Discrete Solvers at the Exascale

Esmond G. Ng
Lawrence Berkeley National Laboratory

SIAM AN14

What are discrete solvers?

- Nonlinear equations solvers
- Linear equations solvers
- Eigen solvers
- Time integrators

- Focus on linear equations solvers

What are the exascale challenges?

- High degree of parallelism
- Algorithmic scalability on heterogeneous systems
- Deep memory hierarchy – data movement or communication
- Limited memory size per code
- Resilience

- The DOE report on Applied Mathematics Research for Exascale Computing identified a number of applied math research areas that aim at tackling these challenges for discrete solvers
  - We will provide some examples to illustrate why and how those areas might be appropriate at exascale

Multiple-precision algorithms

- Facts …
  - Lower precision ops are often faster than higher precision ops
  - Lower precisions require less memory ==> require less data movement
- Use of multiple precisions is not new …
  - E.g., Kurzak & Dongarra (Concurrency and Computation: Practice & Experience, 2007)
    - Gaussian elimination in single precision and iterative refinements in double precision
  - But may become more important in exascale for data movement and limited memory reasons
- Open questions …
  - Determining when lower/higher precisions should be used in different parts of other types of matrix algorithms
  - Reliability, robustness, accuracy of multi-precision algorithms?

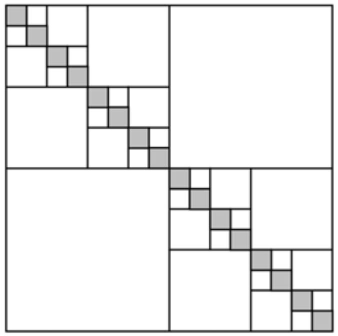
Data compression

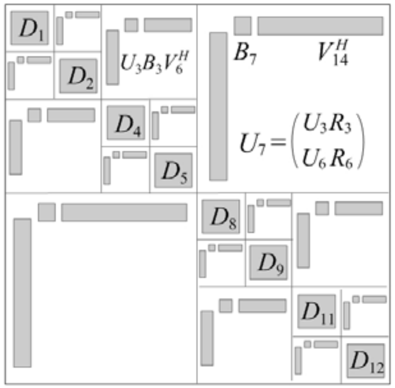
- Matrix computation is data intensive
  ==> require lot of data movement/communication

- Example …
  - One of the recent active research areas has focused on using some form of data compression to improve the performance of certain classes of matrix solvers
  - For matrices arising from the solution of PDEs with smooth kernels, off-diagonal blocks in the LU factorization often have low rank
    - Gu, Li, Xia, …
    - Weisbecker, …

Data compression in sparse matrix factorization

- General idea …
  - Apply SVD to a rank-deficient off-diagonal block
    ==> obtain a compact representation
  - Can apply the idea recursively and result in a hierarchical structure





- Advantages …
  - Lower storage requirements but essentially maintaining same accuracy
  - Result in less communication because the compact representation has to move less data
  - Also often require fewer operations overall (even though additional work is required to compute the compression)

Data compression in sparse matrix factorization

- Test problem (Li): 3D seismic imaging – Helmholtz equations up to 600^3 cubic grids (216M equations)
  - 16,000+ cores: 2x faster, uses 1/5 of memory vs a sparse direct solver based on Gaussian elimination

- Open questions …
  - Generalizations to other classes of matrices?
    - For other matrices, can use the approach to compute approximations, which can then be used as preconditioners
  - Data compression in other matrix algorithms?
  - Complexity analysis - Trade off between compression cost and possible reduction in memory?
  - Robustness, reliability, accuracy?

Randomization and Sampling

- Randomized algorithms have gained quite a bit of popularity in recent years.
  - Not entirely because of exascale computing
  - But some interesting ideas here

- Example …
  - Consider an m x n matrix A, where m and n are very large.
  - Suppose we want to get a low-rank approximation of A.
    - Best rank-k approximation can be obtained using SVD
    - But require access to the entire matrix A

Randomized algorithms

- Friedland, Mehrmann, Miedlar, Nkengla (2011) …
  - Choose p and t_{max}
  - Repeat t_{max} times
    - Generate index sets I and J of size p at random
    - Determine numerical rank r_{IJ} of A(I,J)
    - Compute π_{IJ} = product of the first r_{IJ} singular values of A(I,J)
  - Consider those A(I,J) for which r_{IJ} are the largest and pick the one such that π_{IJ} is the largest. Compute the best rank-k approximation of this particular A(I,J) … denote by A_{IJk}
    - Let C = A(:,J) and R = A(I,:)
    - Let B be the pseudo-inverse of A_{IJk}
    - Use CBR as a rank-k approximation of A

Randomized algorithms

- Does it really work?
  - Apparently work on matrices from image processing
  - Can be extended to tensors
- Advantages …
  - Do not need entire A; just need to be able to sample A
  - Completely parallel
    - Can start different sequences of samples in parallel
    - Can try different t_{max}
- Open questions …
  - Other matrix problems? Other scientific problems?
  - Other randomization/sampling techniques?
  - Robustness, reliability, accuracy?
  - What if the approach fails?

