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

SAND2011-7515P

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

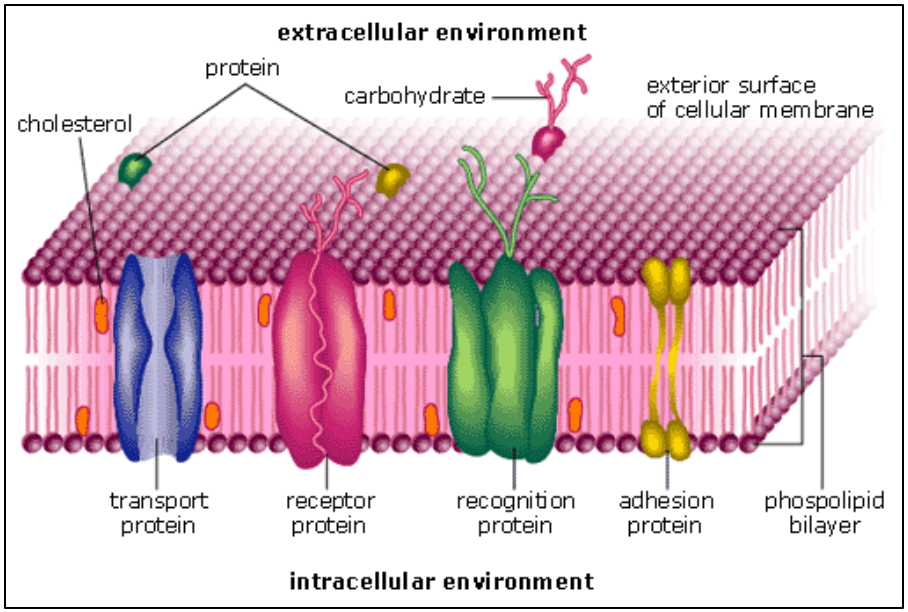
- **Mixed-precision and precision-neutral algorithms**
  - Leverage Trilinos/Tpetra (templated C++) solver stack
  - Performance and storage advantage of float over double
  - Utilize high-precision arithmetic if double inadequate

- **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

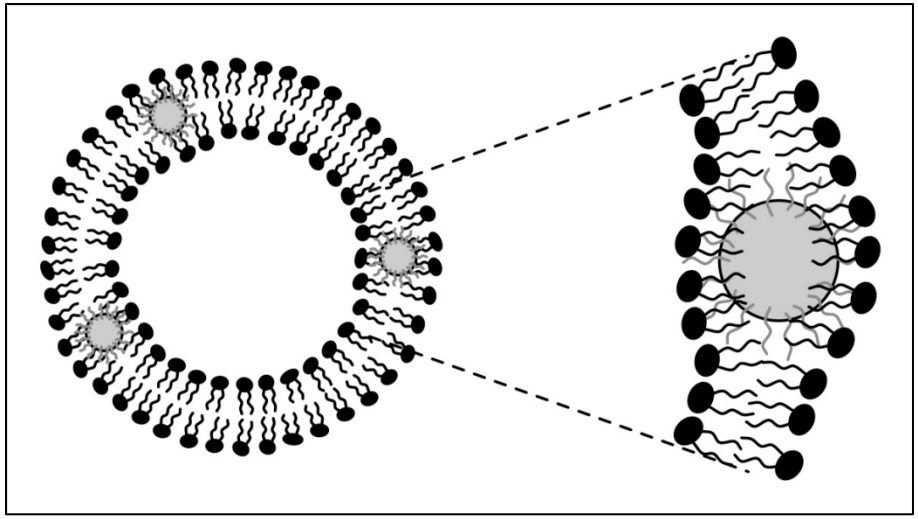


Biological Membranes

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

Engineered Systems

Lipid vesicle/nanoparticle assemblies for drug delivery)



[*] G.D. Bothun, Hydrophobic silver nanoparticles trapped in lipid bilayers: Size distribution, bilayer phase behavior, and optical properties, *J. Nanobiotechnology*, 6, (2008).

