Towards automatic exploration of bifurcation diagrams for large-scale applications

Parallelizing Python code on a budget

Jonas Thies[*], Rebekka-Sarah Hennig[*] and Michiel Wouters[†]

[*] Institute of Simulation and Software Technology
German Aerospace Center (DLR)

[†] University of Antwerp, Belgium

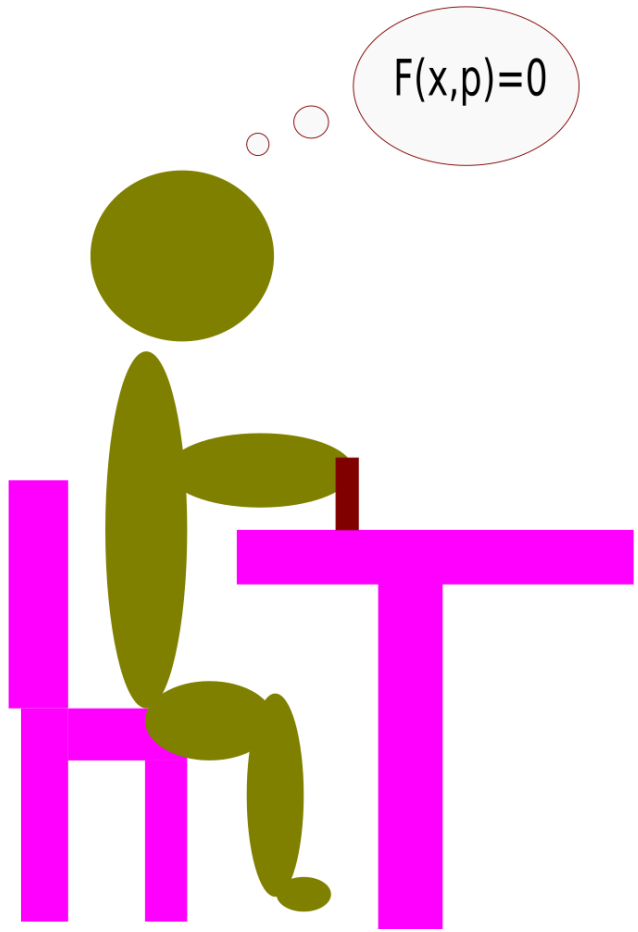
SPPEXA

project ESSEX

DLR

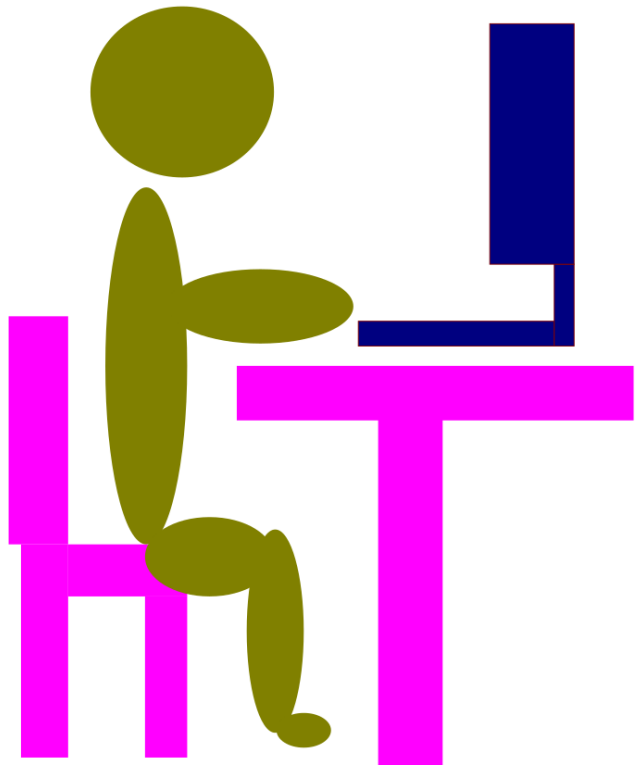
Motivation

- develop method on paper



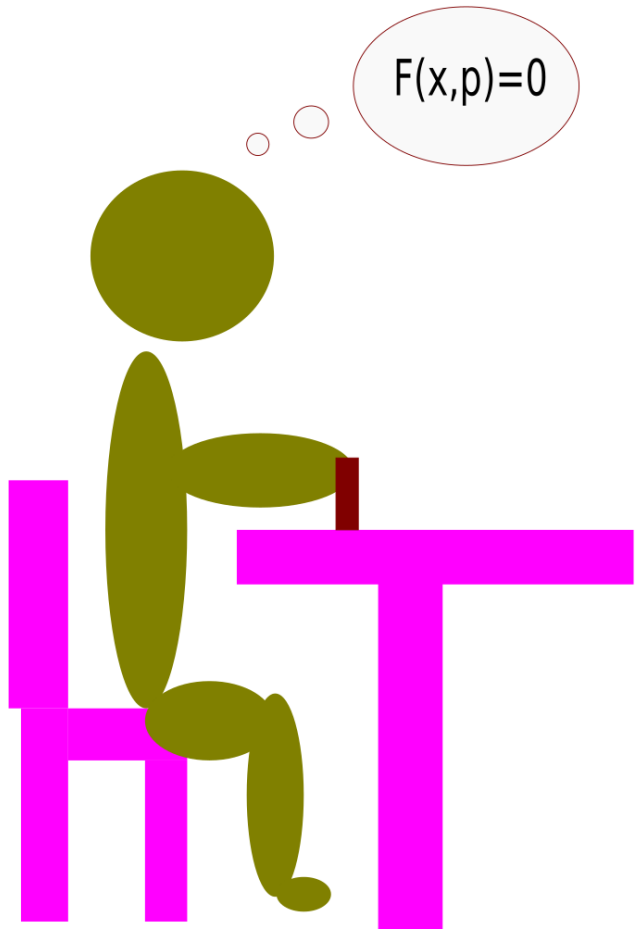
