

Computational aspects of the EDF optimization

Jason Sarich

Mathematics and Computer Science Division,
Argonne National Laboratory

July 21, 2011

Collaborators: M. Kortelainen, T. Lesinski, J. More, W. Nazarewicz, N. Schunck, M. V. Stoitsov, and S. Wild

Optimization of Skyrme Energy Functionals

We want to find x that minimizes

$$\chi^2(x) = \frac{1}{n_d - n_x} \sum_{j=1}^N \left(\frac{s_j(x) - d_j}{w_j} \right)^2$$

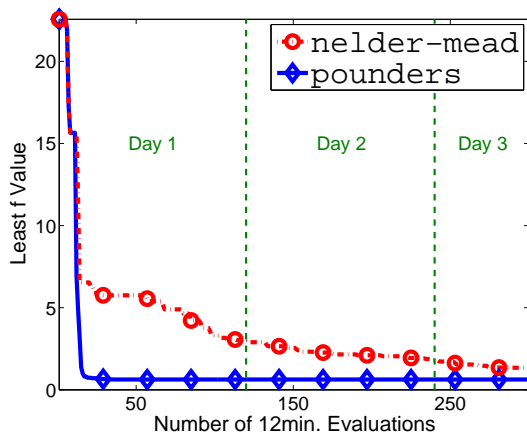
where

- x is the set of EDF parameters (ρ^{NM} , E^{NM}/A , K^{NM} , α_{sym}^{NM} , L_{sym}^{NM} , $1/M_s^*$, $C_0^{\rho\Delta\rho}$, $C_1^{\rho\Delta\rho}$, $V_0^{(\nu)}$, $V_0^{(\pi)}$, $C_0^{\rho\nabla J}$, $C_1^{\rho\nabla J}$)
- d_j is the experimental value of observable j from massexplorer.org
- s_j is the computed value of observable j using HFBTHO
- w_j is the weight attributed to observable j

Computational Challenges

- 79 nuclei to compute (Use MPI)
- Each takes approx. 12 minutes to compute
- No derivative information
- 12 parameters to fit

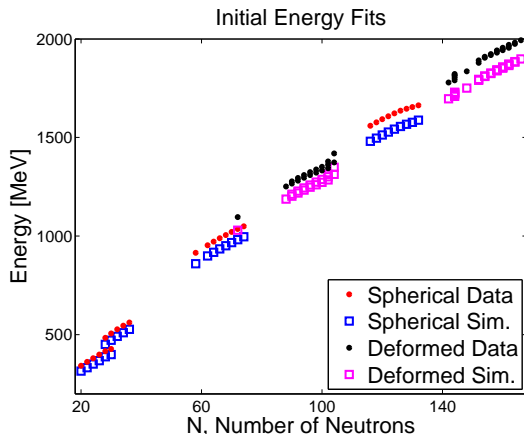
POUNDERS for hfbtho



- 72 cores on Jazz
- 12 wall-clock minutes per $f(\mathbf{x})$
- POUNDERS: acceptable \mathbf{x} in 3.2 hours
- Nelder-Mead: no acceptable \mathbf{x} in 60 hours

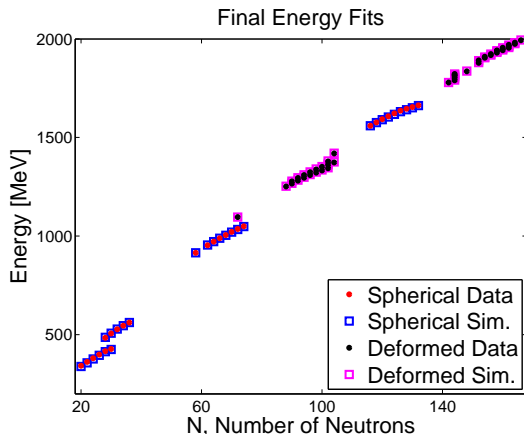
Residual error for SLy4 parameter set

Parameter	Value
ρ^{NM}	0.160
E^{NM}/A	-15.972
K^{NM}	229.901
α_{sym}^{NM}	32.004
L_{sym}^{NM}	45.962
$1/M_s^*$	1.439
$C_0^{\rho\Delta\rho}$	-76.996
$C_1^{\rho\Delta\rho}$	15.657
$V_0^{(\nu)}$	-258.200
$V_0^{(\pi)}$	-258.200
$C_0^{\rho\nabla J}$	-92.250
$C_1^{\rho\nabla J}$	-30.750



Residual error for UNEDF0 parameter set

Parameter	Value
ρ^{NM}	0.156
E^{NM}/A	-16.058
K^{NM}	207.772
α_{sym}^{NM}	36.494
L_{sym}^{NM}	81.953
$1/M_s^*$	0.934
$C_0^{\rho\Delta\rho}$	-66.281
$C_1^{\rho\Delta\rho}$	36.706
$V_0^{(\nu)}$	-188.416
$V_0^{(\pi)}$	-230.431
$C_0^{\rho\nabla J}$	-83.320
$C_1^{\rho\nabla J}$	3.278



