Workshop "HPC enabling of OpenFOAM for CFD applications"
CINECA, Casalecchio di Reno, Bologna, 26 November 2012

Parametric and Optimization study: OpenFOAM and Dakota

Ivan Spisso, i.spisso@cineca.it, CFD Numerical Analyst
SuperComputing Applications and Innovation (SCAI) Department
CINECA

Outline of the presentation

- ✓ DAKOTA in a nutshell (1)
- ✓ The loosely coupled loop of DAKOTA (1)
- ✓ Key DAKOTA Capabilities (4)
- ✓ Parallelism in Dakota (6)
- ✓ DAKOTA on PLX: job_submission, input file and loosely coupled loop (3)
- ✓ Advanced Simulation Code Interfaces: OpenFOAM (2)
- ✓ Simulation Control and quality check (2)

Dakota in a Nutshell

Design and **A**nalysis tool**K**it for **O**ptimization and **T**erascale **A**pplications includes a wide array of algorithm capabilities to support engineering transformation through advanced modeling and simulation.

Adds to simulation-based answering fundamental science and engineering questions:

• What are the crucial factors/parameters and how do they affect metrics? (*sensitivity*)

• How safe, reliable, robust, or variable is my system? (*quantification of margins and uncertainty: QMU, UQ*)

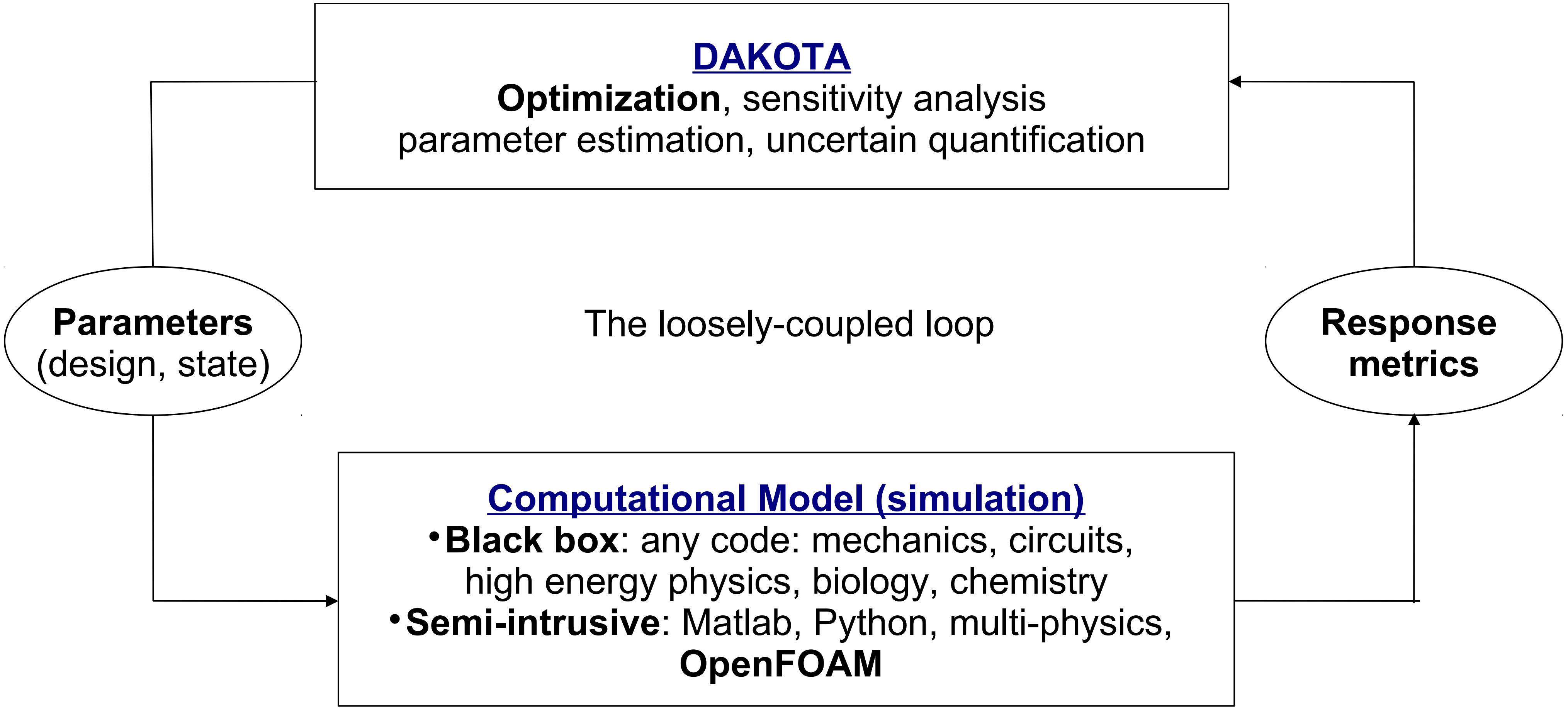
• **What is the best performing design or control? (*optimization*)**

• What models and parameters best match experimental data? (*calibration*)

• All rely on iterative analysis with a computational model for the phenomenon of interest

Automated Iterative Analysis

Automate typical "parameter variation" studies with a generic interface to simulations and advanced methods


DAKOTA
Optimization, sensitivity analysis
parameter estimation, uncertain quantification
Parameters
(design, state)
The loosely-coupled loop
Response metrics
Computational Model (simulation)
•Black box: any code: mechanics, circuits, high energy physics, biology, chemistry
•Semi-intrusive: Matlab, Python, multi-physics, OpenFOAM

Can support experimental testing: examine many accident conditions with computer models, then physically test a few worst-case conditions.

Key DAKOTA Capabilities

• Generic interface to simulations

• Time-tested and advanced algorithms to address nonsmooth, discontinuous, multimodal, expensive, mixed variable, failure-prone

• Strategies to combine methods for advanced studies or improve efficiency with surrogates (meta-models)

• Mixed deterministic / probabilistic analysis

• Supports **scalable parallel computations** on clusters **!!**

• Object-oriented code; modern software quality practices

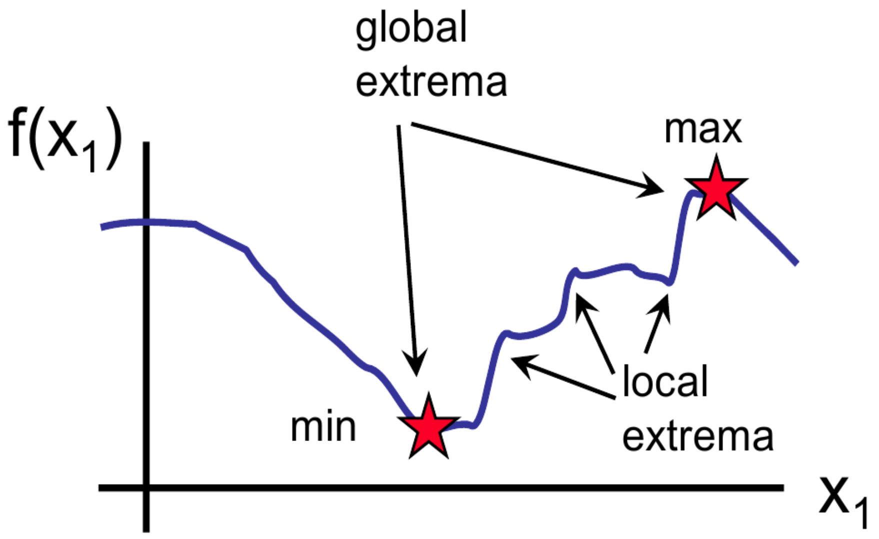
• JAGUAR 2.0, new graphical user interface in Java, based on Eclipse IDE/Workbench. Windows, Mac, Linux support.