Motivation

- develop method on paper
- implement for numerical evidence



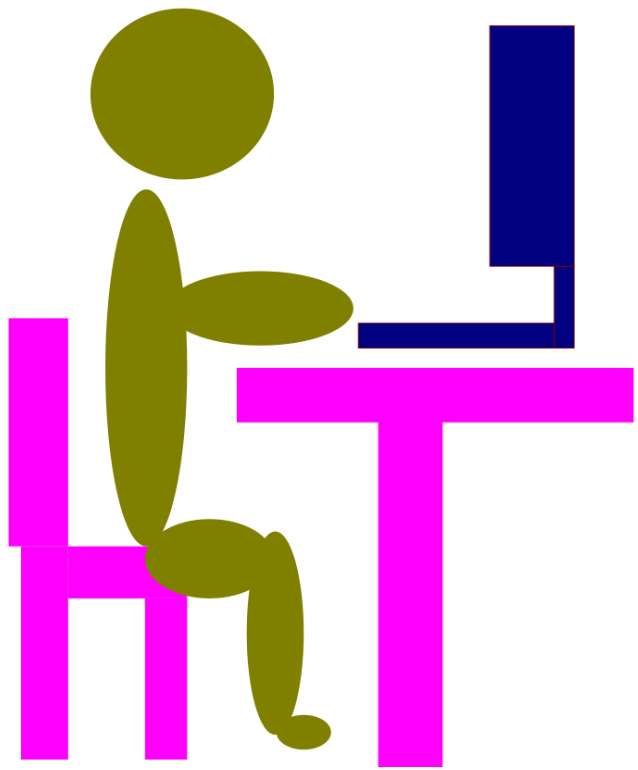
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence



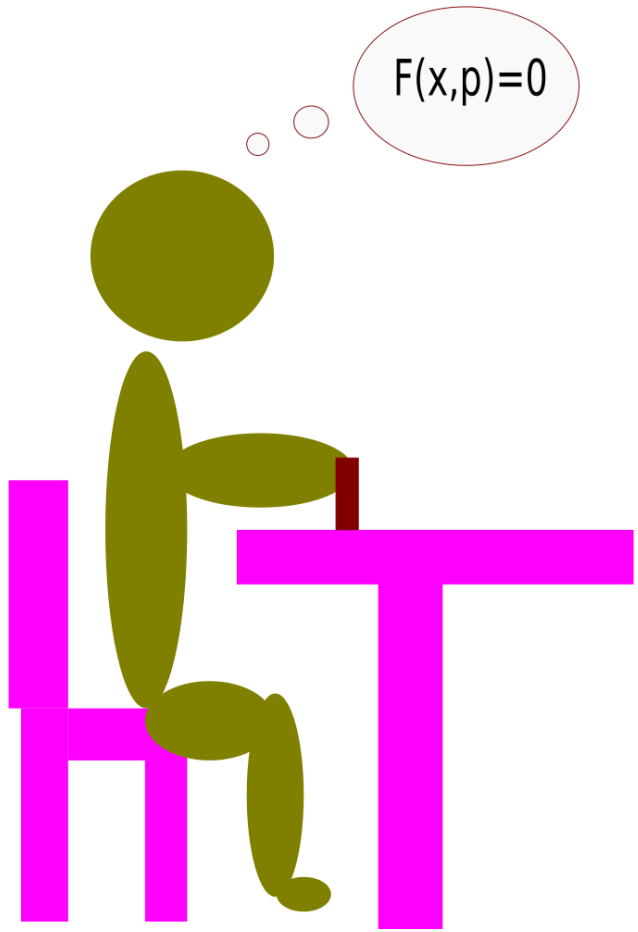
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence



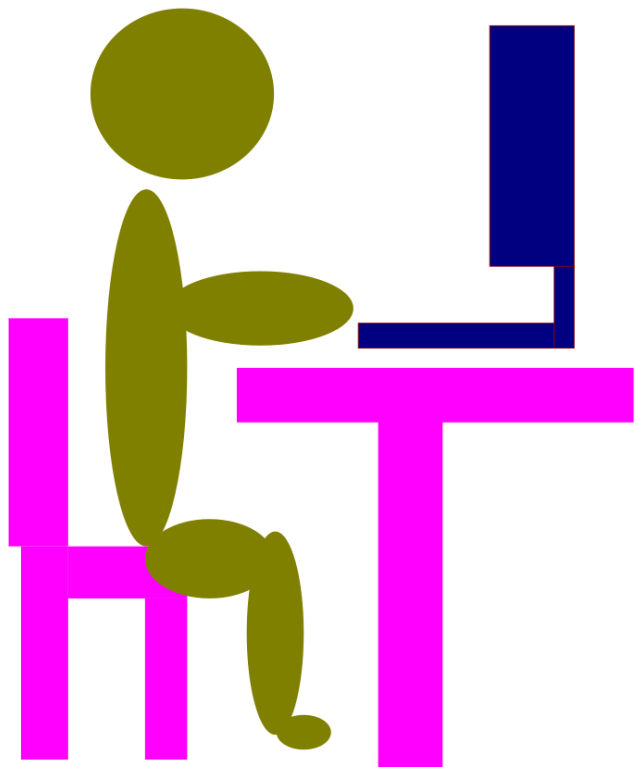
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence



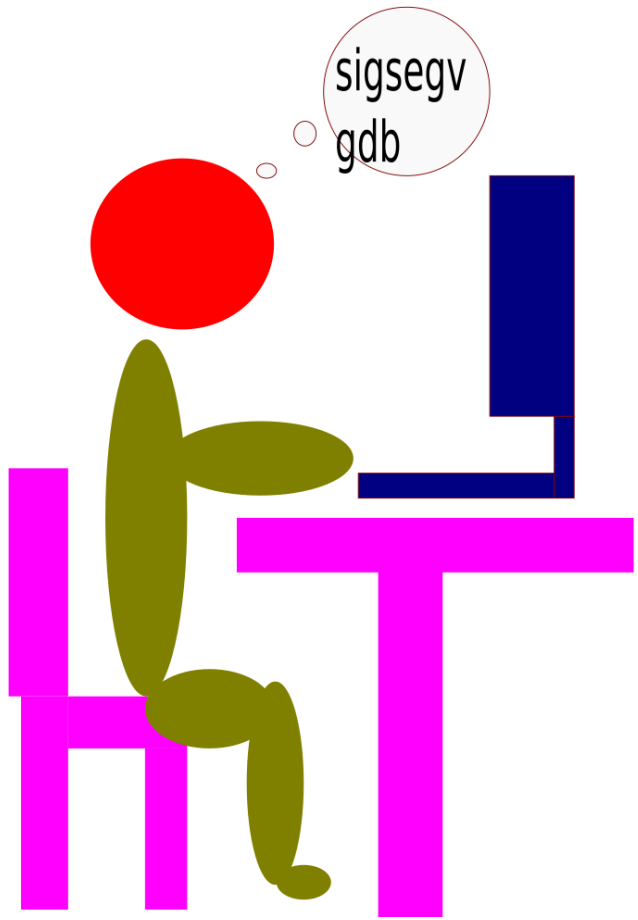
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence



