Enabling Tools for Extreme Scale Computation of Nanoscale Fluids

David Day, Amalie Frischknecht, Michael Heroux, Michael Parks
Sandia National Laboratories

Laura Frink
Colder Insights Corp.

Deaglan Halligan
Purdue University

Kirk Soodhalter
Temple University

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
Overview

- Test several new algorithmic capabilities in Sandia’s Tramonto Fluid DFT code
  - Ability to solve fluid-DFT governing equations in 3D and at large scales crucial to continued scientific progress.

- To realize promised performance of modern high-end multicore systems, we must develop new algorithmic capabilities to efficiently utilize multicore nodes
  - Increase performance by reducing node-level memory bandwidth and size usage

- **Mixed-precision and precision-neutral algorithms**
  - Leverage Trilinos/Tpetra (templated C++) solver stack
  - Performance and storage advantage of float over double
  - Utilize high-precision arithmetic if double inadequate

- **Least-squares methods (LSQR)**
  - Achieve robustness by dynamically adapting precision
  - Shield user from details of mixed-precision computation

- **Block Krylov recycling methods**
  - Recycling subspace information from previous solves to reduce iteration count
  - Block methods have superior convergence properties and computation to bandwidth requirements, improving processor utilization
Nanostructured Fluids

- Structure arises from surfaces, fields, self-assembly
- Density, diffusion, and viscosity different from bulk fluid properties
- Rich phase behavior: wetting, capillary condensation, layering

**Biological Membranes**
Self-assembled fluid bilayer packed with proteins, peptides, etc.

**Engineered Systems**
Lipid vesicle/nanoparticle assemblies for drug delivery

Density Functional Theory for Fluids

- Enable modeling and simulation of a wide range of applications, including fluids at interfaces, colloidal fluids, wetting, porous media, and biological mechanisms at the cellular level.
- Given external field $V(r)$, determine structure of inhomogeneous fluid as captured by density distribution $\rho(r)$ via minimization of free energy functional $\Omega(\rho(r))$.

$$\Omega[\rho(r)] = F_{id} + F_{hs} + F_{vdW} + F_c + F_{assoc} + \int \rho(r)[V(r)-\mu]$$

- Solve $\left(\frac{\delta \Omega}{\delta \rho(r)}\right)_{T,\mu} = 0$ with Newton-Krylov.
- Use Sandia’s Tramonto package for complex fluid systems.
  - Built upon Trilinos software components: trilinos.sandia.gov
  - Open source: software.sandia.gov/tramonto/
Discrete formulation
- Uniform structured grid
- Discretize using collocation at mesh points
- Linear interpolation between mesh points

Newton: Convergence in $O(10)$ iterations

Linear system properties (different than discrete PDEs)
- Strong interphysics coupling
- Large number of DOF/node
- Nonlocal integral equations
  (matrix sparsity dependent upon mesh)

Coarse Mesh
(Few nonzero per row)

Fine Mesh
(Many nonzero per row)
Segregated Schur Complement Solvers*

- Resulting linear systems take the form

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

Each \(A_{ij}\) has own physics-based block structure

- Careful ordering of unknowns makes it advantageous to solve Schur complement

\[Sx_2 = f\]

where

\[S = A_{22} - A_{21}A_{11}^{-1}A_{12}\]
\[f = b_2 - A_{21}A_{11}^{-1}b_1\]

- Schur system may have up to 80% fewer dofs
- Big win for hard sphere systems: \(A_{11}\) is diagonal!
- Similar favorable structure to \(A_{11}\) for polymer problems using Chandler-McCoy-Singer (CMS) DFT
- More complex structure for WJDC (Werthim, Jain, Dominik, and Chapman) DFT

---

Enable Mixed-Precision and Precision Neutral Computation

- Rewrite Tramonto solver managers to template scalar, local ordinal, and global ordinal types (templated C++)
  - Arbitrary scalar types: float, complex, dd_real, qd_real (high precision)
  - Utilize high-precision arithmetic if double precision inadequate
  - Avoid 4GB limit of int - allow arbitrarily large problems (exascale necessity)
  - Enhance performance while maintaining solution accuracy

- Template scalar type through solver stack
## Precision Neutral Computation

- Reduce node-level memory bandwidth and size usage
  - Replace double with float

- Example polymer problem from *Tramonto* (8 linear solves inside Newton loop)

<table>
<thead>
<tr>
<th>NCore</th>
<th>Float</th>
<th>Double</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.753</td>
<td>10.970</td>
<td>2.923</td>
</tr>
<tr>
<td>2</td>
<td>1.766</td>
<td>4.195</td>
<td>2.375</td>
</tr>
<tr>
<td>3</td>
<td>1.203</td>
<td>2.086</td>
<td>1.734</td>
</tr>
<tr>
<td>4</td>
<td>1.380</td>
<td>2.643</td>
<td>1.915</td>
</tr>
<tr>
<td>5</td>
<td>1.211</td>
<td>2.460</td>
<td>2.031</td>
</tr>
<tr>
<td>6</td>
<td>1.056</td>
<td>2.313</td>
<td>2.190</td>
</tr>
<tr>
<td>7</td>
<td>1.036</td>
<td>2.057</td>
<td>1.986</td>
</tr>
<tr>
<td>8</td>
<td>1.524</td>
<td>2.387</td>
<td>1.566</td>
</tr>
</tbody>
</table>
LSQR

