Ginla: A Trilinos-based solver for the Ginzburg–Landau problem

Eurotug 2012, Lausanne

Nico Schlömer   Wim Vanroose

Universiteit Antwerpen

Nonlinear Schrödinger equations

Find energy minimizers $\psi : \Omega \to \mathbb{C}$

$$\psi_{\text{min}} = \operatorname{argmin} \mathcal{G}(\psi) = \operatorname{argmin} \int_\Omega f(|\psi|^2).$$

Necessary:

$$\begin{cases} 0 = (\mathcal{K} + V + g|\psi|^2)\psi \\ \text{boundary conditions} \end{cases}$$

with

- $\mathcal{K}$ linear, Hermitian, positive definite,
- potential $V : \Omega \to \mathbb{R}$,
- coupling parameter $g \in \mathbb{R}$.

Nonlinear Schrödinger equations

- *The* Nonlinear Schrödinger equation

$$0 = \left(-\frac{1}{2}\Delta + \kappa|\psi|^2\right)\psi$$

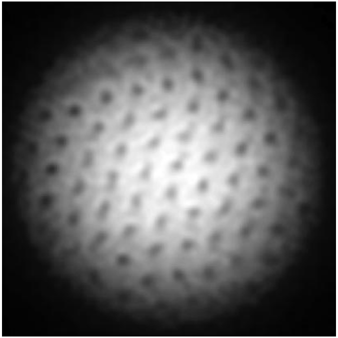
- Gross–Pitaevskii

$$0 = \left(-\Delta + V + \mathsf{g}|\psi|^2\right)\psi$$

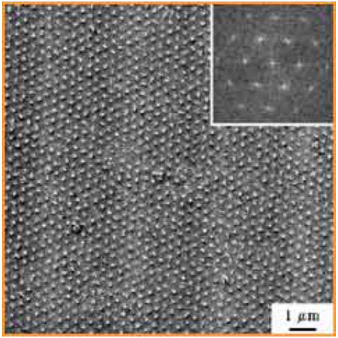
- Ginzburg–Landau

$$0 = (-\mathrm{i}\nabla - \mathbf{A})^2\psi - (1 - |\psi|^2)\psi$$

NLS samples



(a) Rotating superfluid (Bose–Einstein condensate).

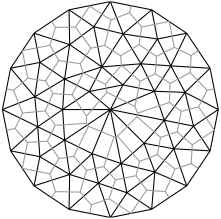


(b) Abrikosov lattice in a crystal (superconductivity).

Appropriate discretizations

Define

a triangulation of $\mathcal{T}$ and a tesselation $\mathcal{P}$ of Ω_h



$$\Omega_h = \bigcup_{k=1}^{m} T_k = \bigcup_{i=1}^{n} P_i.$$

Then

$$\int_{\Omega_k} S(\psi) = \sum_k \int_{T_k} \mathcal{K}\psi + \sum_i \int_{V_i} (V + |\psi|^2)\psi$$

$$0 \stackrel{!}{=} \sum_{\text{edges } e_k} \alpha_k(\psi_{k_0} - \beta\psi_{k_1}) + \sum_i \underbrace{|P_i|(V(\mathbf{x}_i) + |\psi_i|^2)\psi_i}_{\text{“mass lumping”}}$$

Some analytic properties of the operators

- Energy

$$\mathcal{G}(\psi) = \int_\Omega \mathsf{f}(|\psi|^2).$$

  only depends on **density** $|\psi|^2$.
  Two states ψ, $\exp(\mathrm{i}\alpha)\psi$ are *physically* equivalent.
  $\implies \psi : \omega \mapsto \mathbb{C}$; restrict the scalar field to $\mathbb{R}$.

$$\langle \psi, \phi \rangle = \Re \left(\int_\Omega \overline{\psi}\phi \right)$$

  Linearization

$$\mathsf{J}(\psi)\phi = (\mathcal{K} + \mathsf{V} + 2|\psi|^2)\phi + \psi^2\overline{\phi}.$$

  is actually linear.