• Additional details: http://www.cs.sandia.gov/dakota
→ Software downloads: stable releases and nightly builds (freely available worldwide via GNU LGPL)
→ ***Installed on PLX*** (*module load profile/advanced autoload dakota*) like Sandia National Lab

Optimization

• GOAL: Vary parameters to extremize objectives, while satisfying constraints to find (or tune) **the best design**, estimate best parameters, analyze worst-case surety, e.g., determine:

– delivery network that maximizes profit while minimizing environmental impact
– **case geometry that minimizes drag and weight, or maximize the pressure force, yet is sufficiently strong and safe**
– material atomic configuration of minimum energy



DAKOTA Optimization Methods

Dakota includes

- Gradient and non-gradient-based

methods.

- Several numerical package are available: commercial, developed internally to Sandia and free software from open-source community.

Gradient-based methods

(DAKOTA will compute finite difference gradients and FD/quasi-Hessians if necessary)

- DOT (various constrained)
- CONMIN (CONstrained MINinization) Library: FRCG (Fletcher-Reeves Conjugate Gradient), MFD.
- NLPQL (SQP, Sequential quadratic programming)
- NLPQL (SQP)
- OPT++ (CG, Newton)

Derivative-free methods

- COLINY (PS, APPS, SolisWets, COBYLA2, EAs, DIRECT)
- JEGA (single/multi-obj Genetic Algorithms)
- EGO (efficient global opt via Gaussian Process models)
- DIRECT (Gablonsky, Sandia developed)
- OPT++ (parallel direct search)

Calibration (least-squares)

- NL2SOL (GN + QH)
- NLSSOL (SQP)
- OPT++ (Gaussian-Newton)

Considerations when Choosing an Optimization Method

Key considerations:

- Local and global sensitivity study data; trend and smoothness
- Simulation expense
- Constraint types present
- Goal: local optimization (improvement) or global optimization (best possible)

Unconstrained or bound-constrained problems:

- Smooth and cheap: nearly any method; gradient-based methods will be fastest
- Smooth and expensive: gradient-based methods
- Nonsmooth and cheap: non-gradient methods such as pattern search (local opt), genetic algorithms (global opt), DIRECT (global opt), or surrogate-based optimization (quasi local/global opt)
- Nonsmooth and expensive: surrogate-based optimization (SBO)*

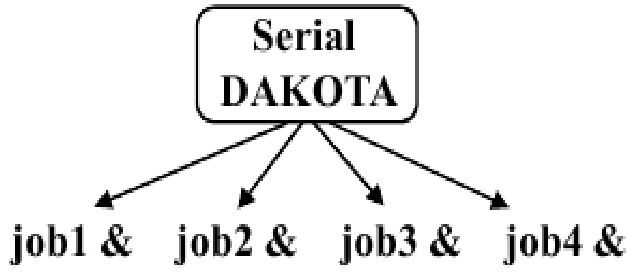
Non-linearly-constrained problems:

- Smooth and cheap: gradient-based methods
- Smooth and expensive: gradient-based methods
- Nonsmooth and cheap: non-gradient methods w/ penalty functions, SBO
- Nonsmooth and expensive: SBO

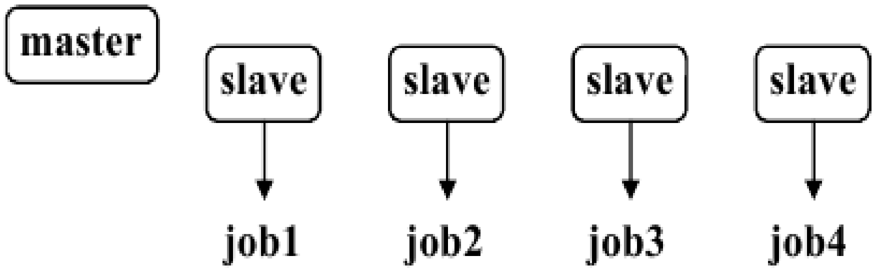
Scalable Parallelism

Nested parallel models support large-scale applications and architectures.

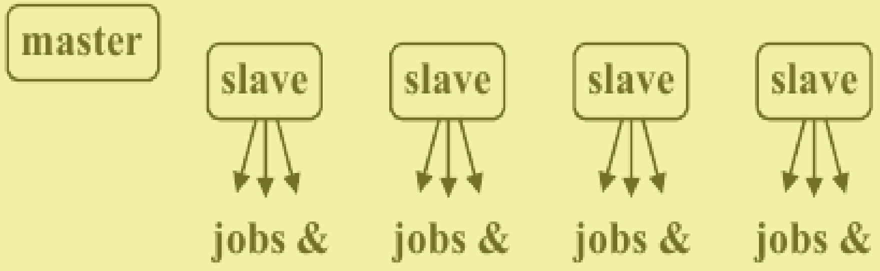
1. *SMP/multiprocessor workstations: Asynchronous (external job allocation)*



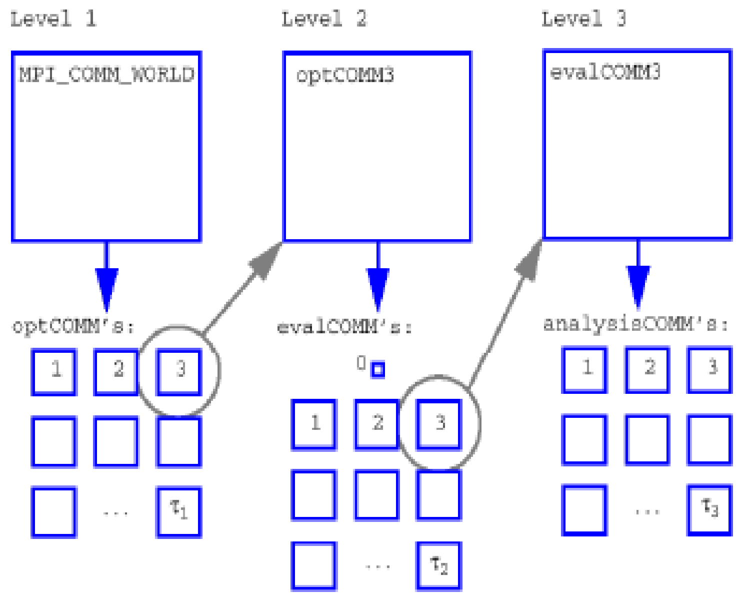
2. *Cluster of workstations: Message-passing (internal job allocation)*



3. *Cluster of SMP's: Hybrid (service/compute model)*



4. *MPP (Red Storm/ White): Internal MPI partitions (nested parallelism)*



User’s Manual: Application Parallelism Use Cases

- The parallel computing capabilities provided by DAKOTA are extensive and can be daunting at first

- Single-level parallel computing models use: asyncrhronus local, message passing, and hybrid approaches.

- This method can be combined to build multiple level of parallelism.

