SAND2012-8608P

DAKOTA Advanced Topics: Interfacing and Parallelism

http://dakota.sandia.gov

Basic Steps to Using DAKOTA

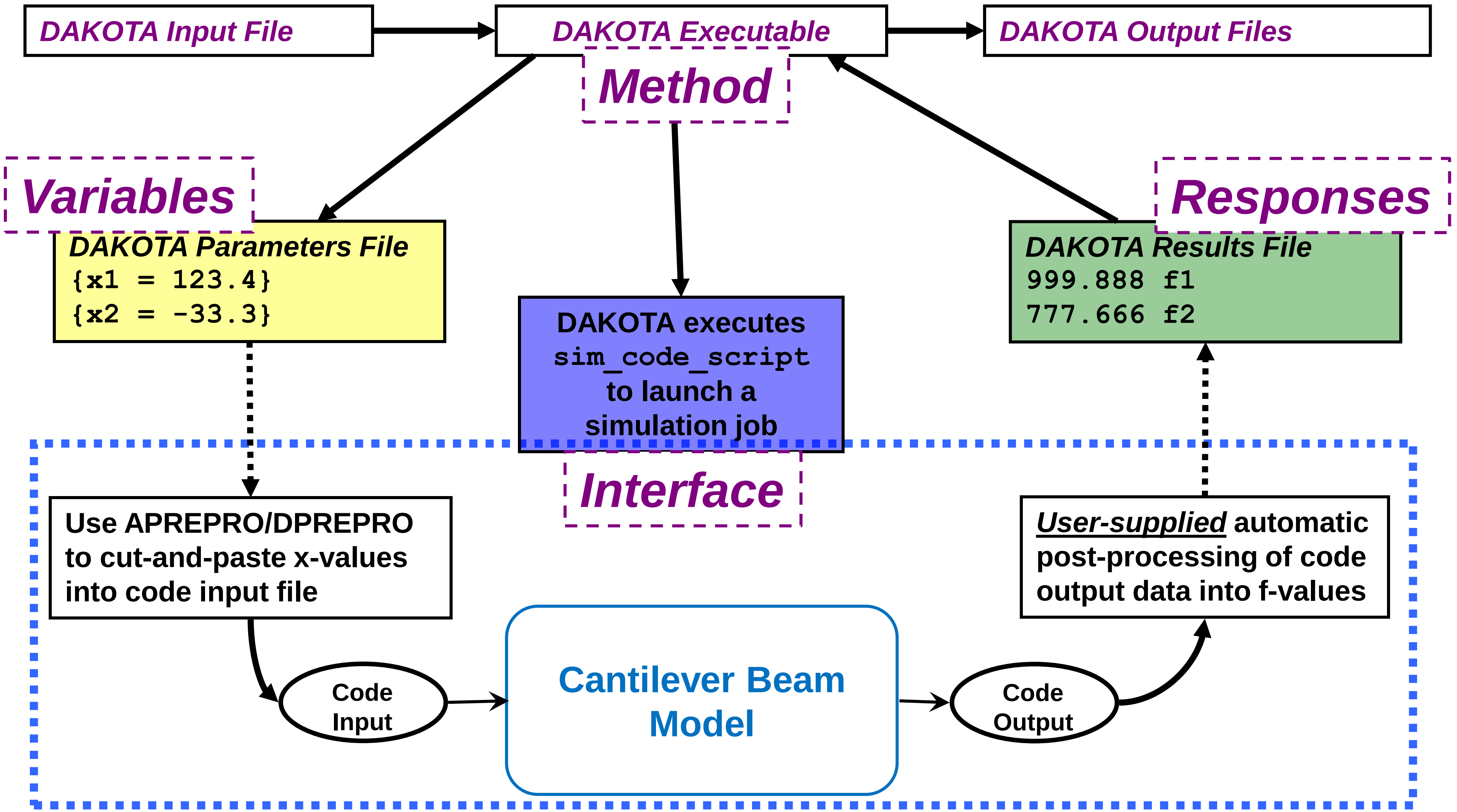
1. Define analysis goals; understand how DAKOTA helps and select a method to use
2. Access DAKOTA and understand help resources
3. Workflow: create an automated workflow so DAKOTA can communicate with your simulation (Advanced Topic)
   – Parameters to model, responses from model to DAKOTA
   – Typically requires scripting (Python, Perl, Shell, Matlab) or programming (C, C++, Java, Fortran)
   – Workflow usually crosscuts DAKOTA analysis types
4. DAKOTA input file: Jaguar GUI or text editor to configure DAKOTA to exercise the workflow to meet your goals
   – Tailor variables, methods, responses to analysis goals
5. Run DAKOTA: command-line; text input / output