Intermission

Real-valued vs. complex-valued formulation

Solving the complex-valued linear system

$$Ax = b,$$

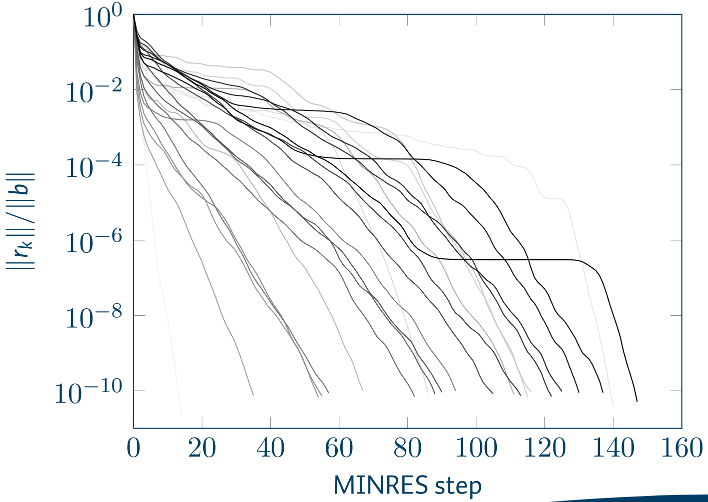
over $\mathbb{C}$ with anything that involves inner products (e.g., Krylov solvers), is **not** equivalent to solving the real-valued system

$$\begin{pmatrix} \Re(A) & -\Im(A) \\ \Im(A) & \Re(A) \end{pmatrix} \begin{pmatrix} \Re(x) \\ \Im(x) \end{pmatrix} = \begin{pmatrix} \Re(x) \\ \Im(x) \end{pmatrix}. \tag{1}$$

Better

Solving (1) is equivalent to solving $Ax = b$ over $\mathbb{R}$.

Typical residual curves for the Jacobian



Some analytic properties of the operators (cont'd)

- Energy

$$\mathcal{G}(\psi) = \int_\Omega \mathsf{f}(|\psi|^2).$$

  only depends on **density** $|\psi|^2$.

  $\Longrightarrow$

$$\mathcal{S}(\psi) = 0 \quad \Longleftrightarrow \quad \mathcal{S}(\exp(\mathrm{i}\alpha)\psi) = 0,$$

  $\Longrightarrow$

$$\mathcal{S}(\psi) = 0 \quad \Longleftrightarrow \quad \dim\ker(\mathcal{J}(\psi)) > 0.$$

- Possible remedies:
  - phase condition
  - deflation

Algorithm

initialize system $\mathcal{S}(\mu_0)$ with ψ_0
while *true* **do**
  corrector initial guess ψ_k: Newton (NOX)
  while *Newton not converged* **do**
    Solve $J(\psi_k)x = -S(\psi_k)$
    using preconditioned (ML), deflated MINRES (Belos).
  end
  predict new solution ψ_k at μ_k (LOCA)
end

Algorithm 1: NLS parameter continuation

Why Trilinos?

Number of unknowns for 3D systems: 10^6, 10^9

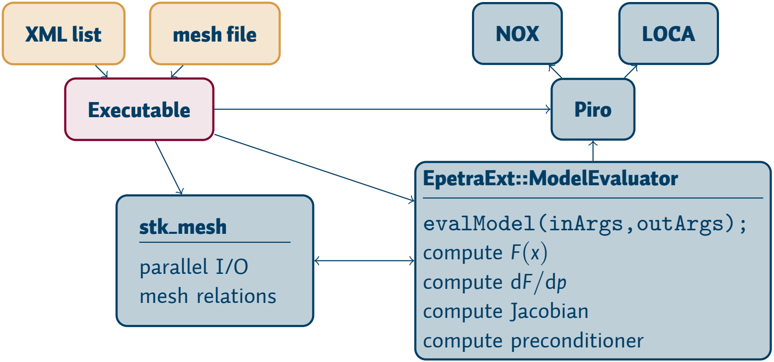
$\Longrightarrow$ A serial implementation is not sufficient.