Communication reduction

- Communication is becoming more and more expensive relative to computation
  - Either moving data within the local memory system or across the network in a distributed memory setting
- Important to design algorithms to reduce the amount of communication as much as possible

- Example …
  - QR factorization of a tall, skinny matrix

Communication reduction in QR factorization

- Demmel, Grigori, Hoemmen, Langou (2008)
  - Consider computing the QR factorization of an m x n dense matrix A, where m >> n
  - TSQR (Tall Skinny QR):
    - Orthogonal reductions based on a binary tree
    - Partition rows of A into blocks and compute QR factorization of each block
    - Reduce the triangular factors in a pairwise fashion
    - Then continue the reduction repeatedly until only one triangular factor is left

$$A = \begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \\ \vdots \\ A_{s-3} \\ A_{s-2} \\ A_{s-1} \\ A_{s} \end{bmatrix} \rightarrow \begin{bmatrix} R_{11} \\ R_{21} \\ R_{31} \\ R_{41} \\ \vdots \\ R_{s-3,1} \\ R_{s-2,1} \\ R_{s-1,1} \\ R_{s,1} \end{bmatrix}$$

Communication reduction in QR factorization

$$A = \begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \\ \vdots \\ A_{s-3} \\ A_{s-2} \\ A_{s-1} \\ A_{s} \end{bmatrix} \rightarrow \begin{bmatrix} R_{11} \\ R_{21} \\ R_{31} \\ R_{41} \\ \vdots \\ R_{s-3,1} \\ R_{s-2,1} \\ R_{s-1,1} \\ R_{s,1} \end{bmatrix} \rightarrow \begin{bmatrix} R_{12} \\ R_{32} \\ \vdots \\ R_{s-3,2} \\ R_{s-1,2} \end{bmatrix} \rightarrow \begin{bmatrix} R_{13} \\ \vdots \\ R_{s-3.3} \end{bmatrix} \rightarrow \cdots \rightarrow R$$

- New?
  - Maybe … proposed in 80's for sparse matrix computation
- More storage if Q is needed; more computation

Communication reduction in QR factorization

- Complexity …

  - P processors, 1D mapping, counting along critical path

	TSQR	ScaLAPACK
# messages	$\log(P)$	$2n \log(P)$
# words	$\frac{1}{2}[n^2 \log(P)]$	$\frac{1}{2}[n^2 \log(P)]$
# flops	$(1/P)[2mn^2] + \frac{2}{3}[n^3 \log(P)]$	$(1/P)[2mn^2] + \frac{1}{2}[n^2 \log(P)]$

  - m = 100,000, $\lceil n/\sqrt{P} \rceil = 50$, time in seconds

P	TSQR	ScaLAPACK
1	9.68	12.63
2	15.71	19.88
4	16.07	19.59
8	11.41	17.85
16	9.75	17.29
32	8.15	16.95
64	9.46	17.74

Synchronization reduction

- Synchronizations can be become bottlenecks
  - Known for a long time
  - But may become worse under exascale

- Example …
  - The conjugate gradient algorithm
    - An iterative method for solving sparse system of linear equations
    - Rely on matrix-vector multiplication and inner products

$$
\begin{aligned}
\gamma_k &= \langle r_k, r_k \rangle \\
\beta_k &= \gamma_k / \gamma_{k-1} \\
p_k &= r_k + \beta_k p_{k-1} \\
v_k &= A p_k \\
\sigma_k &= \langle p_k, v_k \rangle \\
\alpha_k &= \gamma_k / \sigma_k \\
x_{k+1} &= x_k + \alpha_k p_k \\
r_{k+1} &= r_k - \alpha_k v_k
\end{aligned}
$$

one step of the conjugate gradient algorithm

One step of the conjugate gradient algorithm

$$\gamma_k = \langle r_k, r_k \rangle$$
$$\beta_k = \gamma_k / \gamma_{k-1}$$
$$p_k = r_k + \beta_k p_{k-1}$$
$$v_k = A p_k$$
$$\sigma_k = \langle p_k, v_k \rangle$$
$$\alpha_k = \gamma_k / \sigma_k$$
$$x_{k+1} = x_k + \alpha_k p_k$$
$$r_{k+1} = r_k - \alpha_k v_k$$

$$s_k = A r_k$$
$$\gamma_k = \langle r_k, r_k \rangle$$
$$\delta_k = \langle r_k, s_k \rangle$$
$$\beta_k = \gamma_k / \gamma_{k-1}$$
$$p_k = r_k + \beta_k p_{k-1}$$
$$v_k = s_k + \beta_k v_{k-1}$$
$$\sigma_k = \delta_k - \beta_k^2 \sigma_{k-1}$$
$$\alpha_k = \gamma_k / \sigma_k$$
$$x_{k+1} = x_k + \alpha_k p_k$$
$$r_{k+1} = r_k - \alpha_k v_k$$