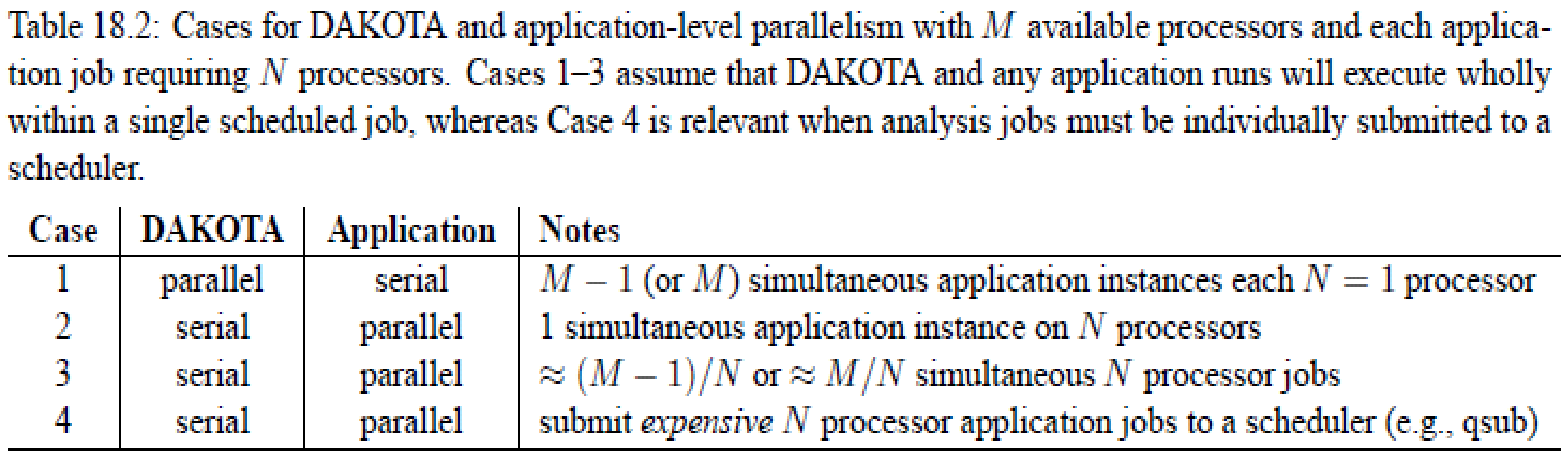


Table 18.2: Cases for DAKOTA and application-level parallelism with M available processors and each application job requiring N processors. Cases 1–3 assume that DAKOTA and any application runs will execute wholly within a single scheduled job, whereas Case 4 is relevant when analysis jobs must be individually submitted to a scheduler.

Case	DAKOTA	Application	Notes
1	parallel	serial	$M-1$ (or M) simultaneous application instances each $N=1$ processor
2	serial	parallel	1 simultaneous application instance on N processors
3	serial	parallel	$\approx (M-1)/N$ or $\approx M/N$ simultaneous N processor jobs
4	serial	parallel	submit *expensive* N processor application jobs to a scheduler (e.g., qsub)

Dakota Parallelism, Case 3: Dakota Serial, Tile N Processor Jobs

Given an allocation of M = S*N processors, schedule S simultaneous jobs

Example: 42 nodes reserved nodes (PLX 1 node =12 procs) , S=21 simultaneous jobs, N=24 processor application runs + 1 nodes to run dakota in serial

How would you achieve this?

Running dakota in *serial*

asynchronous evaluation_concurrency = 21

Launching application:

mpirun -np 24 machinefile simpleFoam -parallel

job scheduler PBS with machinefile list

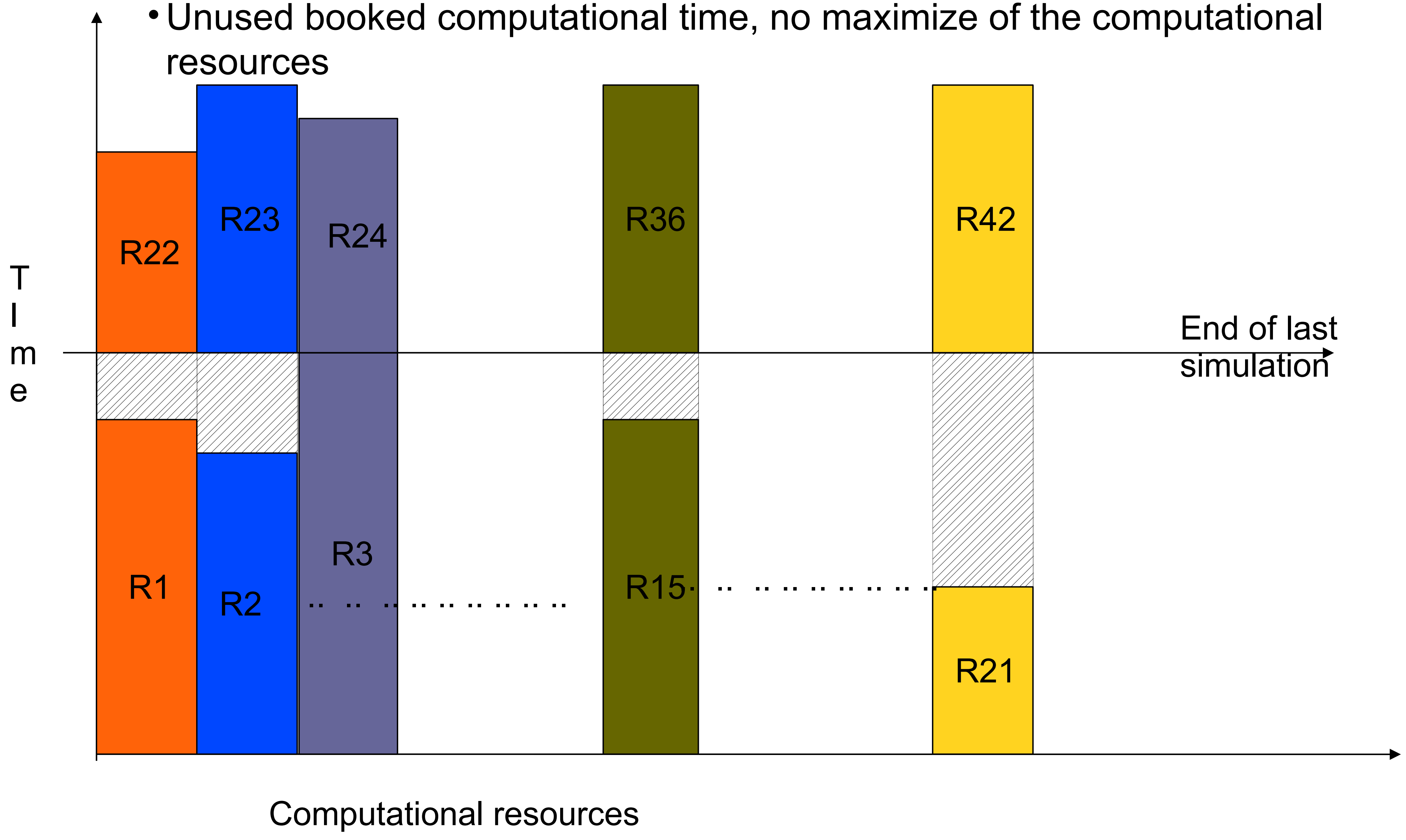
Total time: residual control on OF, wall time limit for the job

