Communication-avoiding optimization methods for fermion matrix inverters

Alexei Strelchenko

Scientific Computing Division @ Fermilab

July 23, 2018

Overview

- ECP activity
  - ECP solver group communicates with P. Eller, UIUC, and M. Hoemmen, Sandia Lab
- CA optimization approaches
  - Computation optimization (power-kernels , block orth. etc.)
  - Algorithm optimization (pipelining, enlarged search subspace)
- Numerical stability
- Trilinos library
  - CA optimized algorithms and kernels (TSQRT, iter. solvers and preconditioners)

CA solvers in focus

- S-step Krylov solvers
- Pipelined solvers:
  - Ghysels and Vanroose (PipePCG algorithm)
  - Gropp and Eller (Pipe2PCG algorithm)
  - Cornelis, Cools and Vanroose (Deep pipelined p(l)-PCG)
- EKS and Block methods:
  - Grigori et al (SRE-CG)

Two-term reccurence piplened PCG scheme

- P. Ghysels and W. Vanroose, Journal of Parallel Computing 40 (2014) 224-238

1 Input: problem matrix A, precond. matrix M, source vector x.
2 Output: solution vector x.
3 $r_0 = b - Ax_0$, $u_0 = Mr_0$, $w_0 = Au_0$
4 $\delta_0 = (w_0, u_0)$, $\gamma_0 = (r_0, u_0)$, $norm(u_0)$
5 Start main loop:
6 Compute CG coefficients α , β
7 Update iter. residual, solution and aux fields: 8 saxpy's
8 $\delta_j = (w_j, u_j)$, $\gamma_j = (r_j, u_j)$, $norm(u_j)$
9 $m_j = Mw_j$, $n_j = Am_j$

Three-term reccurence pipelined PCG scheme

- P. Eller, W. Gropp SC 2016 204-215

1 Input: problem matrix A, precond. matrix M, source vector x.
2 Output: solution vector x.
3 $r_0 = b - Ax_0$, $u_0 = Mr_0$, $w_0 = Au_0$
4 $\delta_0 = (w_0, u_0)$, $\gamma_0 = (r_0, u_0)$, $norm(u_0)$
5 $p_0 = Mw_0$, $q_0 = Ap_0$
6 $c_0 = Mq_0$, $d_0 = Ac_0$
7 Start main loop (combined two iterations):
8 Compute CG coefficients
9 Update iter. residual, solution and aux fields: 14 saxpy's
10 Compute 10 dot products
11 $c_j = Mq_j$, $d_j = Ac_j$
12 $g_j = Md_j$, $h_j = Ag_j$

Enlarged Krylov SubspaceMethods (L. Grigori et. al. 2014)

- Partition the lattice into t subdomains
- Split the residual r_0 into t vectors corresponding to the t domains:

$$r_0 \rightarrow T(r_0) = \begin{bmatrix} \star & 0 & 0 & \dots & 0 \\ \star & 0 & 0 & \dots & 0 \\ 0 & \star & 0 & \dots & 0 \\ 0 & \star & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \star \\ 0 & 0 & 0 & \dots & \star \end{bmatrix}$$

- The EKS solver generate t new basis vectors to obtain an enlarged Krylov subspace

$$K_{t,k+1}(A, r_0) = Span\{T(r_0), AT(r_0), ..., A^k T(r_0)\}$$

- The approximate solution: $x_{k+1} \in x_0 + K_{t,k+1}(A, r_0)$

Enlarged Krylov Methods (cont.)

- The EKS is a superset of the classical KS : $K_{k+1} \subset K_{t,k+1}$
- The enlarged subspaces are increasing subspaces, yet bounded

$$K_{t,1}(A, r_0) \subsetneq \cdots \subsetneq K_{t,kmax}(A, r_0) = K_{t,kmax+n}(A, r_0)$$

- We can consider EKM as a particular case of a block Krylov method:

$$AX = T(b), R_0 = T(b - Ax_0)$$

Enlarged CG algorithm

- Idea: use BlockCG framework for $R^0 = T(r_0)$
- The iter. residual $r_k = \sum_s R_s^{(k)}$
- The final solution $x = \sum_s X_s$
- The ECG converged in $k_{max}^{ECG} \leq k_{max}^{CG}$

Enlarged CG (Grigori et al, 2014)

1 $x_0 = 0$, $R^{(0)} = T(b - Ax_0)$.
2 $V^{(0)} =$ A-orthonormalize$(R^{(0)})$
3 Start main loop untill $||\sum_{s=0}^{t} R_s^{(k)}|| < \epsilon * ||b||$:
4 $V^{(k)} = AV^{(k-1)}$
A-orthonormalize against $V^{(k-1)}$ and $V^{(k-2)}$:
5 $V^{(k)} = V^{(k)} - V^{(k-1)}V^{(k-1)T}AV^{(k)} - V^{(k-2)}V^{(k-2)T}AV^{(k)}$
6 $V^{(k)} =$ A-orthonormalize$(V^{(k)})$
7 $T^{(k)} = V^{(k)T}R^{(k-1)}$
8 $X^{(k)} = X^{(k-1)} + V^{(k)}T^{(k)}$
9 $R^{(k)} = R^{(k-1)} - AV^{(k)}T^{(k)}$
10 $T^{(k)} = V^{(k)T}AR^{(k)}$