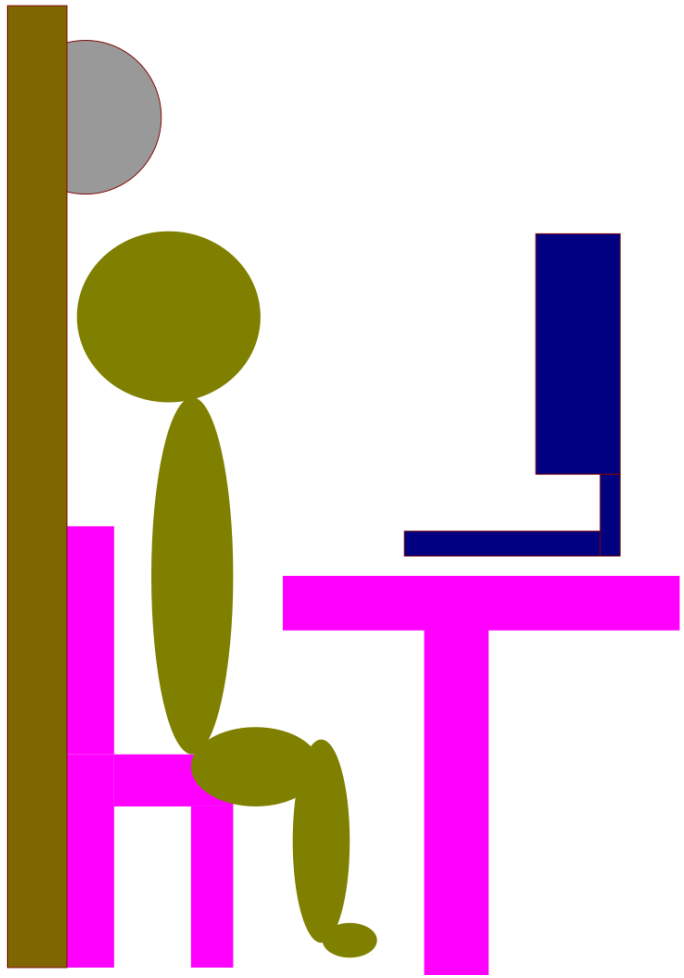
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence
- can be frustrating...



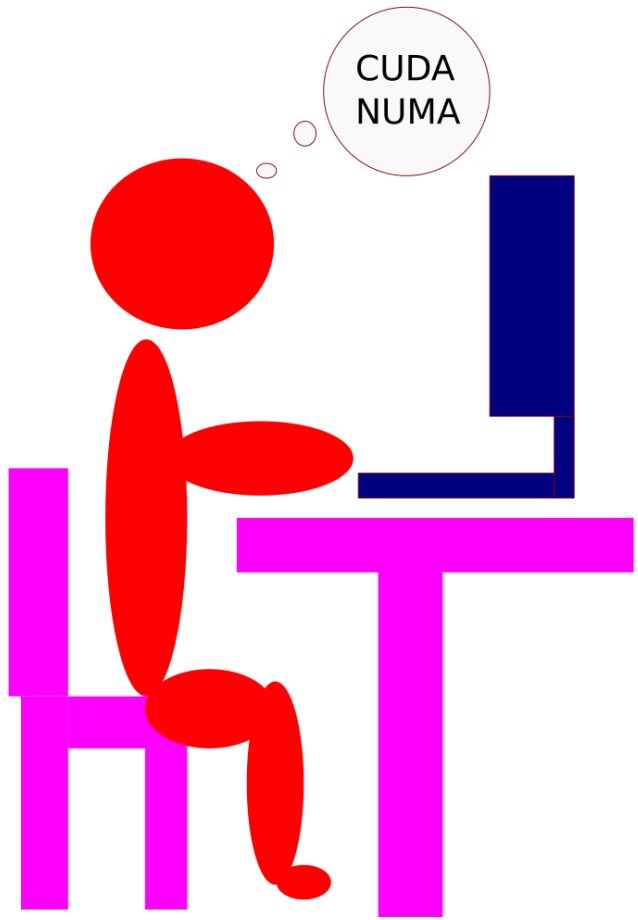
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence
- can be frustrating...
- 3 months beforee the end of your defense, your supervisor asks you to demonstrate scalability on a heterogenous supercomputer.



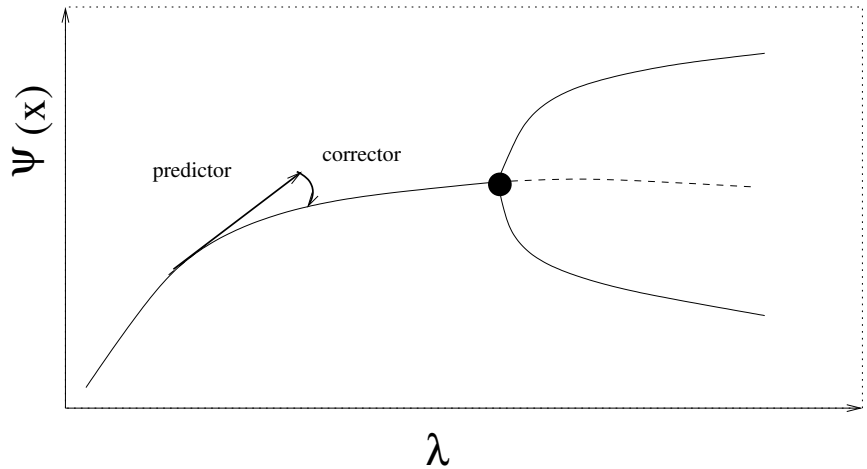
Motivation

- develop method on paper
- implement for numerical evidence
- iterate until convergence
- can be frustrating...
- 3 months beforee the end of your defense, your supervisor asks you to demonstrate scalability on a heterogenous supercomputer.