Possible Directions

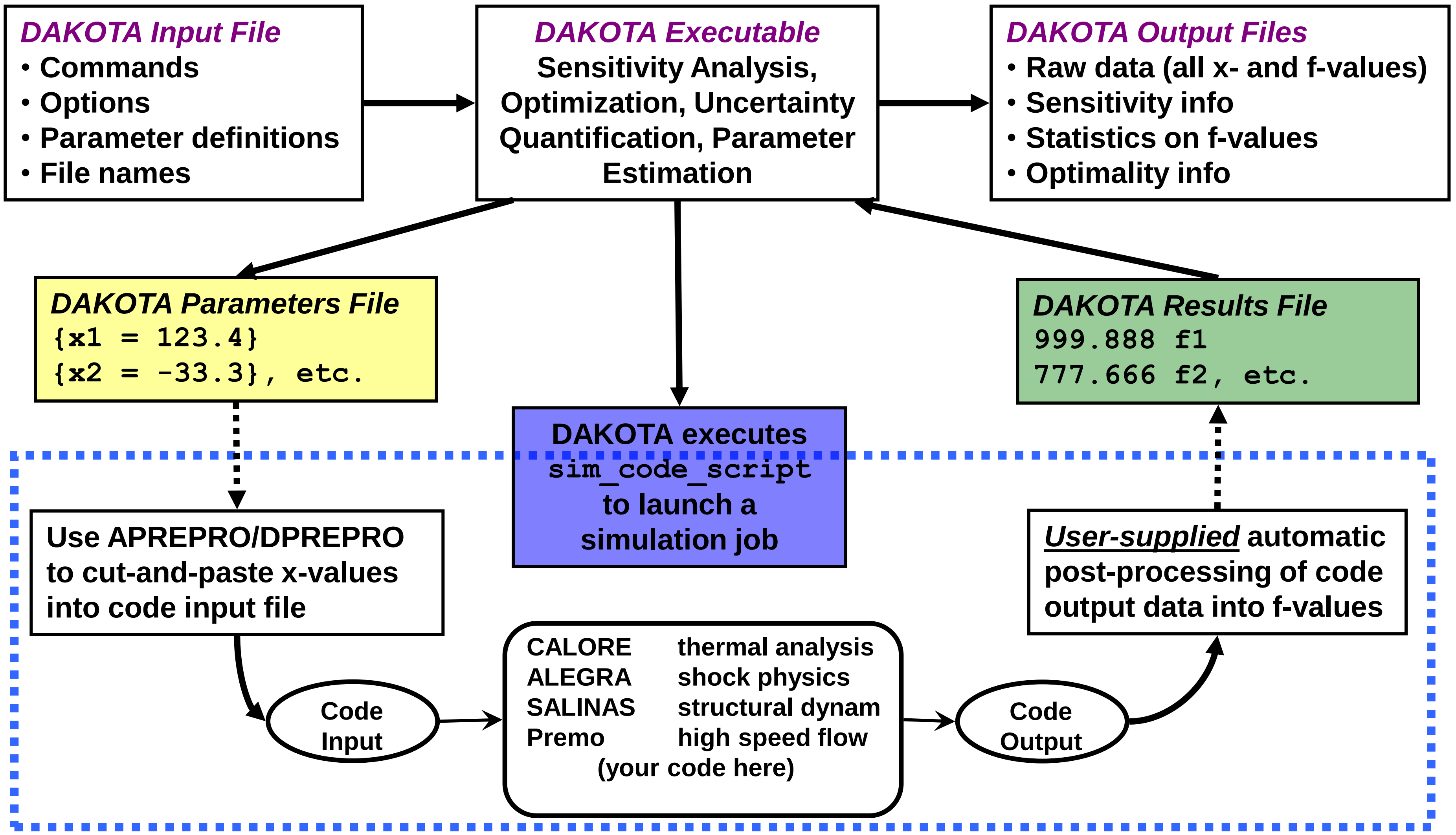
- See process of interfacing DAKOTA to a black-box application through file system
- See current state of DAKOTA library interface
- Understand MPI vs. local parallelism
- Understand modes of application parallelism (in queue, out of queue, serial, parallel apps)

