGS for Ortho/BOrth, CholQR for TSQR
  - with reorthogonalization if needed.
- Newton basis to enhance MPK stability,
  \[ v_{k+1} = \prod_{i=1}^{k} (A - \theta_i)q_1. \]
  - \( \theta_i \) is eigenvalues of \( H \) from first restart loop (GMRES)
- use ELLPACKT for SpMV
  (many sparse formats with different performance).
- parameter selection
  - pick “good” \( m \) for GMRES on GPU, and
  “stable” \( s \) for CA-GMERS.
- one node of Keeneland
  (2 \( \times \) 8 Intel SandyBridge + 3 NIVIA M2090).
CA-GMRES Performance

- obtained speedups of up to 2.0 (∼ iteration count).
  - **BOrth** and **TSQR** with speedups of up to 4.2 over **Orth**.
  - **MPK** got a speedup of 1.6, but could be slower than **SpMV**, especially with a relatively large s preferred by **Orth** (s = 15 ∼ 20).

- if comm is expensive, it may worth avoiding it.
  - speedups depends on hardware, implementation, matrix, etc.
    - newer GPU (e.g., Kepler) has a larger gap between comp and comm.
Final Remarks:

- we studied existing CA techniques on GPUs (please see our paper for references)
  - if comm is significant, it may worth to avoid it (requires careful implementations)

- we now have a framework to build on:
  - CA-GMRES with an adaptive step size:
    - adjust step size for MPK at runtime.
  - Mixed-precision CholQR (same comm but more comp)
    - improve overall stability by selective use of higher precision.
  - CA-GMRES on a hybrid CPU/GPU cluster
    - exhibit similar benefits on 120 GPUs.

Current Research:

- more GPU kernels: e.g.,
  - MPK: read $A$ once, and read and write $q_j$ once.
  - CAQR: batched QR.

- more effective usage of CPUs.
- hypergraph partitioning for MPK and CA-GMRES.
- CA-Preconditioner.
- inner-outer iteration.
- eigenvalue/SVD/low-rank approximation.
Thank you!!