- LSQR*
  - Implemented in Trilinos/Belos package (C++, templated)
  - Krylov method for $Ax=b$ based upon Golub-Kahan bidiagonalization process
  - Algebraically equivalent to MINRES applied to normal equations $A^H Ax = b$, but with better numerical properties (especially if $A$ ill-conditioned)

- Governing equations
  \[ A^H U_k = V_k B_k^H \]
  \[ \text{span}(U_k) = \mathcal{K}(AA^H, b) \]
  \[ AV_k = U_{k+1} \bar{B}_k \]
  \[ \text{span}(V_k) = \mathcal{K}(A^H A, A^H b) \]
  \[ \|b - Ax_k\| = \min_y \|b - AV_k y\| = \min_y \|e_1 \beta - \bar{B}_k y\| \]

- Short-term recurrence; Fixed memory-footprint
- Sharp estimates of $\|A\|$, $\|A^{-1}\|$ -> estimate of $\text{cond}(A)$

- Robustness under reduced precision
  - Return least-squares solution to $Ax=b$ even is $A$ numerically singular due to use of lower precision

Balance speed and solution accuracy by dynamically adapting solver precision

1. Solve $Ax=b$ in float
2. If $\text{condest}(A) < \text{machEpsSingle}$ return
3. Else solve $Ax=b$ in double
4. If $\text{condest}(A) < \text{machEpsDouble}$ return
5. Else solve $Ax=b$ in double-double
6. If $\text{condest}(A) < \text{machEpsDouble-Double}$ return
7. ... 

Shield end user from details of adaptive precision!

Adaptive precision example with LSQR

- Case #1: Well-conditioned matrix (nonsingular in float)
  - Requested relative residual tolerance = $5e^{-4}$

<table>
<thead>
<tr>
<th>Scalar Type</th>
<th>Solve Time (s)</th>
<th># Iters</th>
<th>CondTest</th>
<th>Residual Norm</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>1.049</td>
<td>826</td>
<td>Nonsingular</td>
<td>$4.98e^{-4}$</td>
<td>Success</td>
</tr>
</tbody>
</table>

- Case #2: Ill-conditioned matrix (singular in float, nonsingular in double)
  - Requested relative residual tolerance = $1e^{-6}$

<table>
<thead>
<tr>
<th>Scalar Type</th>
<th>Solve Time (s)</th>
<th># Iters</th>
<th>CondTest</th>
<th>Residual Norm</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>8.155</td>
<td>528</td>
<td>Singular</td>
<td>$9.80e^{-6}$</td>
<td>Failure</td>
</tr>
<tr>
<td>double</td>
<td>107.568</td>
<td>4658</td>
<td>Nonsingular</td>
<td>$9.99e^{-7}$</td>
<td>Success</td>
</tr>
</tbody>
</table>
Leverage two important algorithmic techniques: Krylov recycling + block methods

Krylov subspace recycling
- In Krylov subspace methods, building search space is dominant cost
- For sequences of systems, get fast convergence rate and good initial guess immediately by recycling selected search spaces from previous systems
- Family of recycling methods: Recycling GMRES (GCRODR), recycling CG (RCG), recycling MINRES (RMINRES), recycling BiCG (RBiCG).

Block methods
- Performance advantages over single-vector methods (BLAS 1 → BLAS3, SpMV → SpMM)
- Reduce per-core bandwidth usage
- Introduce fictitious right-hand-sides to enhance search space
Block Recycling GMRES (BGCRODR)

- **Block Recycling GMRES**  
  - Implemented in Trilinos/Belos package (C++, templated)

  1. Solve \( A_1X_1 = B_1 \)
  2. Compute \( k \) recycle vectors \( U_k \) (for example, harmonic Ritz vectors)
  3. Solve next linear system \( A_2X_2 = B_2 \) by iterating orthogonally to image of \( U_k \):

\[
A_2 \begin{bmatrix} U_k & W_m \end{bmatrix} = \begin{bmatrix} C_k & W_{m+1} \end{bmatrix} \begin{bmatrix} I_k & B_k \\ 0 & H_m \end{bmatrix}
\]

\[
B_k = C_k^H A W_m \quad C_k = A_2 U_k \quad C_k^H C_k = I_k
\]

4. Repeat

- Example hard sphere problem from Tramonto (electrostatics + attractions)
- 7 linear solves in from Newton loop
- Savings: 60 matvecs / 36% (1 RHS), 50 matvecs / 40%, (3 RHS)

![BGCRODR on Tramonto Polymer Example](Sandia National Laboratories)
Summary

- Tested several new algorithmic capabilities in Sandia’s Tramonto Fluid DFT code
- **Improved performance via reduction of node-level memory bandwidth and size usage**

- **Enabling mixed-precision and precision-neutral algorithms**
  - Leverage Trilinos (templated C++) solver stack
  - 2x or more speedup with float instead of double
  - High-precision arithmetic if double inadequate

- **Least-squares methods (LSQR)**
  - Achieve robustness by dynamically adapting precision
  - Shield user from details of adaptive precision computation

- **Block Krylov recycling methods**
  - Recycling subspace information from previous solves to reduce iteration count
  - Block methods have superior convergence properties and computation to bandwidth requirements, improving processor utilization