Case 3 Mechanics: Machine File Management-based

- When job starts, parse available resource list (e.g., $SLURM_NODELIST or $PBS_NODEFILE) into a single list
- Divide the resources into S files (applicNodeFile.*), each containing N resources
- For each evaluation, lock a nodefile, run the application using the nodefile, free the nodefile
- Many variations possible, including specializations where the application size N either divides the number of processors per node or is a multiple of

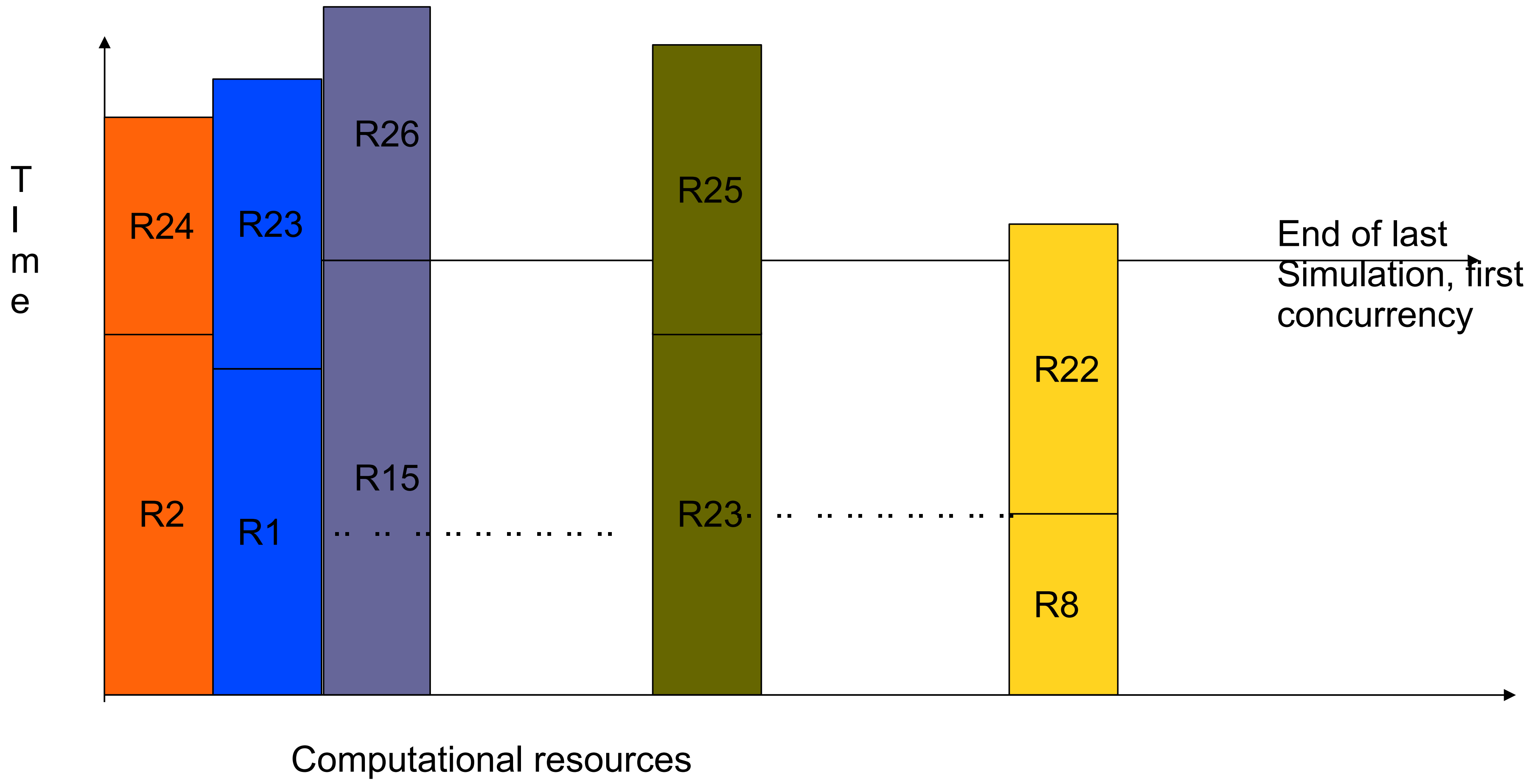
Standard Dakota Paralellism

- Standard Dakota implementation, need to wait the completion of a slot of evaluation concurrency to restart
- Unused booked computational time, no maximize of the computational resources