Parallel tool kits:

- PetSC (mainly C/C++, Fortran)
- Trilinos (mainly C++)
- FEniCS (Python, parallelization through PetSC/Epetra)
- ...

Nonlinear solver capabilities made the difference.

Code structure



XML input file

```
[...]
  <ParameterList name="LOCA">
    <ParameterList name="Bifurcation"/>
    <ParameterList name="Constraints"/>
    <ParameterList name="Predictor">
      <Parameter name="Method" type="string" value="Tangent"/>
      <!--Parameter name="Method" type="string" value="Secant"/
    </ParameterList>
    <ParameterList name="Stepper">
      <Parameter name="Continuation Method" type="string" value
      <Parameter name="Initial Value" type="double" value="1.0"
      <Parameter name="Continuation Parameter" type="string" va
      <Parameter name="Max Steps" type="int" value="10"/>
      <Parameter name="Max Value" type="double" value="100.0"/>
      <Parameter name="Min Value" type="double" value="-100.0"/
      <Parameter name="Compute Eigenvalues" type="bool" value="
      <ParameterList name="Eigensolver">
        <Parameter name="Maximum Restarts" type="int" value="3"
        <Parameter name="Method"
```

Common mesh file formats

Exodus/ExodusII

- based on netCDF, based on HDF5

+ possibility to store several (time) steps in one file, i.e., on one mesh
+ Exodus part of Trilinos

- A bunch of small things: Cannot store char arrays, underscores are added to field names,...

VTK formats (with its variants)

+ outstanding support through VTK (C++, Python)

- different states $\rightarrow$ different files

Creating the mesh

1. Generate a domain triangulation.
    - 2D: triangle,...
    - 3D: Gmsh, NETGEN, TetGen, MeshSim, CUBIT,...
    - Interfaces: MeshPy,...

   Unfortunately quite some fragmentation in this field.
2. Equip the mesh with an initial guess ψ_0, and optionally
    - vector field values $\mathbf{A}(\mathbf{x}_i)$, potential values $V(\mathbf{x}_i)$
3. *Optional* Split your mesh up in different files.

Creating the mesh (cont'd)

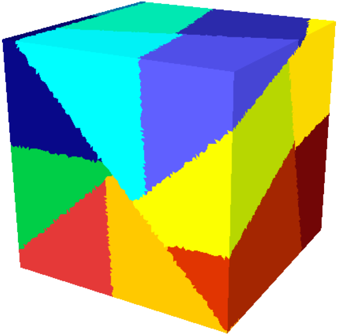
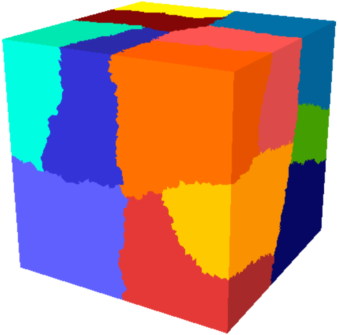
3 *Optional* Split your mesh up in different files, e.g., with Trilinos:

- Built `nemslice` (Chaco), `nemspread` in a *serial* build with

```
$ cmake \
    -D TPL_ENABLE_MPI:BOOL=OFF \
    -D Trilinos_ENABLE_STK:BOOL=ON \
    -D Trilinos_ENABLE_SEACASIoss:BOOL=ON \
    -D TPL_ENABLE_Netcdf:BOOL=ON
```

```
$ nem_slice -v -o "brick3holes-balanced.nemI" \
  -e -m mesh=4x4 "brick3holes.e"
# create nem_spread input file
$ nem_spread
```

Sliced cubes

- 
  (c) Inertial slicing
  
  (d) multi_kl slicing

stk_mesh

Q: Given the state files, how does the data get into Trilinos?
Parallel I/O packages in Trilinos Trios, stk/stk_mesh ("maintenence mode"?, integration with Trilinos?).

```
// Declare meta data object.
Teuchos::RCP<stk::mesh::fem::FEMMetaData> metaData =
    Teuchos::rcp(new stk::mesh::fem::FEMMetaData());
metaData->FEM_initialize(3);
Teuchos::RCP<stk::mesh::BulkData> bulkData =
  Teuchos::rcp(new stk::mesh::BulkData(
    stk::mesh::fem::FEMMetaData::get_meta_data( *metaData ),
    MPI_COMM_WORLD,
    1001)
  );
```

stkmesh: good for 3d meshes, can create faces/edges,...