Toolkit for Advanced Optimization (TAO)

- You can download TAO from the webpage
<http://www.mcs.anl.gov/tao>
- The documentation online includes installation instructions, a user's manual and a man page for every TAO function.
- The download includes then manual and several examples for using TAO in C and Fortran.
- POUNDERS is in the beta release of TAO 2.0, you can download directly from
<http://www.mcs.anl.gov/tao/tao-2.0-beta2.tar.gz>
- This beta version works with PETSc development version
- If you have any questions, please contact us at
tao-comments@mcs.anl.gov

How to optimize your objective function using TAO (or any other optimization software for that matter)

You need to write subroutines (in C, C++, or Fortran) to

- Initialize a starting vector
- Compute your function at a given set of variables or parameters
- Compute the gradient at the same set of variables (nice to have, but not always necessary)

You will also need to write a driver to initialize MPI, PETSc, and TAO, and register your starting vector, objective and gradient routines.

TAO will perform parallel operations on PETSc matrices and vectors, but you are responsible for making the objective and gradient evaluation routines run in parallel.

TAO Optimization Applications

The objective function evaluation routine should look like:

```
int MyFandG(TaoSolver tao, Vec x,
            double *fcn, Vec g, void *userCtx){
    /* compute function value fcn and gradient g */
    return 0;
}
```

And your driver should look like:

```
PetscInitialize(&argc,&argv,0,0);
TaoInitialize(&argc,&argv,0,0);
VecCreateMPI(PETSC_COMM_WORLD,n,N,&x); VecSet(x,0);
TaoSolverCreate(PETSC_COMM_WORLD,&tao);
TaoSolverSetInitialSolution(x);
TaoSolverSetObjectiveAndGradientRoutine(tao,MyFandG,0);
TaoSolverSetFromOptions(tao);
TaoSolverSolve(tao);
```

TAO Optimization Applications

To use POUNDERS, your objective function needs to be separated into a vector of (weighted) residuals

```
int MyFunction(TaoSolver tao, Vec x,  
              Vec f, void *userCtx){  
    /* Compute vector f of residuals */  
    return 0;  
}
```

And registered with

```
TaoSolverSetSeparableObjectiveRoutine(tao,f,  
                                       MyFunction,(void*)&user);
```


Diagonalizing the Hamiltonian

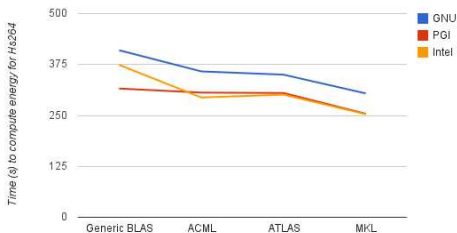
$$Hx = Ex$$

- Matrix is 3542×3542
- Biggest block is 242×242 , which takes 0.05 seconds
- 41 blocks needs to be solved 400-1000 times per nuclei per parameter set
- Blocks are too small for threaded BLAS libraries to help
- Solve entire blocks on separate threads

BLAS Sidebar

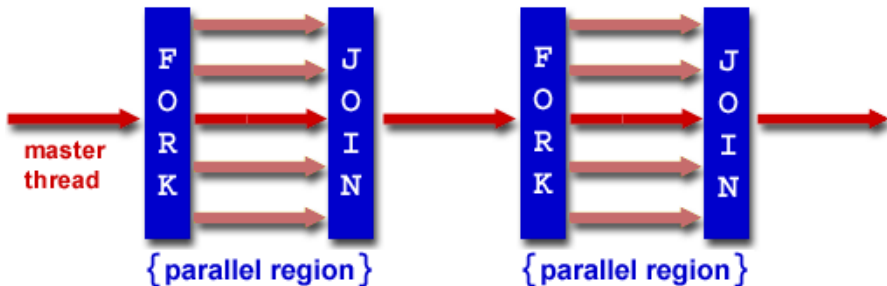
Basic Linear Algebra Subroutines

- Common interface for dense matrix and vector operations
- Generic BLAS available from netlib.org
- Use optimized BLAS libraries for your architecture



OpenMP sidebar

“The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. The OpenMP API defines a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer. “
–OpenMP.org



Advantages

- Provides a portable method for spreading work across multiple cores
- You don't need to explicitly set up initial states or pass information to threads
- Supported by most modern compilers (GNU, IBM, Oracle, Intel, PGI, Absoft, Lahey, PathScale, HP, Microsoft, Cray)

Disadvantages

- Does nothing for internode communication
- You need to identify which variables are shared among all threads and which variables are private
- If you fail to correctly identify variables, you probably won't get any syntax or run-time errors – just wrong results.
- Can be difficult to develop/maintain
- Computations will be done in an undefined order, may be impossible to exactly reproduce results

Simple OpenMP example

Computing the dot product of A and B on one thread

```
SUM = 0.0
```

```
DO I = 1, N
```

```
    SUM = SUM + (A(I) * B(I))
```

```
ENDDO
```

```
PRINT *, '    Sum = ', SUM
```