- Also not new …
  - D'Azevedo, Eijkhout, Romine (1993)
    - The two are mathematically equivalent
    - Based on identities in conjugate gradient
- There are other variants, but not all have the same numerical behavior

Comm/Sync avoiding/reducing algorithms

- Notes …
  - Some of the ideas in some of these algorithms are not entirely new, but being re-discovered
  - It's often the case that such algorithms may require more memory and/or more computation
  - Some algorithms have communication/synchronization complexities that match lower bounds (Demmel's group)
  - In some cases, the algorithms may not be as stable as conventional algorithms
- Open questions …
  - New algorithms that require less communication/synchronization?
  - Can an existing algorithm be reformulated to reduce communication/synchronization?
  - Numerical behavior of such algorithms?

Fine-grained parallel algorithms

- Exascale computing promises high degree of parallelism
- Fine-grained parallel algorithms for matrix problems?
- Probably need to come up with out-of-the-box ideas

- Example …
  - Compute an incomplete LU factorization
  - Traditional approaches – incomplete version of Gaussian elimination
  - Chow (2014)
    - All nonzero entries of L and U are computed in parallel and asynchronously
    - Let S be the desired sparsity pattern of L+U

Fine-grained ILU

- ❑ Compute $L_{ij}, i > j, (i,j) \in S$, $U_{ij}, i \le j, (i,j) \in S$

  subject to $\sum_{k=1}^{\min(i,j)} L_{ik} U_{kj} = A_{ij}, (i,j) \in S$

- ❑ This results in $L_{ij} = \frac{1}{U_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \right), \quad i > j$

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj}, \quad i \le j$$

  which is just a nonlinear equation of the form x = G(x)

  - Starting with an initial guess of L and U, one can iterate until convergence
  - In the extreme case, each L_{ij}/U_{ij} can be assigned to one processing unit and computed asynchronously, leading to a very fine-grained parallel algorithm

Fine-grained parallel algorithms

- Results …
  - See Hittinger's talk
- Advantages …
  - Since L, U are incomplete factors, really no need to compute them accurately ==> just a few iteration may be enough
  - Possibility of exploiting a lot of cores
- Open questions …
  - Similar fine-grained algorithms for other matrix problems?
  - Techniques for solving the nonlinear equations?
    - Converge to the desired solution?

Resilience

- Resilience is concerned with dealing with and recovering from faults

- Example …
  - Suppose we are solving a linear system

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

  - Suppose there is a fault and x_1 needs to be recovered
  - Assume that x_2 is known and the value is trustworthy
  - How can x_1 be recovered?

Resilient linear solvers

- Langou, Chen, Bosilca, Dongarra (2007)
  - Linear interpolation: Solve $A_{11}\, x_1 = b_1 - A_{12}\, x_2$
  - A-norm of forward error associated with iterates computed by restarted CG or PCG is monotonically decreasing
- Giraud et al (2014)
  - Least squares interpolation

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} x_1 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} - \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} x_2$$

  - Solve for x_1 as a least squares problem
  - Monotonic decrease of residual norm of minimal residual Krylov subspace methods after restart

Resilient linear solvers

- Techniques can be extended to multiple faults
- Similar ideas can be applied to eigenvalue problems

- Open problems …
  - Resilient algorithms for other matrix problems?
  - Numerical behavior of such algorithms?
  - What to do if recovery fails?

Summary

- Challenges along the path to exascale …
  - High degree of parallelism
  - High communication & synchronization overhead
  - Deep memory hierarchy
  - Limited memory
  - Resilience

- What we need to overcome these challenges …
  - Some existing approach may evolve
  - Re-visit old ideas
  - Need new and out-of-the-box ideas

Summary

- Research opportunities …
  - Fine-grained parallel algorithms
  - Communication and synchronization avoiding/reduction algorithms
  - Algorithms based on randomization and sampling
  - Multiple-precision algorithms
  - Use of data compression
  - Resilient algorithms

- One of the common themes …
  - Robustness, reliability, accuracy