- From DAKOTA 101:
  - Matlab, Python interfacing
  - DAKOTA as a library
  - Basics of HPC at SNL

Interface communicates through file system and user-supplied script



DAKOTA Execution & Info Flow



DAKOTA Application Interfacing Class

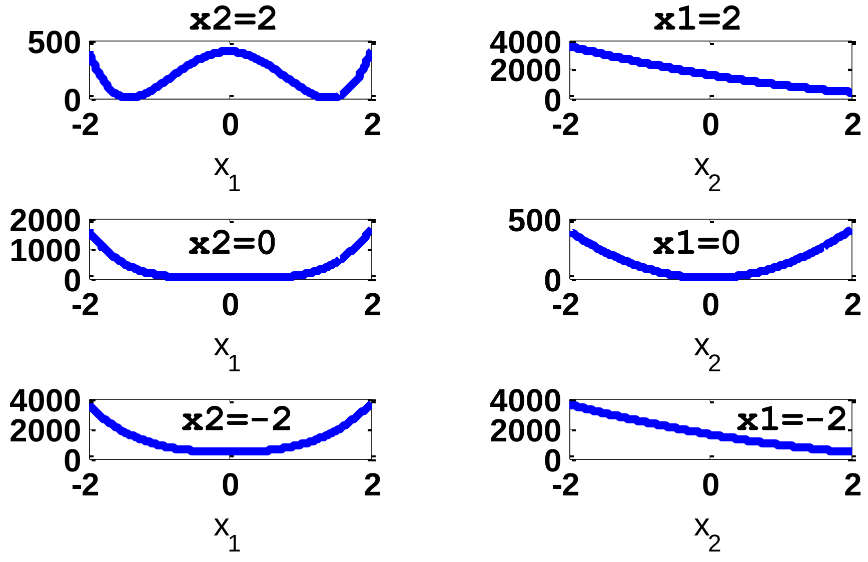
Application Stand-in: Rosenbrock "Banana" Function

