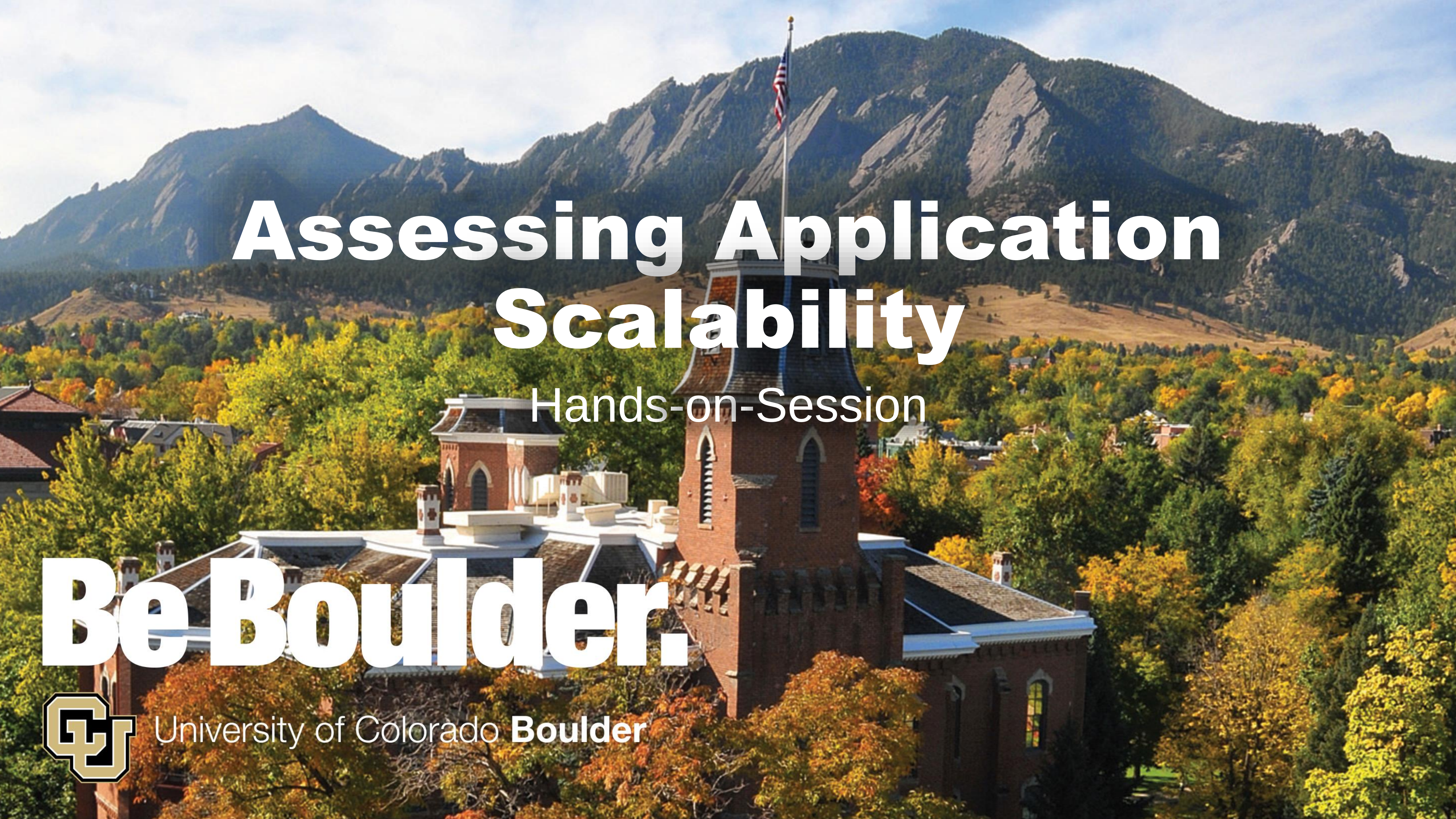


Assessing Application Scalability

Hands-on-Session

Outline

- What do we mean by “scaling”?
- Strong vs. Weak Scaling
- Example and Exercise

Analysis of Parallel Performance

Run a sample problem at multiple core counts and measure elapsed time.
(i.e., conduct a scaling study).

As core count increases, how does your application perform relative to ideal behavior?

Weak Scaling:
Increase problem size alongside number of cores used.
"Can I run a larger problem?"

Strong Scaling:
Fix problem size, and increase core count.
"Can I decrease my time to solution?"

Ideal Scaling

The expected performance in the absence of any communication overhead, etc.

Never achieved in practice, but standard reference point.

