Dakota Software Training

Parallelism

http://dakota.sandia.gov
Module Goals

- Discuss what to consider when designing a parallelized study
- Understand what Dakota provides and its limitations
- Be able to choose the best parallelism approach
- Know how to configure Dakota and your interface for your parallelism approach
Opportunities for parallelization

Example 1: Parallel simulation
- The user’s simulation code has been parallelized using MPI, OpenMP, GPU, etc.

Example 2: Gradient-based optimization
- Finite differencing can be performed in parallel

Example 3: Sampling
- Every sample is independent of all the others

Example 4: Multi-start optimization
- Every optimization is independent of all the others
Things to Consider

- **Available Concurrency**
  - Adaptive vs. single pass algorithms

- **Characteristics of your simulation**
  - Serial or parallel
  - Parallel scaling/efficiency
  - Memory requirements
  - Duration

- **Characteristics of computing resource**
  - Number of cores and memory
  - Time limits
  - On some HPCs, “fork” and “system” are disallowed
Local Parallelism

One instance of Dakota launches multiple instances of the analysis driver

- Simple and portable
- Works with either serial or parallel simulation codes
- Method of choice for desktop computing

- Evaluations will not be launched across a network (Hence “local”)
- Iterators run sequentially
Suppose your simulation has been parallelized and your workstation has **24** cores.

Naturally, you want to use all of them and minimize how long your Dakota study will take.

**Which combination is best?**

<table>
<thead>
<tr>
<th>Evaluation Concurrency</th>
<th>Cores per Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
</tr>
</tbody>
</table>
Serial versus Parallel Simulation

- **Parallel efficiency**
  - Fewer cores are better

- **Memory requirements**
  - Upper limit on number of concurrent evaluations

- **Available Concurrency**
  - Another upper limit on number of concurrent evaluations

---

Amdahl’s Law

![Graph showing speedup vs. number of cores with different lines for 100.0%, 99.5%, 99.0%, 95.0%, and 90.0% efficiency.]
Parallel Dakota

$ mpirun -np 4 dakota my.in

Dakota launched in parallel; each “rank” runs analysis drivers

- Still pretty simple..
- Works across the network
- Parallel iterators (experimental)
- Dakota highly configurable

- Serial simulations ONLY
- Not supported on Windows
- Dakota must be built with MPI support
- Dakota highly configurable
Dakota, “Large” Simulations, and HPC

How can Dakota manage evaluations that require large*, parallel simulations on many cores?

*More than will fit on a workstation

Two strategies—

• Evaluation Submission
• Evaluation Tiling
**Approach 1: Evaluation Submission**

### Evaluation Steps

1. Dakota invokes analysis driver as usual
2. Driver performs pre-processing
3. Driver submits a job to the queue and waits for it to finish
4. Job starts, runs the simulation, and finishes
5. Driver performs post-processing and exits
6. Dakota reads results file and continues

<table>
<thead>
<tr>
<th>JOB</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval.1</td>
<td>Running</td>
</tr>
<tr>
<td>eval.2</td>
<td>Running</td>
</tr>
<tr>
<td>eval.3</td>
<td>Running</td>
</tr>
<tr>
<td>eval.4</td>
<td>Waiting</td>
</tr>
<tr>
<td>eval.5</td>
<td>Waiting</td>
</tr>
<tr>
<td>eval.6</td>
<td>Waiting</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>eval.N</td>
<td>Waiting</td>
</tr>
</tbody>
</table>
Analysis driver snippet

```bash
#!/bin/bash

#SBATCH --nodes=64
#SBATCH --time=08:00:00
#SBATCH --account=my_account
#SBATCH --job-name=eval.1

module load my_simulation

mpirun -np 1024 my_simulation
```

---

Example Interface

```bash
# Pre-processing done above (omitted)

sbatch eval.sbatch > sbatch.out

# Wait until the batch job finishes before continuing.

jobid=$(tail -1 sbatch.out | grep -o '[0-9]+')

while [ $(squeue -j $jobid | wc -l) -ne 0 ];
do
  sleep 300
done

# Post-processing done below (omitted)
```
Instead of waiting

When using ‘single-pass’ methods, Dakota can be run in two steps

- **Step 1: Job Creation**
  - Analysis driver set up to submit jobs then immediately exit, returning “dummy” values to Dakota