Reading into stk_mesh (cont'd)

```
// Declare fields.
typedef stk::mesh::Field<double,stk::mesh::Cartesian>
    VectorFieldType;
typedef stk::mesh::Field<double> ScalarFieldType;

Teuchos::RCP<VectorFieldType> coordinatesField =
  Teuchos::rcpFromRef(
  metaData->declare_field<VectorFieldType>("coordinates"));
stk::io::set_field_role(*coordinatesField,
                        Ioss::Field::ATTRIBUTE);

Teuchos::RCP<VectorFieldType> mvpField =
  Teuchos::rcpFromRef(
  metaData->declare_field<VectorFieldType>("A"));
stk::mesh::put_field(*mvpField, metaData->node_rank(),
                     metaData->universal_part());
stk::io::set_field_role(*mvpField, Ioss::Field::TRANSIENT);
```

Reading into stk_mesh (cont'd)

```
// Declare fields.
typedef stk::mesh::Field<double,stk::mesh::Cartesian>
    VectorFieldType;
typedef stk::mesh::Field<double> ScalarFieldType;

Teuchos::RCP<VectorFieldType> coordinatesField =
  Teuchos::rcpFromRef(
  metaData->declare_field<VectorFieldType>("coordinates"));
stk::io::set_field_role(*coordinatesField,
                        Ioss::Field::ATTRIBUTE);

Teuchos::RCP<VectorFieldType> mvpField =
  Teuchos::rcpFromRef(
  metaData->declare_field<VectorFieldType>("A"));
stk::mesh::put_field(*mvpField, metaData->node_rank(),
                     metaData->universal_part());
stk::io::set_field_role(*mvpField, Ioss::Field::TRANSIENT);
```

Reading into stk_mesh (cont'd)

```
// Open database and set field separator.
Ioss::DatabaseIO *dbi =
    Ioss::IOFactory::create("exodusii", fileName_,
                            Ioss::READ_MODEL, MPI_COMM_WORLD);
dbi->set_field_separator( 0 );
Teuchos::RCP<stk::io::MeshData> meshData =
  Teuchos::rcp(new stk::io::MeshData());
meshData->m_input_region = in_region;
stk::io::create_input_mesh(meshType, fileName_,
                           MPI_COMM_WORLD, *metaData,
                           *meshData);
metaData->commit();

stk::io::populate_bulk_data(*bulkData, *meshData);
// Read the data.
stk::io::process_input_request(*meshData, *bulkData, index);
```

Using stk_mesh

```
// Get local nodes.
std::vector<stk::mesh::Entity*>
StkMesh::
getOwnedNodes() const
{
  stk::mesh::Selector select_owned_in_part =
      stk::mesh::Selector(metaData_->universal_part())
    & stk::mesh::Selector(metaData_->locally_owned_part());

  std::vector<stk::mesh::Entity*> ownedNodes;
  stk::mesh::get_selected_entities(
      select_owned_in_part,
      bulkData_->buckets( metaData_->node_rank() ),
      ownedNodes);

  return ownedNodes;
}
```

Using stk_mesh

```
// Get overlapping nodes.
std::vector<stk::mesh::Entity*>
StkMesh::
getOwnedNodes() const
{
  stk::mesh::Selector select_overlap_in_part =
       stk::mesh::Selector(metaData_->universal_part())
    & (stk::mesh::Selector(metaData_->locally_owned_part())
      |stk::mesh::Selector(metaData_->globally_shared_part()));

  std::vector<stk::mesh::Entity*> overlapNodes;
  stk::mesh::get_selected_entities(
      select_overlap_in_part,
      bulkData_->buckets( metaData_->node_rank() ),
      overlapNodes);

  return overlapNodes;
```

stk_mesh: What else is possible?

- query nodes, cells, node-cell relations
- 3D meshes: build “adjacent entities”, i.e., faces and edges

- build unique/overlapping node maps
- query fields (coordinates, node values,...)