Introduction

- We seek steady states of some discretized PDE
- $\Longrightarrow$ nonlinear system $F(x, \lambda) = 0, F : \mathbb{R}^N \to \mathbb{R}^N$
- Jacobian $J(x, \lambda) = \frac{\partial F_i(x,\lambda)}{\partial x_j}$
  - large
  - sparse
  - non-Hermitian
  - possibly singular



Numerical methods:

1. Newton-Raphson
2. Krylov (GMRES)
3. preconditioning
4. bordering/deflation
5. sparse eigensolver (e.g. JDQR)

Existing Software: LOCA

LOCA

Library **O**f **C**ontinuation **A**lgorithms

(+)

- parallel linear algebra from Epetra/Tpetra
- algorithmic building blocks from Trilinos (NOX, Belos, ...)
- pseudo-arclength continuation ('stepper')
- basic techniques for bifurcation detection and branch switching

(–)

- requires good C++ skills
- no automatic exploration techniques

$\Longrightarrow$ significant human effort

https://trilinos.org/

Existing Software: PyNCT

PyNCT **P**ython **N**umerical **C**ontinuation **T**oolkit

(+)

- written in Python
- advanced algorithms, focus on robustness
- handle symmetries
- automatic exploration

(–)

- restricted to NumPy/SciPy
- limited problem size
- re-invention of the wheel (GMRES etc.)
- no unit tests

https://pypi.org/project/PyNCT/

Another sparse solver library: PHIST

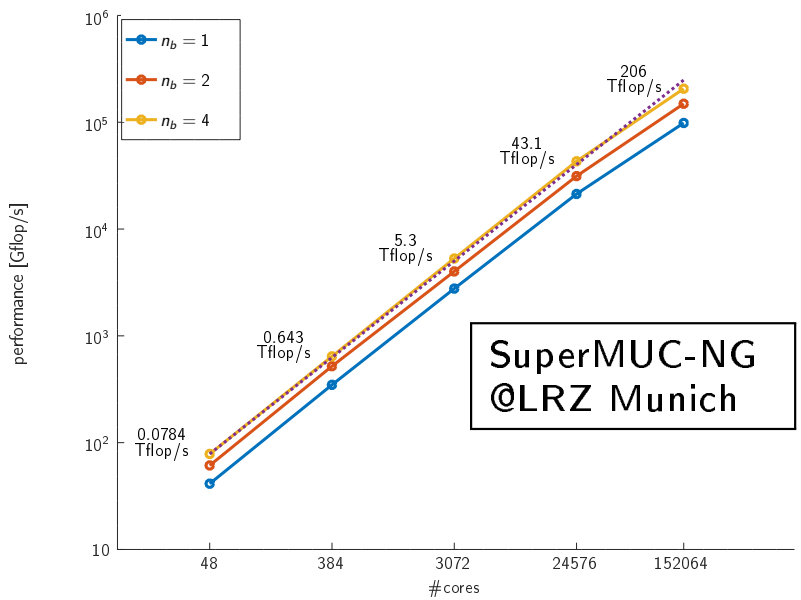
PHIST Pipelined, Hybrid-parallel Iterative Solver Toolkit

interfaces: C, C++, Fortran, Python
eigensolver: Jacobi-Davidson QR
interoperability and portability:

- extensive testing and benchmarking capabilities
- select backend at compile time:

GHOST, builtin (F'03), TRILINOS, PETSc

Extrme-scale performance



https://xsdk-info.org/

https://bitbucket.org/essex/phist/

DLR

Block Jacobi-Davidson QR

- Aim: partial QR decomposition, $\mathrm{A}Q = QR$, $R \in \mathbb{C}^{k \times k}$ upper triangular,
- $\frac{1}{2}Q^T Q - \frac{1}{2}I = 0, Q \in \mathbb{R}^{N \times k}$.

Newton's method, let $Q = \tilde{Q} + \Delta Q$

- $A\Delta Q - \Delta Q\tilde{R} = A\tilde{Q} - \tilde{Q}\tilde{R}$
- $\tilde{Q}^T \Delta Q = 0$

Block Jacobi-Davidson QR (2)

This leads to a set of *correction equations*

$$(I - \tilde{Q}\tilde{Q}^T)A(I - \tilde{Q}\tilde{Q}^T)\Delta Q - \Delta Q\tilde{R} = A\tilde{Q} - \tilde{Q}\tilde{R}$$

- Subspace acceleration: add corrections to expanding search space V
- Ritz-Galerkin: $M = V^T \mathrm{A} V$, $M = S^H R S$
- Lock converged eigenpairs $\Rightarrow$ growing projection space $\tilde{Q}$
- Solve correction eq. using (deflated) GMRES or MINRES Krylov solver
- Restart with $m_{min} > n_{eigs}$ vectors when basis becomes too large

Exploiting the PHIST interface

Example PHIST function (in C): $W \leftarrow V \cdot C$

```
void phist_Dmvec_times_sdMat(double alpha, Dconst_mvec_ptr V,
        Dconst_sdMat_ptr C, double beta,  Dmvec_ptr W, int* iflag);
```

using the auto-generated Python function:

```
    from phist_kernels import *
    V=Dmvec_ptr()
    PYST_CHK_IERR(Dmvec_create,V,map,ncols)
    [...]
    PYST_CHK_IERR(Dmvec_times_sdMat,-2,V,C, 1,W)
    [...]
    PYST_CHK_IERR(Dmvec_delete,V)
```

Exploiting the PHIST interface (2)

More user-friendly:

- `linalg` module with objects like `mvec` (multi-vector)
- store pointer to underlying PHIST object
- small objects (`sdMats`) stay NumPy

Note: we deliberately don't overload operators to stay in control

- which kernels are used
- when data is copied or 'viewed'

An **mvec cache** is used to avoid creating/deleting temporary vectors all the time