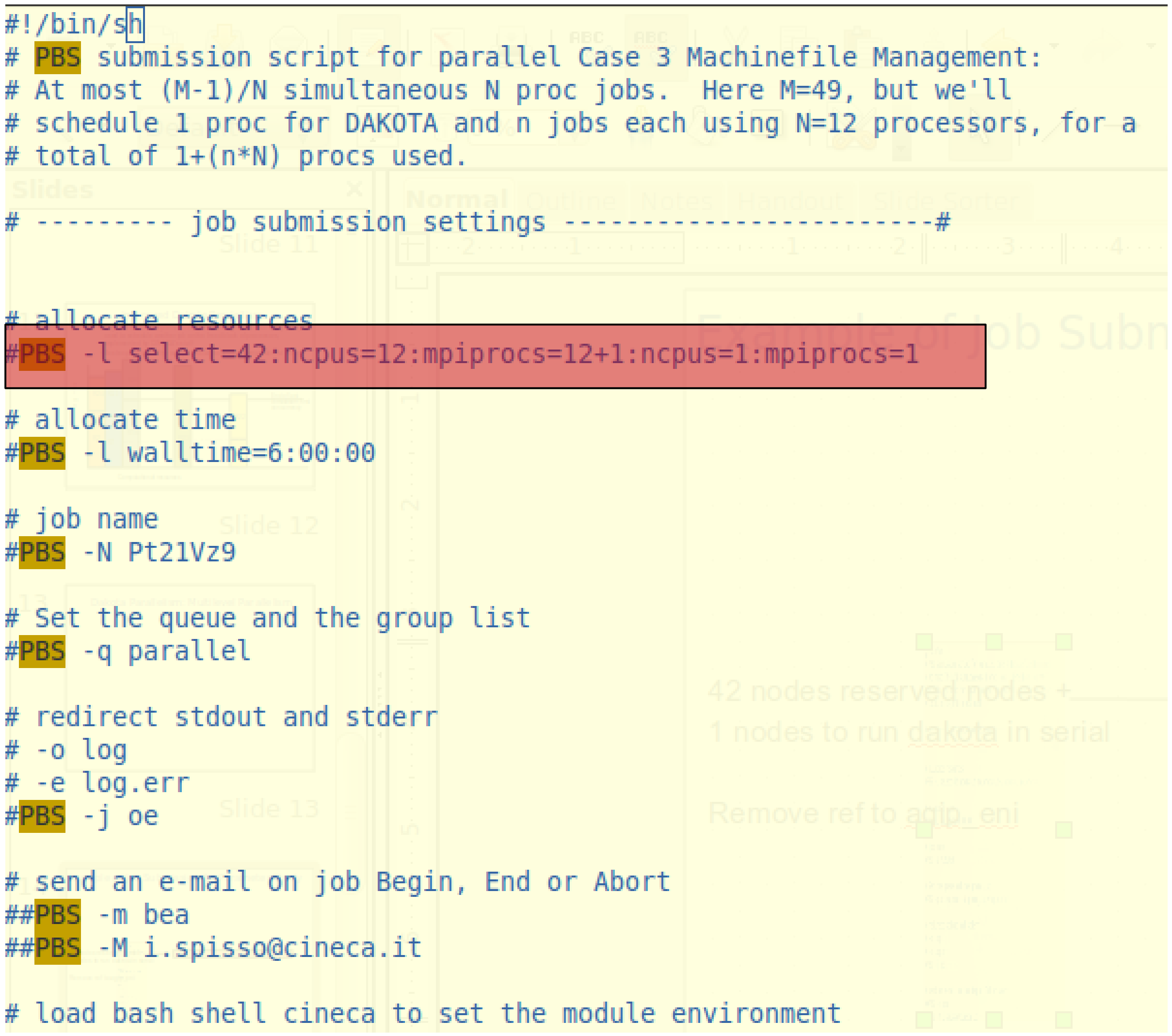
Improved Dakota Paralellism

- Improved Dakota implementation: when an application run completes, need to schedule another job on the freed block of processors, implemented by CINECA's staff
- Best exploitation of the computational resources. Computational “relay”



Example of Job Submission for Parameter Study

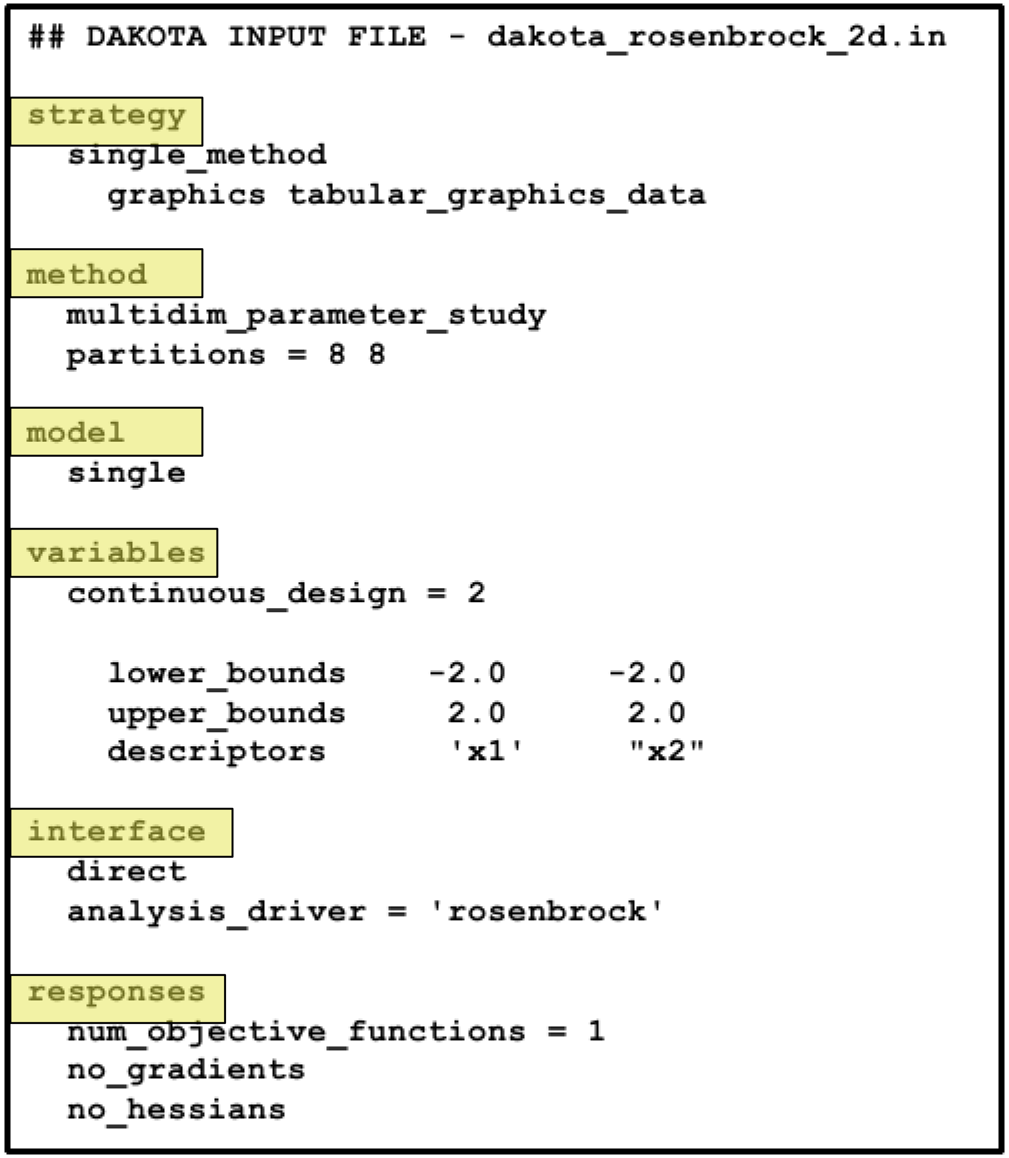
42 nodes reserved nodes +
1 nodes to run dakota in serial



```
#!/bin/sh
# PBS submission script for parallel Case 3 Machinefile Management:
# At most (M-1)/N simultaneous N proc jobs.  Here M=49, but we'll
# schedule 1 proc for DAKOTA and n jobs each using N=12 processors, for a
# total of 1+(n*N) procs used.

# --------- job submission settings -----------------------#


# allocate resources
#PBS -l select=42:ncpus=12:mpiprocs=12+1:ncpus=1:mpiprocs=1

# allocate time
#PBS -l walltime=6:00:00

# job name
#PBS -N Pt21Vz9

# Set the queue and the group list
#PBS -q parallel

# redirect stdout and stderr
# -o log
# -e log.err
#PBS -j oe

# send an e-mail on job Begin, End or Abort
##PBS -m bea
##PBS -M i.spisso@cineca.it

# load bash shell cineca to set the module environment
```