The solution $x = \sum_{s=0}^{t} X_s^{(k)}$:

Numerical stability of CA solvers

- Residual gap for classic CG algorithm:

$$f_{i+1} = b - A\bar{x}_{i+1} - \bar{r}_{i+1} = f_0 - \sum_{k=0}^{i}(A\epsilon_k^x + \epsilon_k^r)$$

- Local rounding errors are trivially accumulated in residual gap
- Unfortunately, for the CA optimized versions rounding errors propagation pattern is much more complicated: (see, e.g. Cools, arXiv:1804.02962)
$f_{i+1} = f_0 - \sum_{k=0}^{i} F(\epsilon_k^x, \epsilon_k^r, \epsilon_k^p, \epsilon_k^u, ...)$

Numerical stability of CA solvers

- In general, 3-term algo is less stable than 2-term one
  - M. Gutknecht, Z.Strakos, 2000
- (deep) pipelines further aggravate rounding error effects

Issues with variable preconditioning

- The precond. residual itself is computed via recursion:

$$u_{i+1} = M^{-1} r_{i+1} = M^{-1}(r_i - \alpha_i s_{i+1}) \approx u_i - \alpha_i q_{i+1}$$

- that is inexact even in exact arithmetics!
- $\rightarrow$ accumulation of errors in the preconditioned residual u_{i+1}

BCGrQ (A. Dubrulle, 2001)

Idea : improve BCG algorithm stability by performing a QR decomposition of the residual matrix R_k before constructing the descent directions ($V^{(k)} = P^{(k)} C^{(k-1)}$)

1 $X^{(0)} = 0$, $Q^{(0)} C^{(0)} = B$, $S^{(0)} = I$, $P^{(0)} = 0$.
2 Start main loop:
3 $P^{(k)} = Q^{(k-1)} + P^{(k-1)} S^{(k-1)T}$
4 $T^{(k)} = (P^{(k)T} A P^{(k)})^{-1}$
5 $X^{(k)} = X^{(k-1)} + P^{(k)} T^{(k)} C^{(k-1)}$
6 $Q^{(k)} S^{(k)} = Q^{(k-1)} - A P^{(k)} T^{(k)}$
7 $C^{(k)} = S^{(k)} C^{(k-1)}$

Example: mixed precision pipePCG

- Test setup:
  - Size : $32^3 \times 256$
  - Outer solver: pipePCG ($tol = 1e-8$)
  - Inner solver: minRes (10 iters)
  - Precision of outer solver: single
  - Precision of inner solver: half
- 16 FNAL GPU nodes (4 K40m per node)
  - Convergence in 65 iters
  - Residual replacements: 10
  - Asynchronous global reduction : 1.2 secs
  - Blocking (single) global reduction : 2.9 secs

Trilinos library

- Approaches
  - Access to Trilinos iterative solvers and preconditioners
  - Access to low-level (CA) kernels
- Kokkos support
- Testing platform

Access to algorithms

- Looking at Belos package:
  - Solver managers (BCGrQ, BGMRESDR etc.)
  - Interfaces to linear algebra, orthog. methods etc.
- Two main solvers stacks for sparse LA and parallel data redistribution facilities:
  - Epetra
  - Tpetra
- Memory management (Teuchos):
  - BLAS and LAPACK wrappers, smart pointers, parameter lists, and XML parsers

Access to CA optimized methods of Tpetra

- Uses the Kokkos sm parallel programming model
  - OpenMP
  - POSIX threads
  - CUDA
- Several impl. for TSQR (CUDA - currently WIP)
- Testing platform : QUDA within Singularity HPC containers

Operator interface

```
1     /// \class Operator
2     /// \brief Abstract interface for operators (e.g., matrices and
3     ///   preconditioners).
4     ///
5     /// \tparam Scalar The type of the entries of the input and output
6     ///   MultiVector objects.
7     /// \tparam LocalOrdinal The type of local indices.
8     /// \tparam GlobalOrdinal The type of global indices.
9     /// \tparam Node The Kokkos Node type.
10    ///
11    /// An Operator takes a MultiVector as input, and fills a given
12    /// output MultiVector with the result.
13    ///
14    /// Operator is just an interface, not an implementation. Many
15    /// different classes implement this interface, including sparse
16    /// matrices, direct solvers, iterative solvers, and
17    /// preconditioners.
18    template <class Scalar = ::Tpetra::Details::DefaultTypes::scalar_type,
19              class LocalOrdinal
    = ::Tpetra::Details::DefaultTypes::local_ordinal_type,
20              class GlobalOrdinal
    = ::Tpetra::Details::DefaultTypes::global_ordinal_type,
21              class Node = ::Tpetra::Details::DefaultTypes::node_type>
22    class Operator : virtual public Teuchos::Describable { ...
```