Sandia National Laboratories

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 **ρ(r)** via minimization of free energy functional **Ω(ρ(r))**

Coulomb interactions; Associations (H-bonding); [Applied field]

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

Ideal gas; Hard sphere; Dispersion attractions

Legendre transform from Canonical to Grand Canonical ensemble

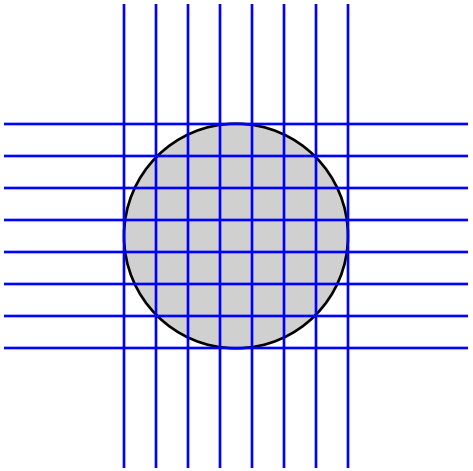
- Solve $\left(\frac{\delta\Omega}{\delta\rho(\mathbf{r})}\right)_{T,\mu} = \mathbf{0}$ with Newton-Krylov.

- Use Sandia's ***Tramonto*** package for complex fluid systems
  - Built upon ***Trilinos*** software components: *trilinos.sandia.gov*
  - Open source: *software.sandia.gov/tramonto/*

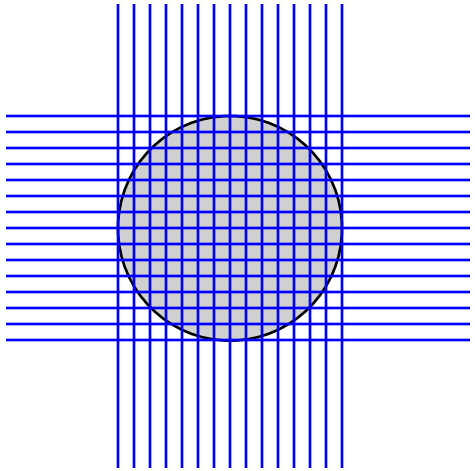
Sandia National Laboratories

Numerical Methods for Fluid-DFTs

- 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

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$$

Each $\mathbf{A}_{ij}$ has own physics-based block structure

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

$$\mathbf{S}\mathbf{x}_2 = \mathbf{f}$$

where

$$\mathbf{S} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \qquad \mathbf{f} = \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1$$

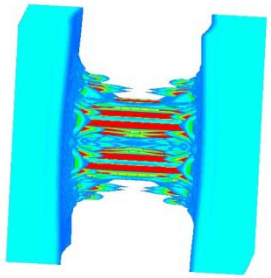
- **Schur system may have up to 80% fewer dofs**
- Big win for hard sphere systems: $\mathbf{A}_{11}$ is diagonal!
- Similar favorable structure to $\mathbf{A}_{11}$ for polymer problems using Chandler-McCoy-Singer (CMS) DFT
- More complex structure for WJDC (Werthim, Jain, Dominik, and Chapman) DFT

* M.A. Heroux, A.G. Salinger, and L.J.D. Frink, *Parallel segregated Schur complement methods for fluid density functional theories,* SIAM J. Sci. Comput., 29, (2007). pp. 1526-1535.

Sandia National Laboratories

Enabling 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**



Fluid-DFT Applications

Tramonto

Fluid DFT Code (C++, Templated)

Scalar Type

Trilinos

Algorithms and Enabling Technologies

Scalar Type

BELOS

Linear Solvers (C++, Templated)

Scalar Type

Tpetra parallel linear algebra library C++, Templated)

Scalar Type

TEUCHOS

Tools Library BLAS/LAPACK wrappers (C++, Templated)

Sandia National Laboratories

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)

NCore	Float	Double	Speedup
1	3.753	10.970	2.923
2	1.766	4.195	2.375
3	1.203	2.086	1.734
4	1.380	2.643	1.915
5	1.211	2.460	2.031
6	1.056	2.313	2.190
7	1.036	2.057	1.986
8	1.524	2.387	1.566

LSQR

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

- Governing equations

$$\mathbf{A^H U_k = V_k B_k^H} \qquad \mathbf{span(U_k) = \mathcal{K}(AA^H, b)}$$

$$\mathbf{A V_k = U_{k+1} \bar{B}_k} \qquad \mathbf{span(V_k) = \mathcal{K}(A^H A, A^H b)}$$

$$\left\| \mathbf{b - A x_k} \right\| = \min_{\mathbf{y}} \left\| \mathbf{b - A V_k y} \right\| = \min_{\mathbf{y}} \left\| \mathbf{e_1 \beta - \bar{B}_k y} \right\|$$