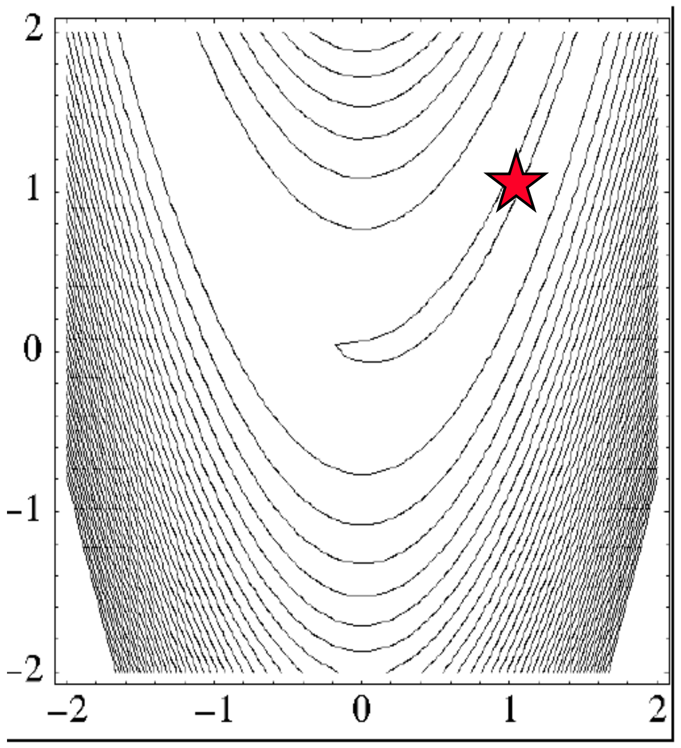
$f(x1,x2) = 100*(x2-x1*x1)^2 + (1-x1)^2$

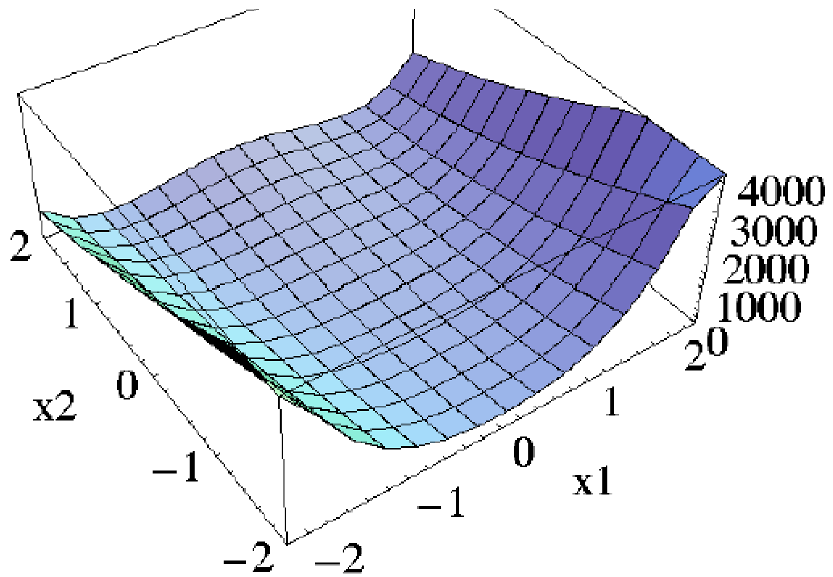
$-2 \leq x1 \leq 2$

$-2 \leq x2 \leq 2$

Minimum: $f(x1,x2) = f(1,1) = 0.0$







Demo:
Rosenbrock as a “black box”

- Locate example in `Dakota/examples/script_interfaces/generic`

- Described in DAKOTA 5.2 User’s Manual 18.1

- Explore top-down (DAKOTA down to application and back)

- Since you’re familiar with your application, may want to build from application up

Interfacing to Your Simulation (Assuming Text-based I/O)

1. **Annotate your input file to create template**
   `{ stress }` `{ alpha1 }`
2. **Create a representative DAKOTA `params.in` file in aprepro format (see User's 11.6) and test:**
   ```
   dprepro params.in analysis.in.template analysis.in
   ```
3. **Verify commands to run application with `analysis.in`**
4. **Determine how to automatically extract results of interest (direct application to export, shell commands, python, perl, visual basic, etc.) to create `results.out` (see User's 13.2)**
5. **Assemble into a script, e.g., `run_analysis.sh`; test script with sample params.in:**
   ```
   ./run_analysis.sh params.in results.out
   ```
6. **Test with a simple DAKOTA input deck, e.g., parameter study**