$\rightarrow$ build {E,T}petra vectors

- no geometry helper functions (simplex volumes, circumcenters,...)

Good maps for m-v products?

range map = domain map

- $$A = \begin{pmatrix} 2 & 1 & \\ 1 & 2 & 1 \\ & & \end{pmatrix} + \begin{pmatrix} & & \\ & & \\ & 1 & 2 \end{pmatrix}, x = \begin{pmatrix} x_0 \\ x_1 \\ \; \end{pmatrix} + \begin{pmatrix} \; \\ \; \\ x_2 \end{pmatrix}$$
  Communicate $x_1 \to p_1, x_2 \to p_0$. Compute.
- $$A = \begin{pmatrix} 2 & 1 & \\ 1 & 1 & \\ & & \end{pmatrix} + \begin{pmatrix} & & \\ & 1 & 1 \\ & 1 & 2 \end{pmatrix}, x = \begin{pmatrix} x_0 \\ x_1 \\ \; \end{pmatrix} + \begin{pmatrix} \; \\ \; \\ x_2 \end{pmatrix}$$
  Communicate $x_1 \to p_1$. Compute. $y_{1b} \to p_0$.

Good maps for m-v products? (cont'd)

- $$A = \begin{pmatrix} 2 & 1 & \\ 1 & 1 & \\ & & \end{pmatrix} + \begin{pmatrix} & & \\ & 1 & 1 \\ & 1 & 2 \end{pmatrix}, x = \begin{pmatrix} x_0 \\ x_{1a} \\ \end{pmatrix} + \begin{pmatrix} \\ x_{1b} \\ x_2 \end{pmatrix}$$

  Communicate $x_{1a} \to p_1, x_{1b} \to p_0$. Compute.
- ...

Current restrictions by ML:

- range map $==$ domain map
- RowMatrixRowMap $==$ OperatorRangeMap

stk_mesh: Writing out data

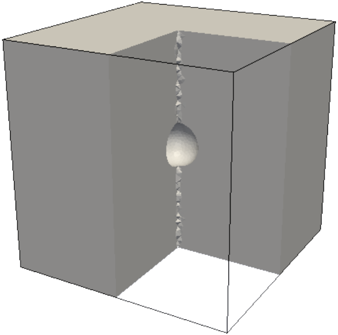
```
// Set owned nodes.
const std::vector<stk::mesh::Entity*> &ownedNodes =
    mesh->getOwnedNodes();
for (unsigned int k=0; k < ownedNodes.size(); k++)
{
  double* localPsiR = stk::mesh::field_data(*psi_field,
                                             *ownedNodes[k]);
  localPsiR[0] = psi[k];
}
// Owned values -> overlapping values
stk::mesh::parallel_reduce(*mesh->getBulkData(),
                           stk::mesh::sum(*psi_field));
```

Experiments: Parameter continuation

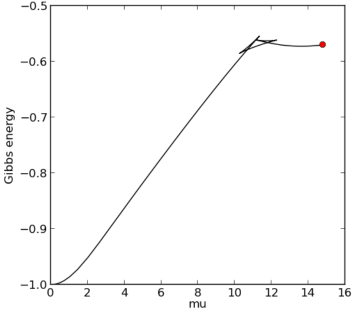
$$0 \stackrel{!}{=} F(\psi) = \begin{cases} (-\mathrm{i}\nabla - \mu \mathbf{A})^2 \psi + (1 - |\psi|^2)\psi \text{ on } \Omega \\ \mathbf{n} \cdot (-\mathrm{i}\nabla - \mu \mathbf{A})\psi \text{ on } \partial\Omega. \end{cases}$$

with

$$\mathbf{A}(\mathbf{x}) := \frac{1}{\|\mathbf{x} - \mathbf{x}_0\|^3} (\mathbf{m} \times (\mathbf{x} - \mathbf{x}_0))$$



A continuation example



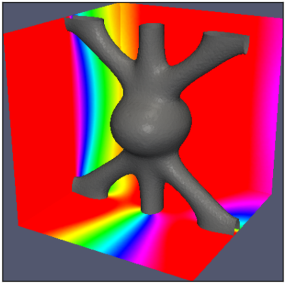


Figure: Typical continuation curve for μ**A** ($|\psi|^2 = 0.1$ isosurface, arg ψ on the boundaries.