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

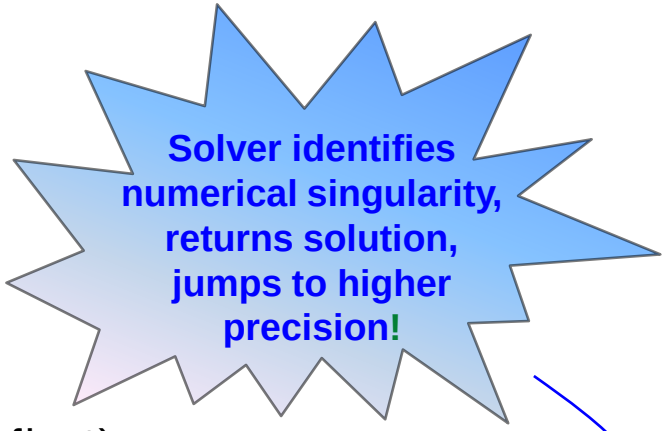
- **Robustness under reduced precision**
  - Return least-squares solution to Ax=b even is A numerically singular due to use of lower precision

Sandia National Laboratories

* C. C. Paige and M. A. Saunders, LSQR: An algorithm for sparse linear equations and sparse least squares, TOMS 8(1), 43-71 (1982).

LSQR

- **Balance speed and solution accuracy by dynamically adapting solver precision**
  - (1) Solve Ax=b in float
  - (2) If condest(A) < machEpsSingle return
  - (3) Else solve Ax=b in double
  - (4) If condest(A) < machEpsDouble return
  - (5) Else solve Ax=b in double-double
  - (6) If condest(A) < 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



Scalar Type	Solve Time (s)	# Iters	CondTest	Residual Norm	Outcome
float	1.049	826	Nonsingular	4.98e-4	Success

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

Scalar Type	Solve Time (s)	# Iters	CondTest	Residual Norm	Outcome
float	8.155	528	Singular	9.80e-6	Failure
double	107.568	4658	Nonsingular	9.99e-7	Success

Sandia National Laboratories

Block Recycling Linear Solvers

- 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 $\rightarrow$ BLAS3, SpMV $\rightarrow$ SpMM)
  - Reduce per-core bandwidth usage
  - Introduce fictitious right-hand-sides to enhance search space

Block Recycling GMRES (BGCRODR)

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

(1) Solve $\mathbf{A_1X_1=B_1}$

(2) Compute **k** recycle vectors $\mathbf{U_k}$ (for example, harmonic Ritz vectors)

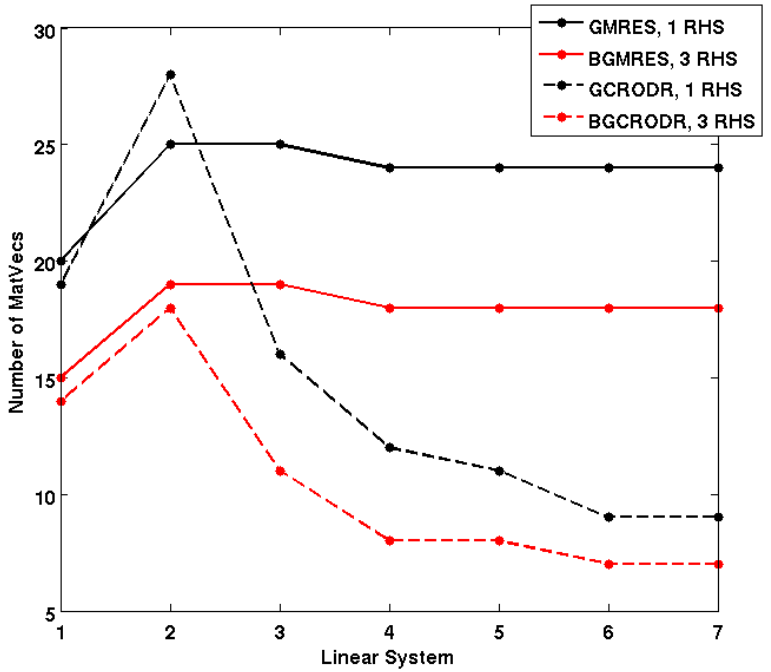
(3) Solve next linear system $\mathbf{A_2X_2=B_2}$ by iterating orthogonally to image of $\mathbf{U_k}$:

$$\mathbf{A_2}\begin{bmatrix}\mathbf{U_k} & \mathbf{W_m}\end{bmatrix}=\begin{bmatrix}\mathbf{C_k} & \mathbf{W_{m+1}}\end{bmatrix}\begin{bmatrix}\mathbf{I_k} & \mathbf{B_k}\\ \mathbf{0} & \underline{\mathbf{H}}_{\mathbf{m}}\end{bmatrix}$$

$$\mathbf{B_k}=\mathbf{C_k^H A W_m} \quad \mathbf{C_k}=\mathbf{A_2U_k} \quad \mathbf{C_k^H C_k}=\mathbf{I_k}$$

(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