PyNCT with transparent linalg objects

Example: MGS orthogonalization in GMRES

original:

```
for j in range(it+1):
    alpha=ip(V[:,j],w)
    H[j,it]+=alpha
    w=w-alpha*V[:,j]
```

with `linalg`:

```
for j in range(it+1):
    alpha=ip(V.view(0,n,j,j+1),w)
    H[j,it]+=alpha
    Vj=V.view(0,n,j,j+1)
    w.axpby(-alpha,Vj,1)
```

Example: a Turing system

Barrio-Varea-Aragon-Maini (BVAM) model:

$$\begin{aligned}\frac{\partial u}{\partial t} &= D\delta\nabla^2 u + \alpha u(1 - r_1 v^2) + v(1 - r_2 u),\\ \frac{\partial v}{\partial t} &= \delta\nabla^2 v + v(\beta + \alpha r_1 uv) + u(\gamma + r_2 v),\end{aligned}$$

- on $\Omega = [0, 1] \times [0, 1]$,
- periodic boundary conditions.

$\Rightarrow$ lots of symmetry, especially in 3D

Preconditioning by standard AMG

Example: a Turing system

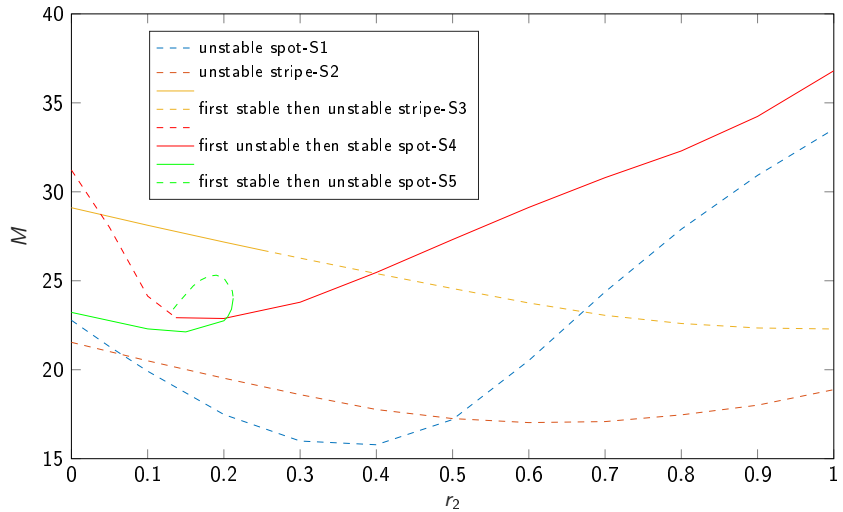
Barrio-Varea-Aragon-Maini (BVAM) model:

$$\frac{\partial u}{\partial t} = D\delta\nabla^2 u + \alpha u(1 - r_1 v^2) + v(1 - r_2 u),$$
$$\frac{\partial v}{\partial t} = \delta\nabla^2 v + v(\beta + \alpha r_1 uv) + u(\gamma + r_2 v),$$

- on $\Omega = [0, 1] \times [0, 1]$,
- periodic boundary conditions.

$\Rightarrow$ lots of symmetry, especially in 3D

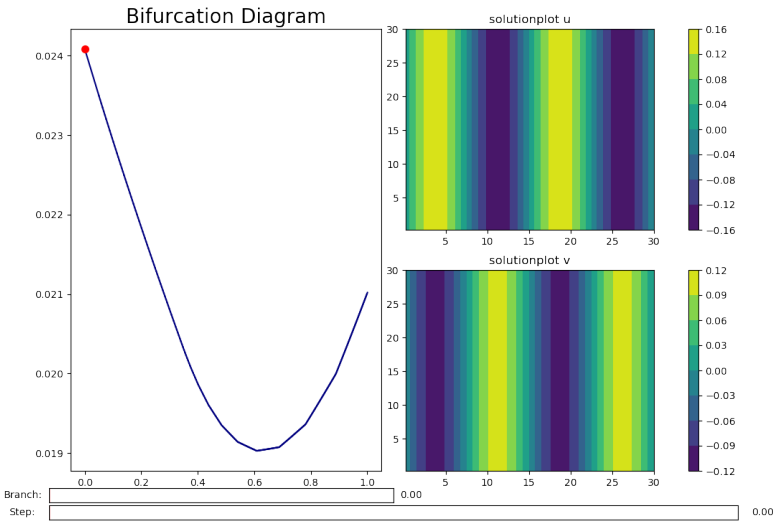
Preconditioning by standard AMG



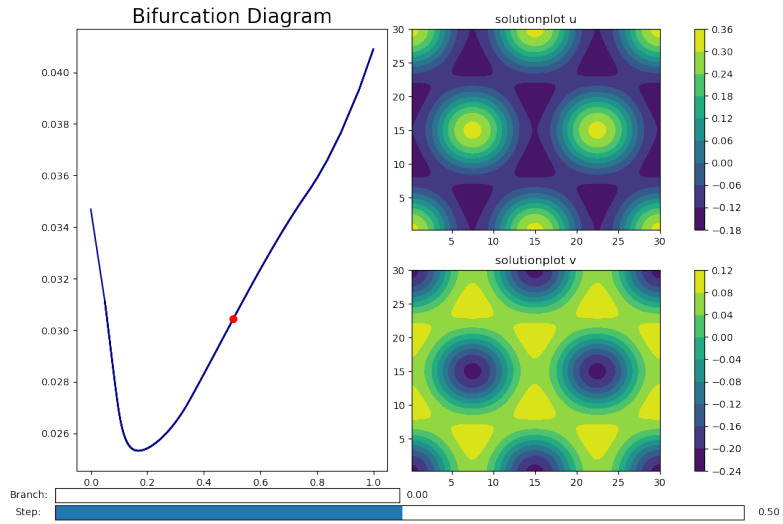
Typical bifurcation diagram

Song, Wubs, Thies & Baars: *Numerical Bifurcation Analysis of a 3D Turing-type Reaction-Diffusion Model*. CNSNS 2018

