HPC enabling of OpenFOAM for CFD applications
CINECA, Casalecchio di Reno, Bologna, 27 March 2014

 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

The loosely coupled loop of DAKOTA

Key DAKOTA Capabilities

Parallelism in Dakota

DAKOTA on PLX: job_submission, input file and loosely coupled loop

Advanced Simulation Code Interfaces: OpenFOAM

Simulation Control and quality check
Dakota in a Nutshell

Developed by Sandia National Lab (USA),

Design and Analysis toolKit for Optimization and Terascale Applications 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)
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

**Computational Model (simulation)**
- **Black box**: any code: mechanics, circuits, high energy physics, biology, chemistry
- **Semi-intrusive**: Matlab, Python, multi-physics, OpenFOAM
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 MINimization) 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 faster

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)**
   - Serial DAKOTA
   - job1 & job2 & job3 & job4 &

2. **Cluster of workstations: Message-passing (internal job allocation)**
   - master
   - slave
   - job1
   - slave
   - job2
   - slave
   - job3
   - slave
   - job4

3. **Cluster of SMP’s: Hybrid (service/compute model)**
   - master
   - slave
   - jobs &

4. **MPP (Red Storm/White): Internal MPI partitions (nested parallelism)**
   - Level 1: MPI_COMM_WORLD
   - Level 2: optCOMM’s:
     - 1
     - 2
     - 3
   - Level 3: evalCOMM’s:
     - analysisCOMM’s:
     - 1
     - 2
     - 3
The parallel computing capabilities provided by DAKOTA are extensive and can be daunting at first. Single-level parallel computing models use: asynchronous local, message passing, and hybrid approaches. This method can be combined to build multiple levels 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.

<table>
<thead>
<tr>
<th>Case</th>
<th>DAKOTA</th>
<th>Application</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>parallel</td>
<td>serial</td>
<td>$M - 1$ (or $M$) simultaneous application instances each $N = 1$ processor</td>
</tr>
<tr>
<td>2</td>
<td>serial</td>
<td>parallel</td>
<td>1 simultaneous application instance on $N$ processors</td>
</tr>
<tr>
<td>3</td>
<td>serial</td>
<td>parallel</td>
<td>$\approx (M - 1)/N$ or $\approx M/N$ simultaneous $N$ processor jobs</td>
</tr>
<tr>
<td>4</td>
<td>serial</td>
<td>parallel</td>
<td>submit expensive $N$ processor application jobs to a scheduler (e.g., qsub)</td>
</tr>
</tbody>
</table>
Given an allocation of \( M = S \times N \) 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 \textit{serial}

\texttt{asynchronous evaluation\_concurrency} = 21

\textit{Launching application:}

\texttt{mpirun -np 24 machinefile simpleFoam -parallel}

\textit{job scheduler PBS with machinefile list}

\textit{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 Parallelism

- 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 Parallelism

- 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

```bash
#!/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:mpi=12+1:ncpus=1:mpi=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.

```yaml
## 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 Optimization** (example: parametric study, gradient-based method)

- **Dakota Parameters File**
- **Simulation Input File**
- **Simulation Output File**
- **Simulation Results File**

**Data Pre-processing**
- Example .glf file
- Parametric mesh readable in PointWise
- createPatch

**Computational Model (Simulation)**
- OpenFOAM run-plx.sh
  - mpirun -np 48 simpleFoam -parallel

**Data Post-processing**

- **PBS Job Scheduler**
- **Cluster of SMP's**
Parametric Mesh

- Script tcl for mesh generation (*file.glf*)
- Parametric generation of the geometry (*setGeometry.tcl*)
- Automatic generation of a high-quality mesh (*pointwise -b file.glf setGeometry.tcl*)
- Algorithm to check the skewness
- Local refinement of the mesh to capture the perturbed flow in region of interest
- Automatic setting of the boundary conditions
- Direct export in OpenFOAM format
Parametric Mesh

- Generation time < 1 minute, sequential generation, synchronization algorithm to avoid concurrency to license server (single license)

- Optimization of the parameter of the solver to reach the convergence with the automatic mesh:

  - Final tolerance at convergence:
    - Eq. continuity: res < 10^-5
    - Eq. Qtà di moto: res < 10^-6
    - Niter_medio = 5-10,000
    - Niter_max = 30,000
• Example modify the geometry and/or boundary conditions, to optimize a cfd quantity

• Use dprepro to as a parser to modify your input
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` in `x₁`
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.
Analysys Driver: driver.sh

Parsing of the parameters in the template file

```
dprepro $TopDir/$1 $TopDir/setGeometry.template $TopDir/setGeometry.tcl
```

Running Pointwise on the parametric mesh on the given input file

```
pointwise -b $PBS_O_WORKDIR/porcJou.glf setGeometry.tcl
```

Decompose the mesh on the number of procs

```
decomposePar -iFRequired
```

Run OpenFoam in parallel on the nodefiles

```
mpirun -np $APPLIC_PROCS -machinefile $nodeFile simpleFoam -parallel
```

Reconstruct the solution

```
reconstructPar
```

Parsing of output on the last Iteration

```
tail -l $TopDir/forces/0/forces.dat | tr -s
```
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
Remarks and Methodology

- 1 single submission script for the parameter study
- Run time visualization of the variables of the parameter study
- Dakota demonstrates less problems to be deployed on a cluster than commercial codes not intended to run on massively parallel architectures
- Availability of tools for the integration of whatever external solver as a black box
- Feasible to set up an optimized chain for a senior HPC developer
- Development of the chain scripts and scripts to optimize the usage of resources
- Accurate Parametric mesh of the pig with low skewness and local refinement
- Specific Pointwise and Tcl programming language required
- Very time consuming
Frequently Asked Questions

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!