Scalability

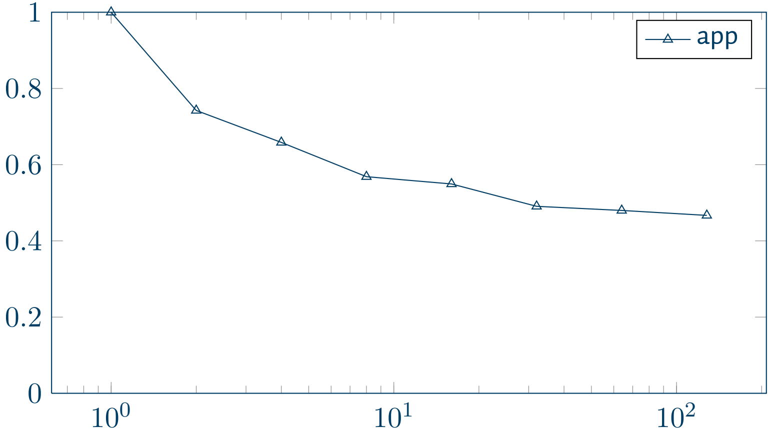


Figure: Scalability tests on NERSC's Hopper (Cray XE6, 12-core AMD 'MagnyCours' 2.1GHz processors).

Scalability

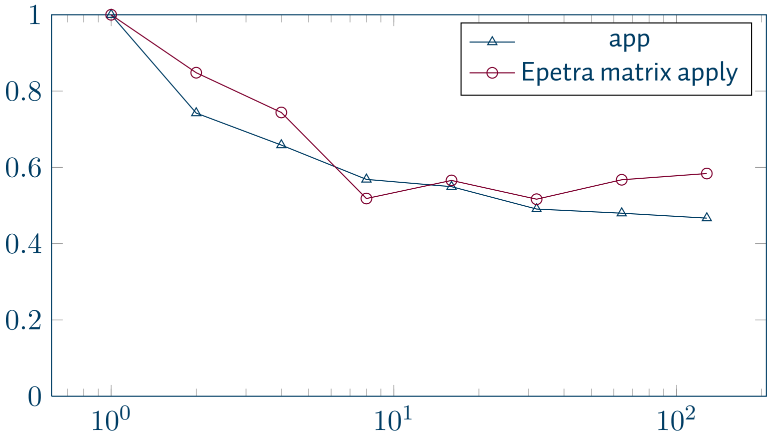


Figure: Scalability tests on NERSC's Hopper (Cray XE6, 12-core AMD 'MagnyCours' 2.1GHz processors).

Epetra vs. Tpetra

Packages not yet available for Tpetra:

- NOX, LOCA (to be ported 2012)
- ML/MueLu
- ModelEvaluator (?)

Extra benefits expected from Tpetra:

- code simplification
- reduction of matrix memory footprint ($A + iB$ vs. $\begin{pmatrix} A & B \\ -B & A \end{pmatrix}$)
- improved AMG aggregation

But **cannot** use complex notation as long as scalar product isn't customizable.

Lessons learned

- “C++ is many languages.”—Bjarne Stroustrup
- Don’t do *development* with Trilinos.
- Use standard formats.
- Unit/capability tests!!

Related publications

S., Avitabile, Vanroose
Numerical Bifurcation Study of Superconducting Patterns on a Square
SIAM Journal on Applied Dynamical Systems, 2012.

S., Vanroose
An optimal linear solver for the Jacobian system of the extreme type-II Ginzburg–Landau problem
Journal of Computational Physics, submitted.

S.
NLS: A scalable code for numerical computations of nonlinear Schrödinger problems
Archives of Numerical Software, submitted.

This research is sponsored by the Fonds Wetenschappelijk Onderzoek/Vlaanderen (FWO).