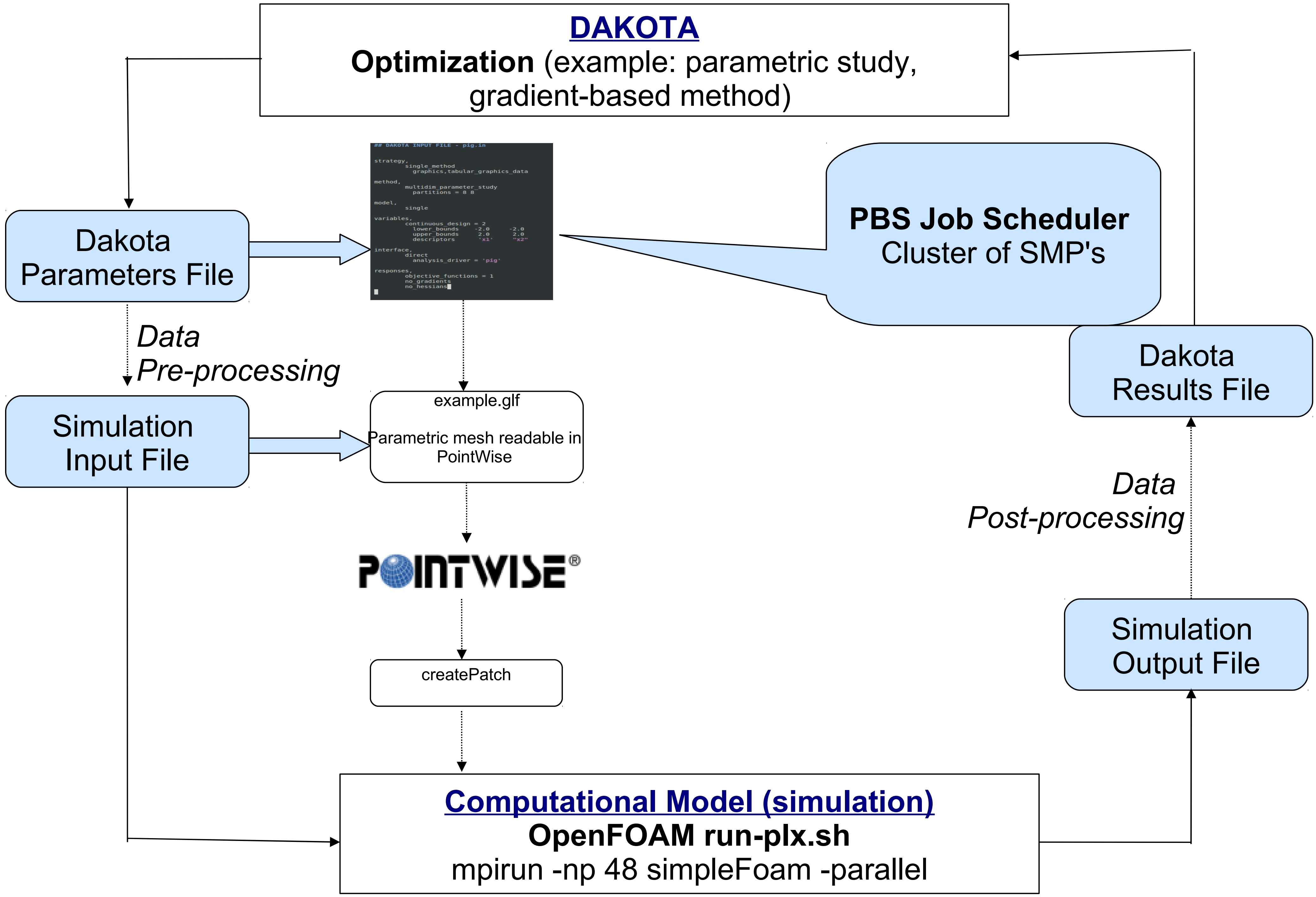
Example of Input File for Parameter Study

There are six specification blocks that may appear in DAKOTA input files.



```
## DAKOTA INPUT FILE - dakota_rosenbrock_2d.in

strategy
   single_method
     graphics tabular_graphics_data

method
   multidim_parameter_study
   partitions = 8 8

model
   single

variables
   continuous_design = 2

     lower_bounds     -2.0      -2.0
     upper_bounds      2.0       2.0
     descriptors       'x1'      "x2"

interface
   direct
   analysis_driver = 'rosenbrock'

responses
   num_objective_functions = 1
   no_gradients
   no_hessians
```

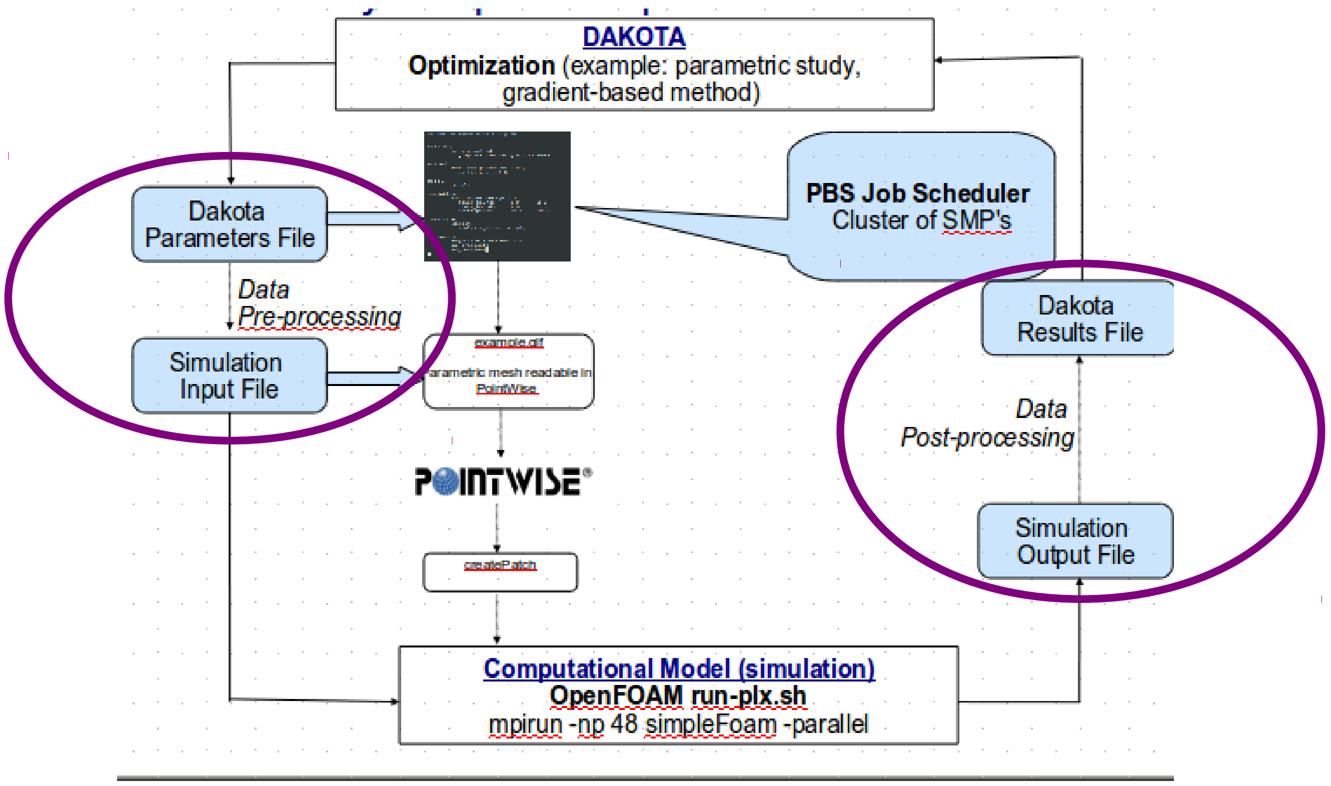
Loosely-coupled loop for DAKOTA in PLX



```
## DAKOTA INPUT FILE - pig.in

strategy,
        single_method
          graphics,tabular_graphics_data

method,
        multidim_parameter_study
          partitions = 8 8

model,
        single

variables,
        continuous_design = 2
          lower_bounds    -2.0      -2.0
          upper_bounds     2.0       2.0
          descriptors      'x1'      "x2"

interface,
        direct
          analysis_driver = 'pig'

responses,
        objective_functions = 1
        no_gradients
        no_hessians
```