Map class

```
1     /// \class Map
2     /// \brief A parallel distribution of indices over processes.
3     ///
4     /// \tparam LocalOrdinal The type of local indices (int).
5     /// \tparam GlobalOrdinal The type of global indices (int or long).
6     ///  sizeof(GlobalOrdinal) >= sizeof(LocalOrdinal);
7     /// \tparam Node A class implementing on-node shared-memory parallel
8     ///   operations.
9     ///   The default \c Node type should suffice for most users.
10    ///   The actual default type depends on your Trilinos build options.
11    ///   This must be one of the following:
12    ///   Kokkos::Compat::KokkosCudaWrapperNode
13    ///   Kokkos::Compat::KokkosOpenMPWrapperNode
14    ///   Kokkos::Compat::KokkosThreadsWrapperNode
15    ///   Kokkos::Compat::KokkosSerialWrapperNode
16    /// Requires the Teuchos memory management classes.
17    /// Allows for "overlapping Maps"
18    template <class LocalOrdinal
    = ::Tpetra::Details::DefaultTypes::local_ordinal_type,
19              class GlobalOrdinal
    = ::Tpetra::Details::DefaultTypes::global_ordinal_type,
20              class Node = ::Tpetra::Details::DefaultTypes::node_type>
21    class Map : public Teuchos::Describable { ...
```

Singularity HPC containers (joined effort with Jim Simone)

- Developed for HPC type infrastructure (and "untrusted users")
  - can be executed like a native programm or script
  - simple intergation with job schedulers
  - bind host system driver libraries (e.g., NVIDIA GPU driver libs)
- Support several (convertable) container formats (squashfs, extfs, dir)
- Compatible with docker
- Valuable development tool for Trilinos framework

Building the stuff

```
1
2   cmake  -D CMAKE_CXX_COMPILER=mpicxx \
3          -D Trilinos_CXX11_FLAGS="--expt-extended-lambda" \
4          -D TPL_ENABLE_MPI=ON \
5          -D TPL_ENABLE_CUDA=ON \
6          -D Kokkos_ENABLE_Cuda=ON \
7          -D KOKKOS_ARCH=Pascal61 \
8          -D Kokkos_ENABLE_Cuda_UVM=ON \
9          -D Kokkos_ENABLE_Cuda_Lambda BOOL=ON \
10         -D Trilinos_ENABLE_Epetra=OFF \
11         -D Trilinos_ENABLE_Tpetra=ON \
12         -D Trilinos_ENABLE_Belos=ON  $TRILINOS_PATH
13
```

Running containers

- Start singularity shell session:
  singularity exec –nv -B /usr/bin:/opt/nvidia ohpc-quda-tril.simg bash
  - –nv option needed to bind GPU environment
- Run the application:
  ./trilinos_block_invert_test –recon 12 –prec double –dslash-type wilson –dim 16 16 16 16 –mass -0.8 –tol 1e-10 –niter 250 –nsrc 8 –msrc 8

Running containers (cont.)

```
================================================================================

                    TimeMonitor results over 1 processor

Timer Name                                  Global time (num calls)
--------------------------------------------------------------------------------
Belos: BlockCGSolMgr total solve time       35.71 (1)
Belos: DGKS[2]: Ortho (Inner Product)       2.618 (816)
Belos: DGKS[2]: Ortho (Norm)                0.2268 (2376)
Belos: DGKS[2]: Ortho (Update)              2.59 (816)
Belos: DGKS[2]: Orthogonalization           6.877 (108)
Belos: Operation Op*x                       21.63 (108)
Belos: Operation Prec*x                     0 (0)
================================================================================
---------- Actual Residuals (normalized) ----------

Problem 0 :     9.85637e-11
Problem 1 :     8.2153e-11
Problem 2 :     9.18115e-11
Problem 3 :     9.66285e-11
Problem 4 :     7.94028e-11
Problem 5 :     8.89036e-11
Problem 6 :     8.48587e-11
Problem 7 :     8.24447e-11

End Result: TEST PASSED
```

Figure : Trilinos execution

Conclusion

- Pipelined:
  - Optimized global comms but may effect convergence
  - Increase arithmetic intensity: merged LA operations
  - Already enabled in a number of solvers
- EKS:
  - Decrease num. of iterations and global comms
  - Increase arithmetic intensity: block LA, multi-rhs mat-vec
  - But this uses a block method for a single rhs: may not give an appropriate performance gain for LQCD tasks
- Test of Trilinos' solvers:
  - matrix-free highly optimized impl. (Grid, Quda, QPhiX)
  - does require kokkos mapping (raw arrays vs kokkos view objects)
- Test of Trilinos' TSQR implementation (Trilinos tpetra package):
  - idea is to consider Trilinos as a collection of specialized low-level LA routines
  - no (GPU-specific) kokkos realization for TSQR yet, WIP