Ideal weak scaling:

Time-to-solution is independent of core-count (though may depend on global problem size)

Ideal strong scaling:

Time-to-solution decreases as $1/n_{cores}$

Efficiency

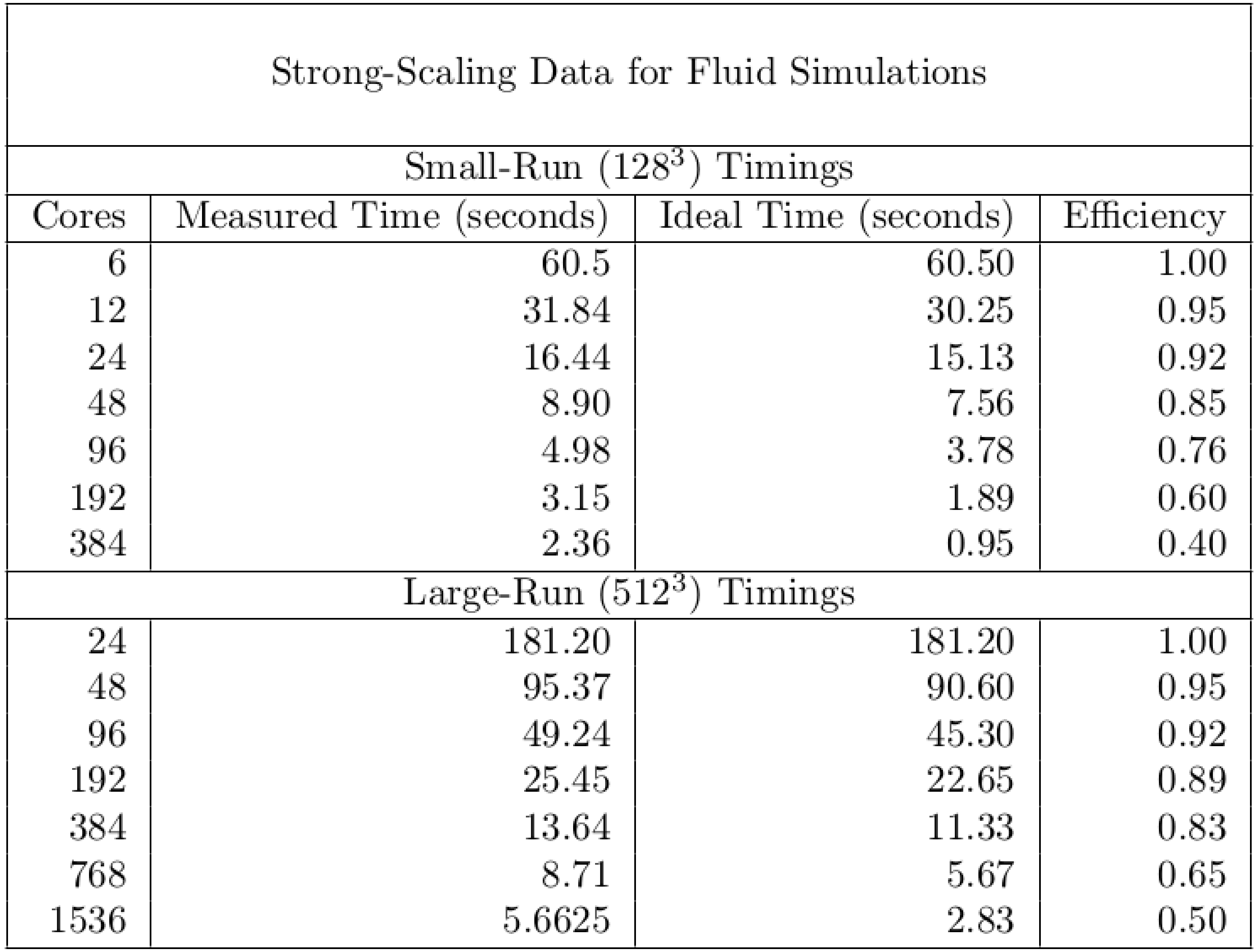
- Efficiency = Expected Time / Actual Time
- Expected (ideal) time is often taken relative to application's serial performance, or performance at a low core count.
- Good efficiency? Who knows? Trade-off between CPU hours and human effort.
- Suggestion: Always run your application with core counts that realize at least 80% efficiency (90 is better…).

Scaling Study Procedure

- Pick a few representative problem sizes (e.g., number of images to process, size of grid domain, number of genomes to examine)
- Decide on a sensible duration for the test. If code iterates in some fashion, run for just a few iterations.
- Run code for a short time at each problem size and with multiple core counts.
- For each test, record or calculate elapsed time, expected time, efficiency
- Plot results and ask for CPU time!