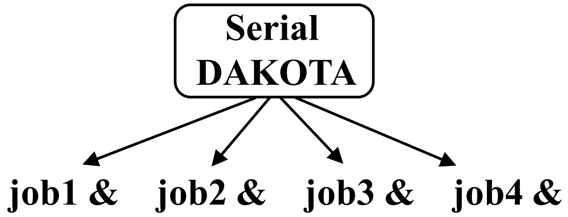
Parallelism

- See Application Parallelism slides shipped in Dakota/examples/parallelism

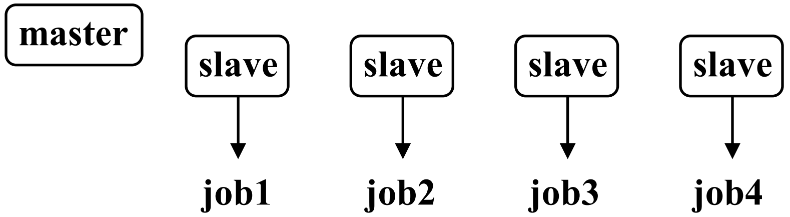
Parallelism from a computing platform perspective

Nested parallel models support large-scale applications and architectures.

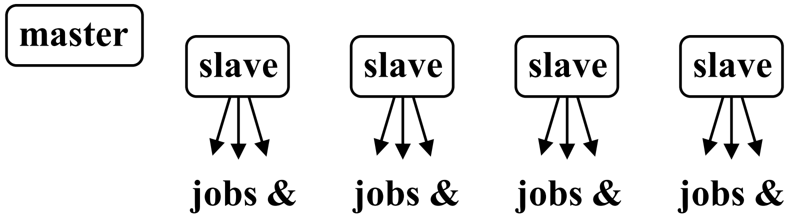
1. SMP/multiprocessor workstations: Asynchronous (external job allocation)



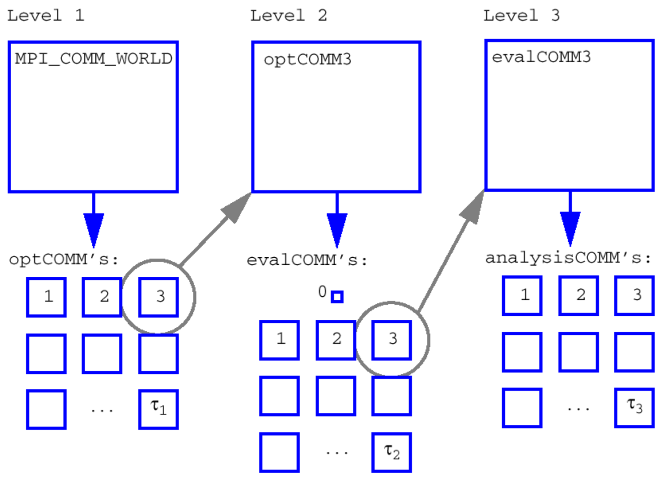
2. Cluster of workstations: Message-passing (internal job allocation)



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



4. MPP: Internal MPI partitions (nested parallelism)



Parallelism from an algorithmic perspective

1. *Algorithmic coarse-grained parallelism*: independent fn. Evaluations performed concurrently:
   - Gradient-based (e.g., finite difference gradients, speculative opt.)
   - Nongradient-based (e.g., GAs, PS, Monte Carlo)
   - Approximate methods (e.g., DACE)
   - Concurrent-method strategies (e.g., parallel B&B, island-model GAs, OUU)
2. *Algorithmic fine-grained parallelism:* computing the internal linear algebra of an opt. algorithm in parallel (e.g., large-scale opt., SAND)
3. *Function evaluation coarse-grained parallelism:* concurrent execution of separable simulations within a fn. eval. (e.g., multiple loading cases)
4. *Function evaluation fine-grained parallelism:* parallelization of the solution steps within a single analysis code (e.g., SALINAS, MPSalsa)