Advanced Simulation Code Interfaces: OpenFOAM

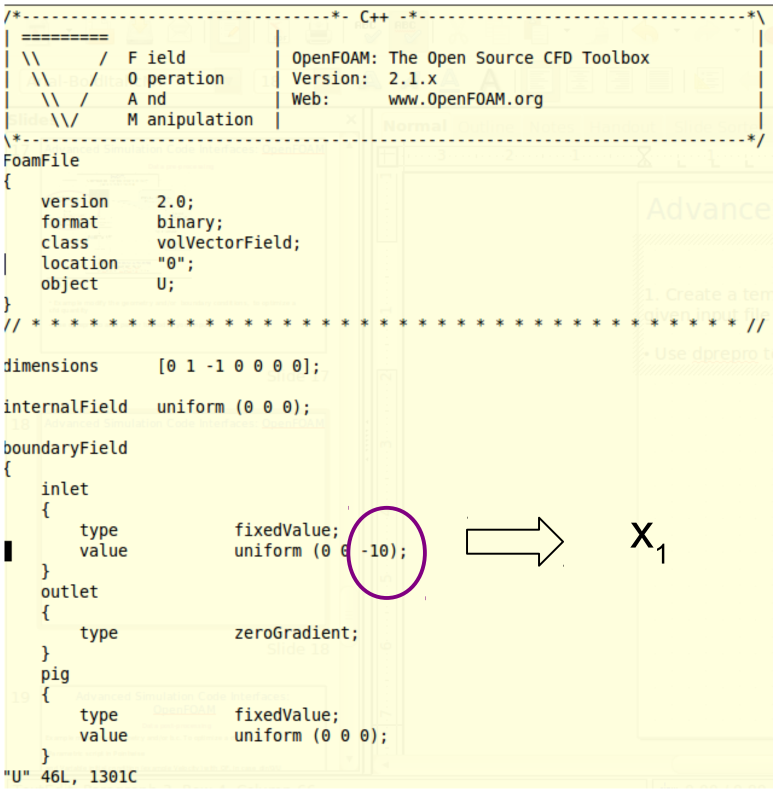
Data pre- and post-processing



- Example modify the geometry and/or boundary conditions, to optimize a cfd quantity
- Use dprepro to as a parser to modify your input

Advanced Simulation Code Interfaces: OpenFOAM

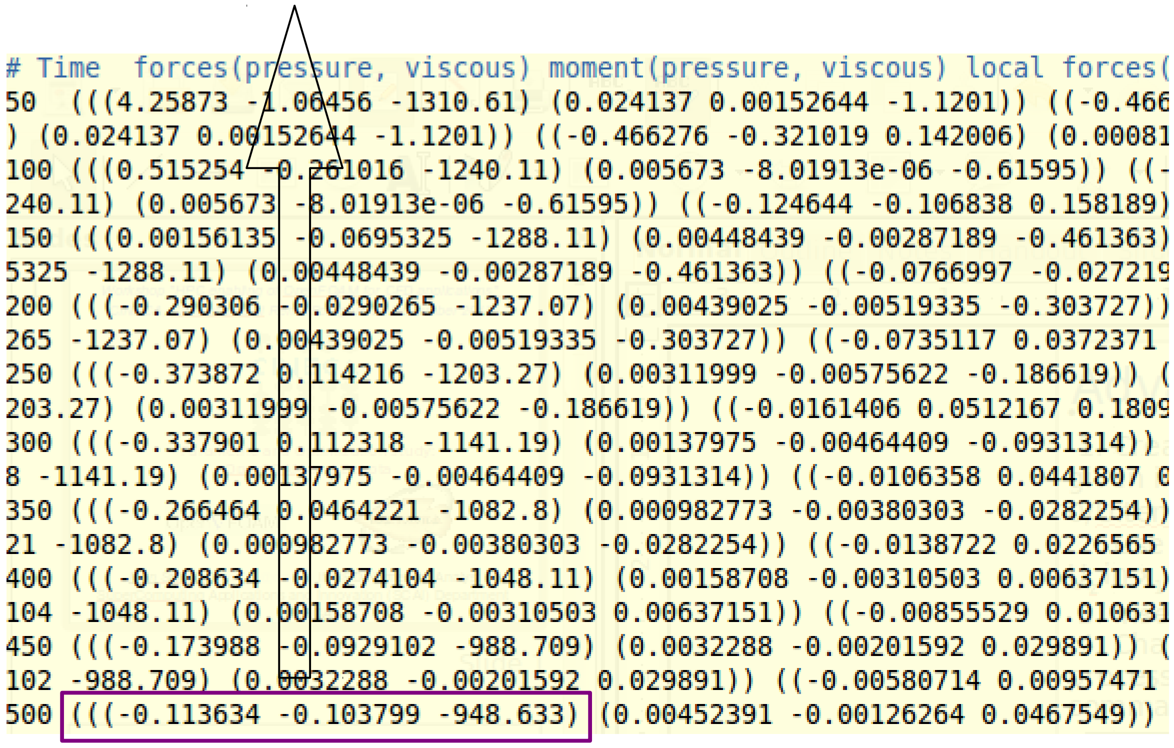
1. Create a template simulation input file by identifying the fields in the given input file that correspond to the input in DAKOTA. Example file 0/U, 0/U.template
2. Use *dprepro* as parser to reflect names of the DAKOTA parameters files U_z in x_1
3. Insert the change in the loosely-coupled loop
4. Change the post-processing section to reflect the revised extraction process. Extract your quantity from the output file, with grep command or more sophisticated extraction tools. Example, extract forces of pressure.



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 -10);
    }
    outlet
    {
        type            zeroGradient;
    }
    pig
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
"U" 46L, 1301C
```

$\Longrightarrow x_1$

output