Simple OpenMP example

Computing the dot product of A and B on multiple threads

```
SUM = 0.0
!$OMP PARALLEL DO REDUCTION(+:SUM)
  DO I = 1, N
    SUM = SUM + (A(I) * B(I))
  ENDDO

PRINT *, ' Sum = ', SUM
```

Real OpenMP example

```
Do ib=1,NB
```

```
    Calculate local densities
```

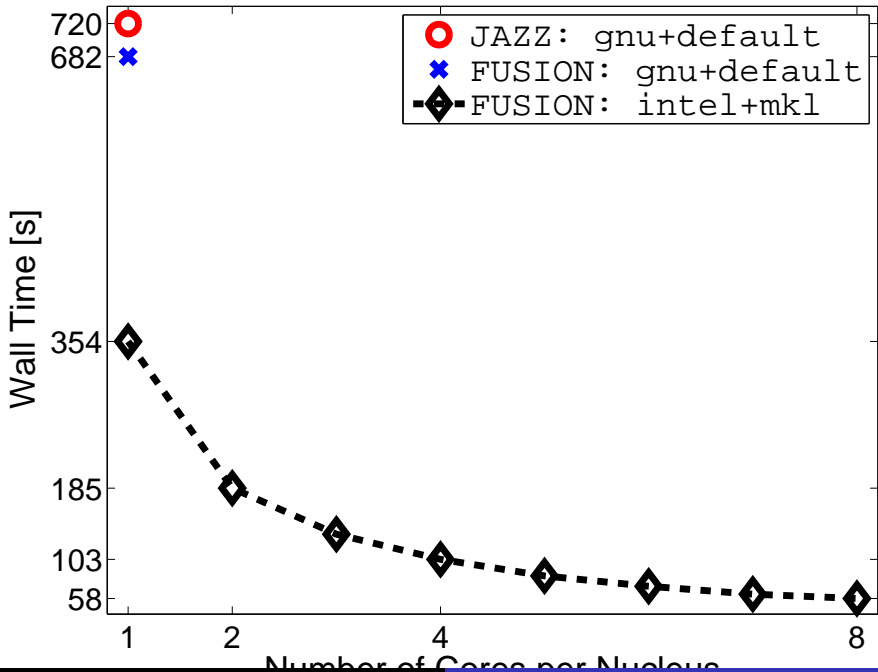
```
End Do
```

Real OpenMP example

```
!$OMP Parallel Do          &
!$OMP& DEFAULT(NONE)      &
!$OMP& SCHEDULE(DYNAMIC)  &
!$OMP& SHARED(BLOBLO,KEYBLO,NB, ID, IA, BLO123D,KA, KD,UV, NGL,NGH, XL, XH, FLI, &
!$OMP&          ZERO,FORTH,HALF, ONE, TWO,FOUR,NL, NR,NZ, NS, QH, QL, QL1, QH1, &
!$OMP&          FP1,FP2,FP3,FP4,FP5,FP6,FS1,FS2,FS3,FS4,FS5,FS6,KOP3,ngh1) &
!$OMP& PRIVATE(IT, IBIBLO, IB, ND, IM, LAPLUS, XLAP, XLAM, XLAP2, XLAM2, KO, K1, K2, IMEN, &
!$OMP&          J, JJ, K, IL, V2, V4, IH, IHIL, U, U2, Y, Y2, XLAMY, XLAMY2, XLAPY, XLAPY2, &
!$OMP&          XLAMPY, OMPFIU, OMPFIUZ, OMPFIUR, OMPFIUD2N, OMPPFIU, OMPFID, OMPFIDZ, &
!$OMP&          OMPFIDR, OMPFIDD2N, OMPPFID, PIU, PIUZ, PIUR, PIUD2, &
!$OMP&          PID, PIDZ, PIDR, PIDD2, JN, I, JA, NLA, NRA, NZA, NSA, SML2, CNZAA, CNRAA, &
!$OMP&          QHA, QLA, QHLA, QHL1A, QH1LA, FI1Z, FI1R, FI2D, FIDD, ANIK, PNIK, &
!$OMP&          TFIU, TFID, TFIUR, TFIDR, TFIUZ, TFIDZ, TPFIU, TPFID, TFIUD2, TFIDD2, &
!$OMP&          TEMP1, TEMP2, TEMP3, TEMP4, TEMP5, TEMP6, TEMP7, TEMP8, TEMP9) &
!$OMP& REDUCTION(+:OMPTAKA,OMPTRO,OMPTDJ,OMPTTAU,OMPTDRO,OMPTSRFI,OMPTSFIR, &
!$OMP&          OMPTSFIZ,OMPTSZFI)
Do ib=1,NB
```

Calculate local densities

End Do



The End

- Brought wallclock time to find reasonable solution to parameter problem from 60+ hours to less than 1
- POUNDERS algorithm is available in beta release of TAO 2.0
- OpenMP and MPI are not competing paradigms, they can work cooperatively