Results for Turing system



Branch 2, $n_x = 128$, solution at $r_2 = 0.0$



Branch 4, $n_x = 128$, solution at $r_2 = 0.5$

Comparison: PyNCT/Epetra $\Longleftrightarrow$ LOCA/Epetra

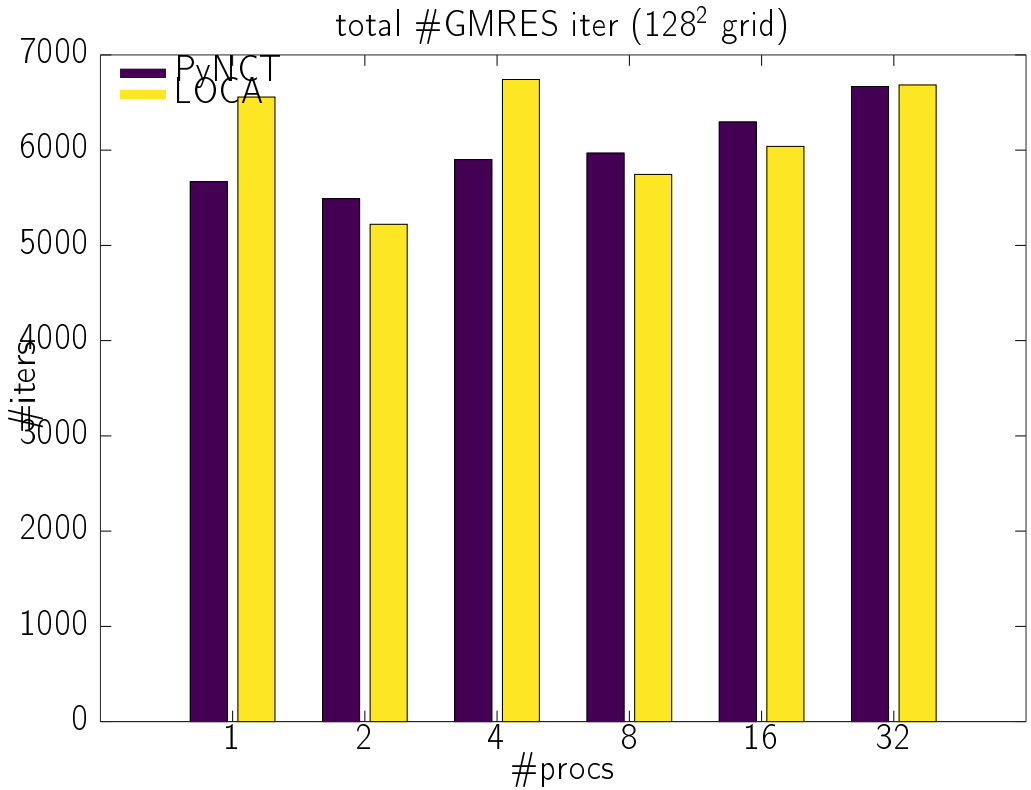
- Experiment: follow branch 4 for $r_2 = 0 \ldots 1$
- 10 continuation steps
- compare operation counts, run time, scalability...

Caveat: algorithmic details are significantly different

Comparison: PyNCT/Epetra $\Longleftrightarrow$ LOCA/Epetra

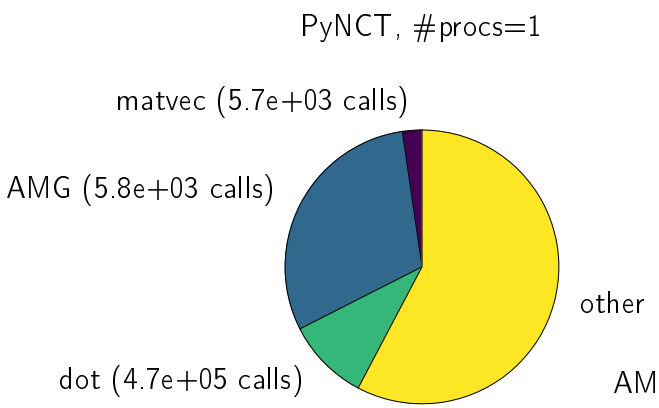
- Experiment: follow branch 4 for $r_2 = 0 \ldots 1$
- 10 continuation steps
- compare operation counts, run time, scalability...

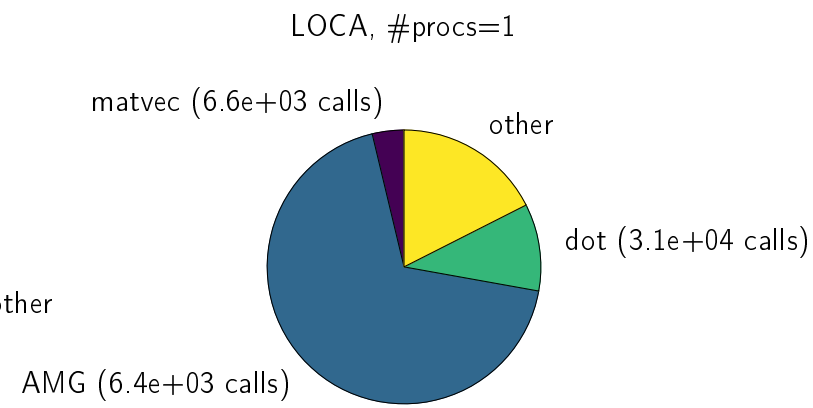
Caveat: algorithmic details are significantly different



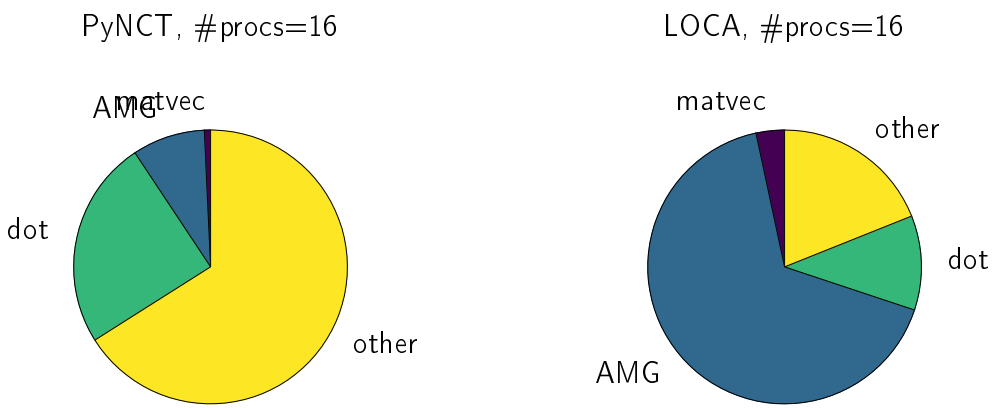
First observation: total number of GMRES iterations is similar