- **Step 2: Data Collection (after all jobs have finished)**
  - Analysis driver set up to post-process and return real result to Dakota

**Tip:** Dakota must generate the same parameters in both steps. For stochastic methods use the seed keyword.
Recommended Dakota Input

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
<td></td>
</tr>
<tr>
<td>analysis_driver &quot;driver.sh&quot;</td>
<td></td>
</tr>
<tr>
<td>fork</td>
<td></td>
</tr>
<tr>
<td>asynchronous</td>
<td>Submit multiple jobs</td>
</tr>
<tr>
<td>evaluation_concurrency 20</td>
<td></td>
</tr>
<tr>
<td>allow_existing_results</td>
<td>Prevent Dakota from erasing existing results</td>
</tr>
<tr>
<td>work_directory &quot;runs/run&quot;</td>
<td>Keep simulation run files separate from one another and preserve run folders</td>
</tr>
<tr>
<td>directory_tag</td>
<td></td>
</tr>
<tr>
<td>directory_save</td>
<td></td>
</tr>
</tbody>
</table>
Approach 2: Evaluation Tiling

Evaluation Steps

1. Dakota invokes analysis driver as usual
2. Driver performs pre-processing
3. Driver determines node placement (if necessary)
4. Driver launches parallel simulation
5. Driver performs post-processing and exits
6. Dakota reads results file and continues
Node Placement Methods

Automatic tiling

- just launch (srun, aprun)

Relative node list or Machine files

- Compute list of relative nodes based on—
  - Number of nodes in allocation
  - Number of MPI tasks per node
  - Number of MPI tasks per simulation run
  - evaluation number (obtain from e.g. file_tag)

- Then launch simulation with relative node list option (-host) or machinefile option (-machinefile)

- Use local_evaluation_scheduling static

- Examples in
  - examples/Case3_OpenMPI/
  - examples/Case3_MachinefileMgmt/
Example Analysis Driver

## Pre-processing done above (omitted)

APPLIC_PROCS=2

# Simple case: srun -n $APPLIC_PROCS my_simulation

num=$(echo $params | awk -F. '{print $NF}')

CONCURRENCY=4

PPN=16

applic_nodes=($(( (APPLIC_PROCS+PPN-1) / PPN )))

relative_node=($(( (num - 1) % CONCURRENCY * APPLIC_PROCS / PPN )))

node_list="+n${relative_node}"

for node_increment in $(seq 1 $((applic_nodes - 1))); do
    node_list="$node_list,+n$((relative_node + node_increment))"
done

mpirun -np $APPLIC_PROCS -host $node_list my_simulation

sleep 30

## Post-processing done below (omitted)
## Recommended Dakota Input

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>interface</code></td>
<td><code>analysis_driver &quot;driver.sh&quot;</code></td>
<td>Run multiple concurrent evaluations</td>
</tr>
<tr>
<td><code>fork</code></td>
<td><code>asynchronous</code></td>
<td>Run multiple concurrent evaluations</td>
</tr>
<tr>
<td><code>evaluation_concurrency</code></td>
<td><code>4</code></td>
<td>Run multiple concurrent evaluations</td>
</tr>
<tr>
<td><code>local_evaluation_scheduling</code></td>
<td><code>static</code></td>
<td>Use static scheduling</td>
</tr>
<tr>
<td><code>file_tag</code></td>
<td></td>
<td>File tagging to extract evaluation number</td>
</tr>
<tr>
<td><code>work_directory</code></td>
<td>&quot;runs/run&quot;</td>
<td>Keep simulation run files separate from one another</td>
</tr>
<tr>
<td><code>directory_tag</code></td>
<td><code>directory_save</code></td>
<td>Keep simulation run files separate from one another</td>
</tr>
</tbody>
</table>

---

17
Tiling versus Submission

Consider submission when..

- Memory or core count requirements are large
- Fork/system is disallowed on the compute nodes

Consider tiling when..

- Memory or core count requirements are modest
- Using an adaptive method
Examples and Documentation

- **Examples folder** (examples/parallelism)
- **User’s Manual** (Chapter 17)

**Note:** In these resources, running Dakota in parallel is referred to as “Case 1” parallelism, Evaluation Submission is “Case 4,” and Evaluation Tiling is “Case 3.” (Sorry.)