```
# Time  forces(pressure, viscous) moment(pressure, viscous) local forces(
50  (((4.25873 -1.06456 -1310.61) (0.024137 0.00152644 -1.1201)) ((-0.466
) (0.024137 0.00152644 -1.1201)) ((-0.466276 -0.321019 0.142006) (0.00081
100 (((0.515254 -0.261016 -1240.11) (0.005673 -8.01913e-06 -0.61595)) ((-
240.11) (0.005673 -8.01913e-06 -0.61595)) ((-0.124644 -0.106838 0.158189)
150 (((0.00156135 -0.0695325 -1288.11) (0.00448439 -0.00287189 -0.461363)
5325 -1288.11) (0.00448439 -0.00287189 -0.461363)) ((-0.0766997 -0.027219
200 (((-0.290306 -0.0290265 -1237.07) (0.00439025 -0.00519335 -0.303727))
265 -1237.07) (0.00439025 -0.00519335 -0.303727)) ((-0.0735117 0.0372371
250 (((-0.373872 0.114216 -1203.27) (0.00311999 -0.00575622 -0.186619)) (
203.27) (0.00311999 -0.00575622 -0.186619)) ((-0.0161406 0.0512167 0.1809
300 (((-0.337901 0.112318 -1141.19) (0.00137975 -0.00464409 -0.0931314))
8 -1141.19) (0.00137975 -0.00464409 -0.0931314)) ((-0.0106358 0.0441807 0
350 (((-0.266464 0.0464221 -1082.8) (0.000982773 -0.00380303 -0.0282254))
21 -1082.8) (0.000982773 -0.00380303 -0.0282254)) ((-0.0138722 0.0226565
400 (((-0.208634 -0.0274104 -1048.11) (0.00158708 -0.00310503 0.00637151)
104 -1048.11) (0.00158708 -0.00310503 0.00637151)) ((-0.00855529 0.010631
450 (((-0.173988 -0.0929102 -988.709) (0.0032288 -0.00201592 0.029891)) (
102 -988.709) (0.0032288 -0.00201592 0.029891)) ((-0.00580714 0.00957471
500 (((-0.113634 -0.103799 -948.633) (0.00452391 -0.00126264 0.0467549))
```

Simulation Control and quality check

Guidelines

- ✓ Start with a parametric study to check the influence of the Design of Experiments variable

- ✓ Estimate your computational budget

- ✓ Check the single simulation, dakot.out

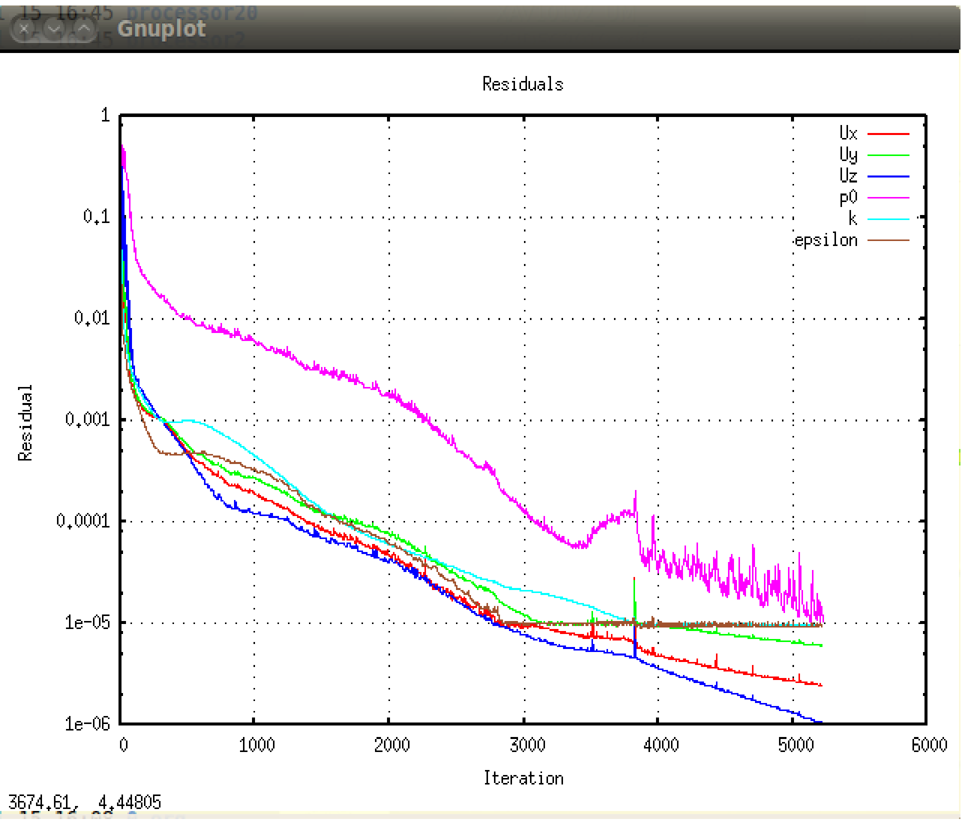
- ✓ Check the residual of OpenFOAM

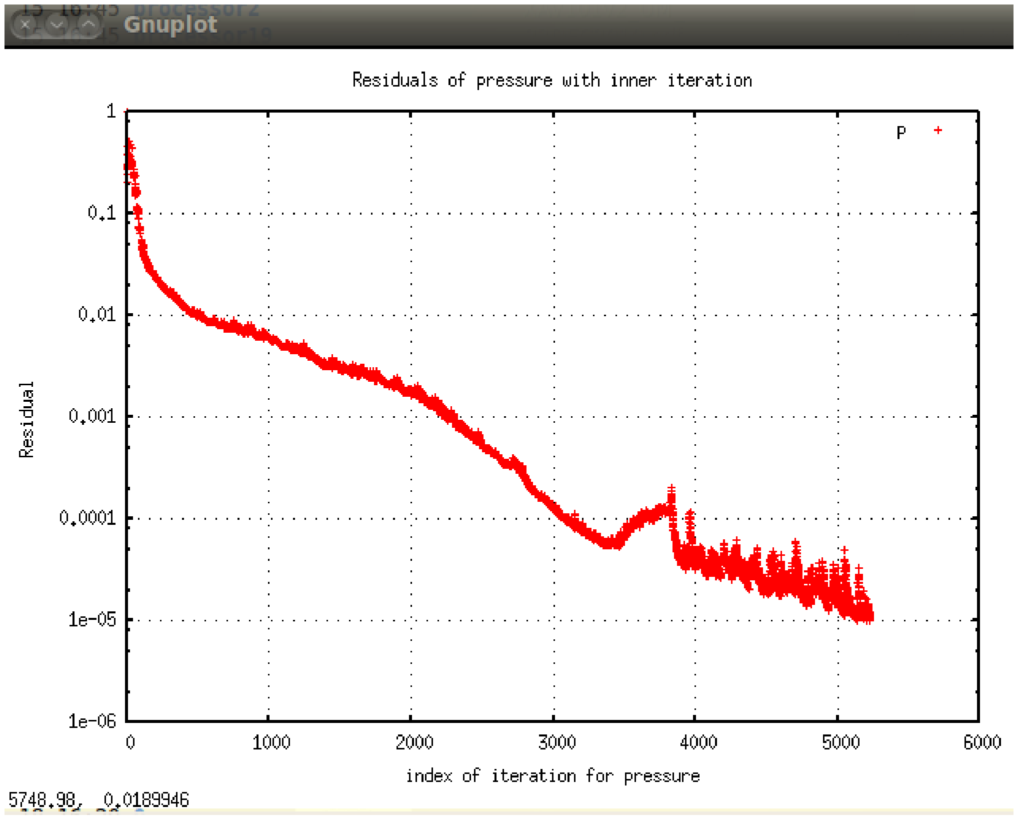
- ✓ For study involving geometrical change, a robust and good quality mesh is mandatory.

- ✓ Use visualization

Check the Quality: Residuals

- Run Time Visualization of residual implemented by CINECA staff
- Go to working dir
- Click one time: Total Residual
- Click again: Residual on pressure





Bonus Slides

Frequently **A**sked **Q**uestions

Why are you releasing DAKOTA as open source?

- To foster collaborations and streamline the licensing process. Of particular note is the fact that an export control classification of "publicly available" allows us to work effectively with universities.

How is it that Sandia can release government software as open source?

- Sandia is a government-owned, contractor-operated (GOCO) national laboratory operated for the U.S. Department of Energy (DOE) by Lockheed Martin Corporation. The authority to release open source software resides with the DOE, and DAKOTA has gone through a series of copyright assertion and classification approvals to allow release to the general public, (under LGPL). Important proponents for the open source release of Sandia software are the DOE's Accelerated Strategic Computing (ASC) Program Office and the DOE's Office of Science.

Personal note

- Reminder: Open Source and GPL does not imply zero price
- Computer time is still expensive – but cost is unavoidable
- Software support, help with running and customization is still required
- Engineers running the code are the most costly part: **better!**