Performance - profiling (128^2 case)



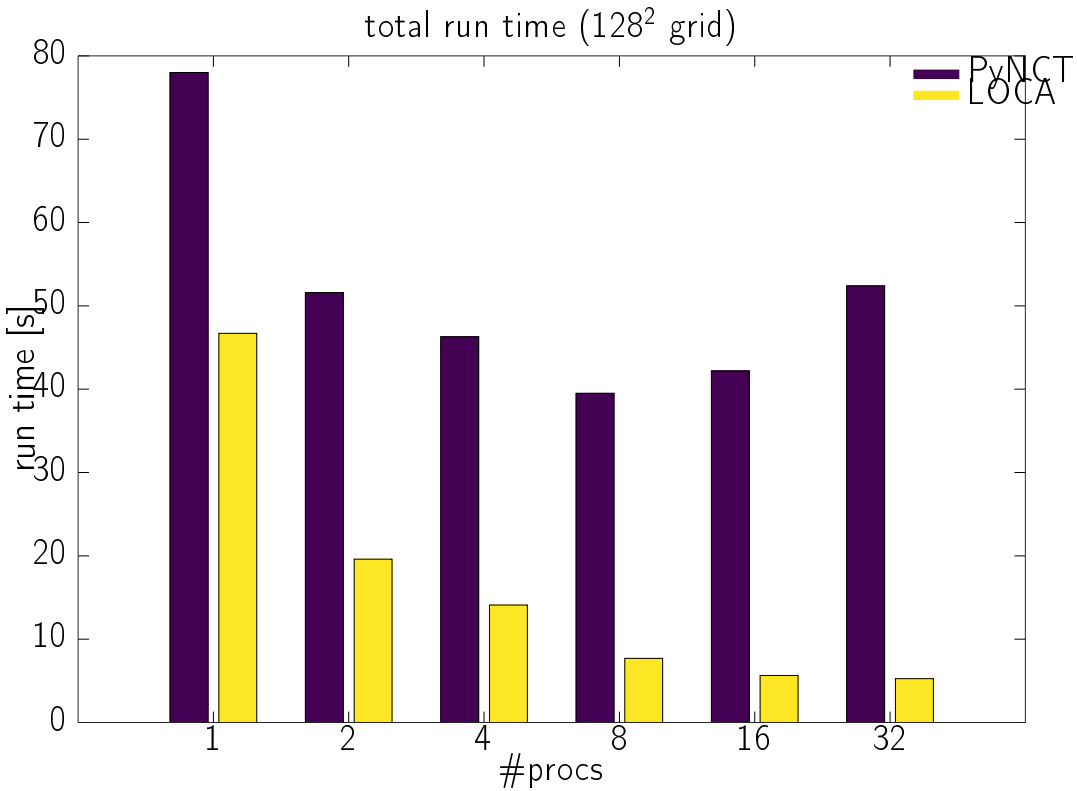


Performance - profiling (128^2 case)

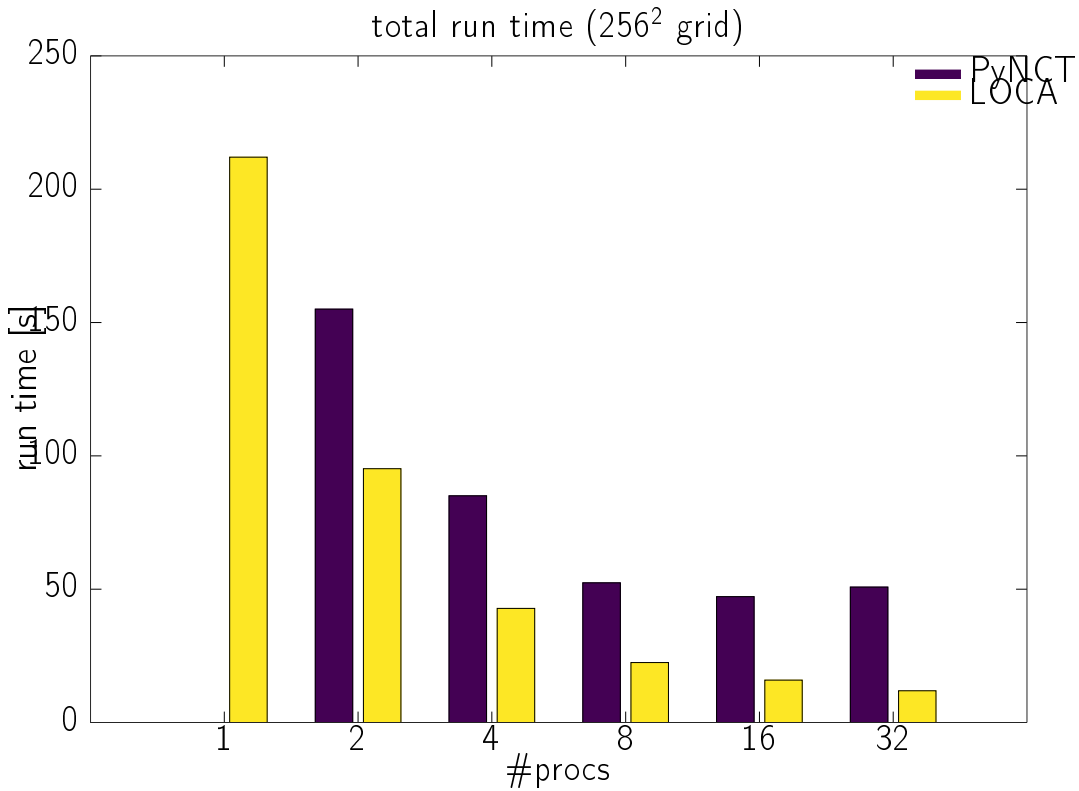


Observation: dot products and Python overhead bad for strong scaling

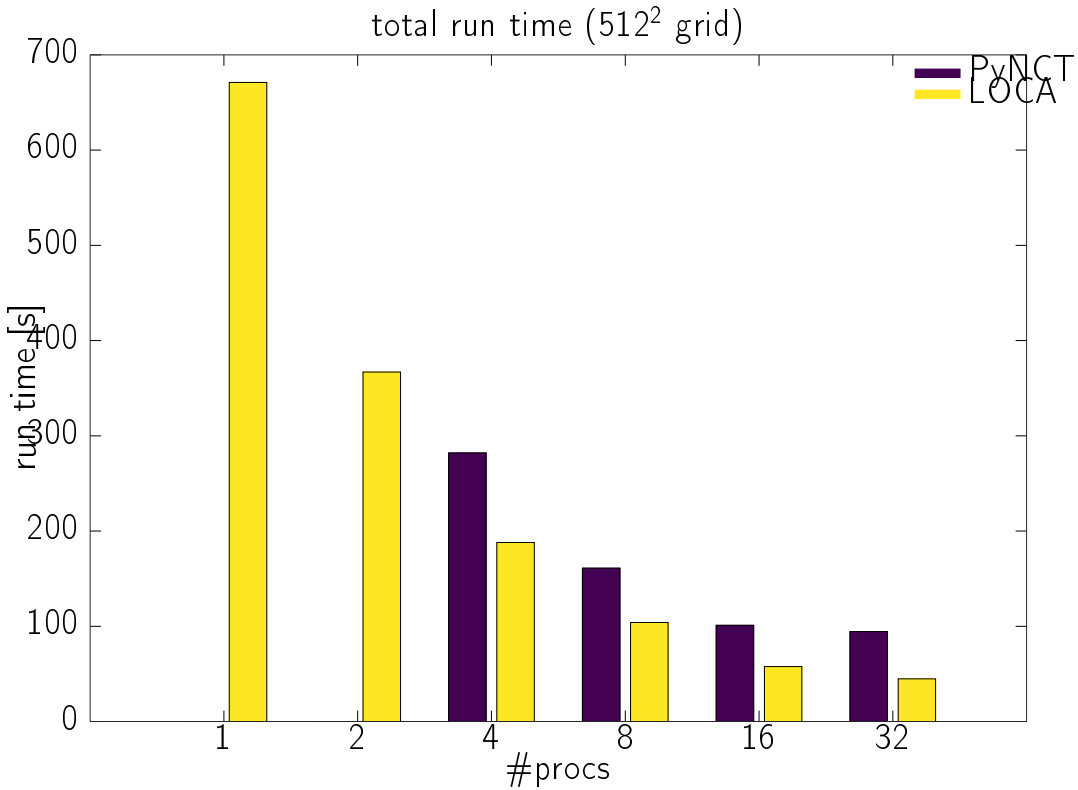
Overall performance comparison



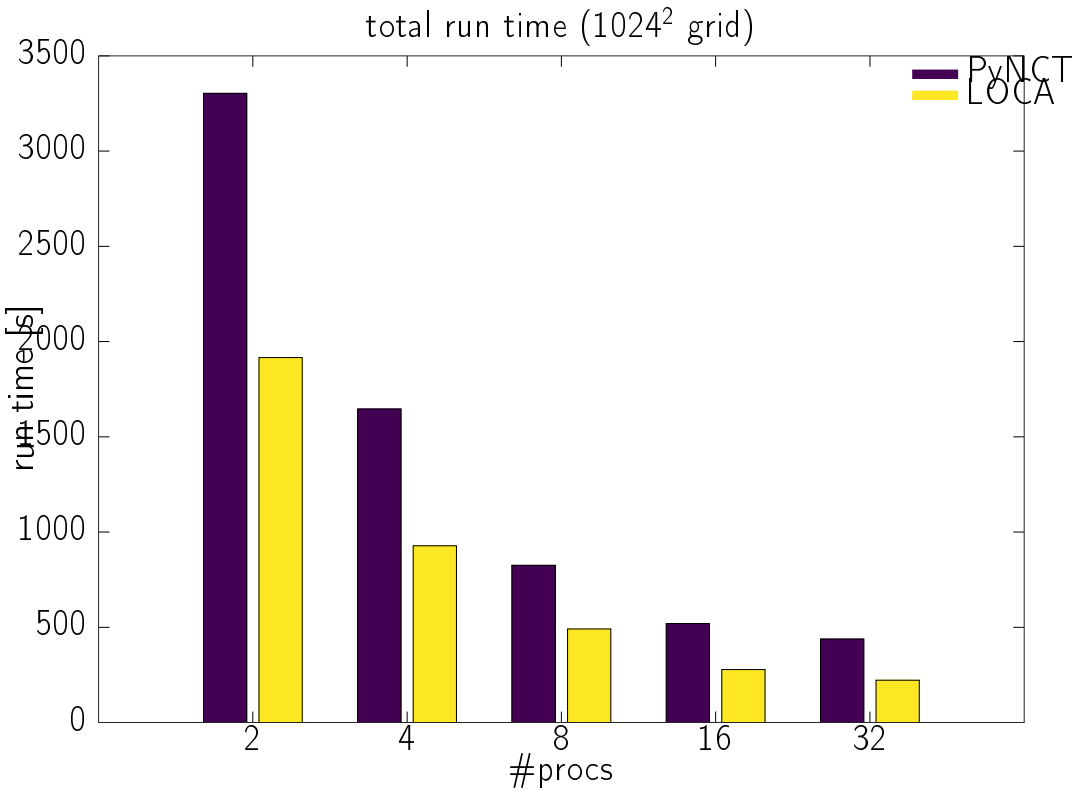
Overall performance comparison



Overall performance comparison



Overall performance comparison



Conclusion and outlook

Achievements so far

- working 'proof of concept'
- independent of 'backend'
- reasonably comfortable Python programming
- shared and distributed memory parallelism
- performance as expected

Improvements/future work

- call larger algorithmic building blocks from PHIST (GMRES, orthog, JDQR) to reduce overhead
- automatic exploration for non-Hermitian problems not working yet
- handle symmetry in 3D
- run 3D superconductor application