Sample Strong Scaling Data

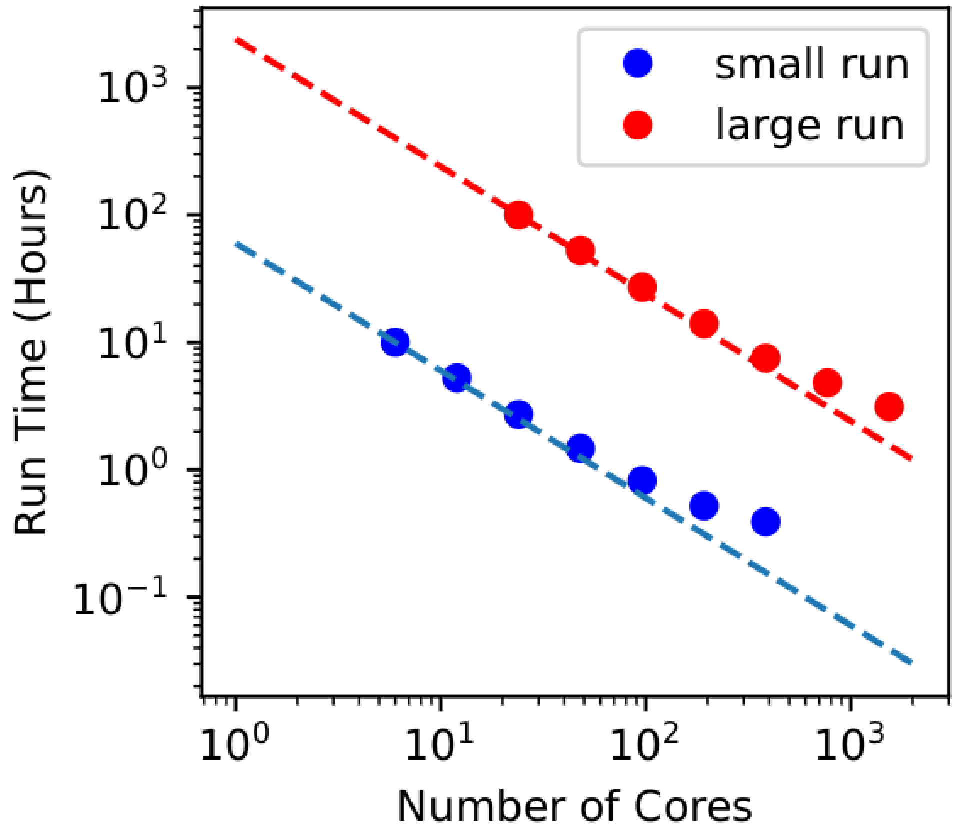
Fixed problem size at different core counts.

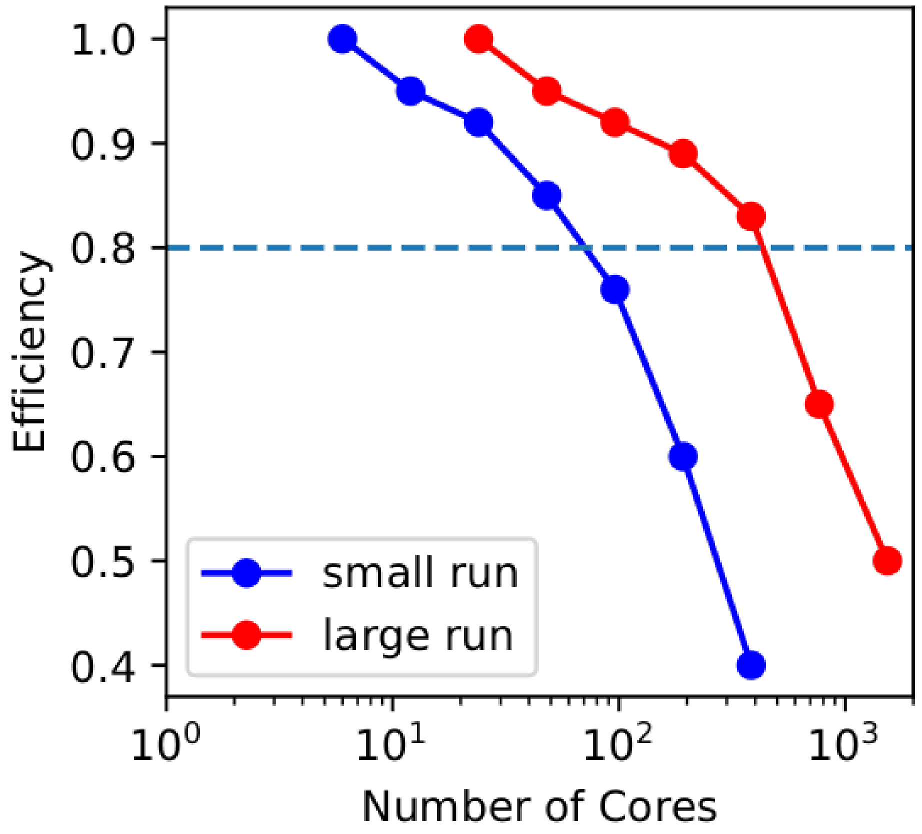


Strong-Scaling Data for Fluid Simulations

Cores	Measured Time (seconds)	Ideal Time (seconds)	Efficiency
Small-Run (128^3) Timings			
6	60.5	60.50	1.00
12	31.84	30.25	0.95
24	16.44	15.13	0.92
48	8.90	7.56	0.85
96	4.98	3.78	0.76
192	3.15	1.89	0.60
384	2.36	0.95	0.40
Large-Run (512^3) Timings			
24	181.20	181.20	1.00
48	95.37	90.60	0.95
96	49.24	45.30	0.92
192	25.45	22.65	0.89
384	13.64	11.33	0.83
768	8.71	5.67	0.65
1536	5.6625	2.83	0.50

Strong Scaling Plots

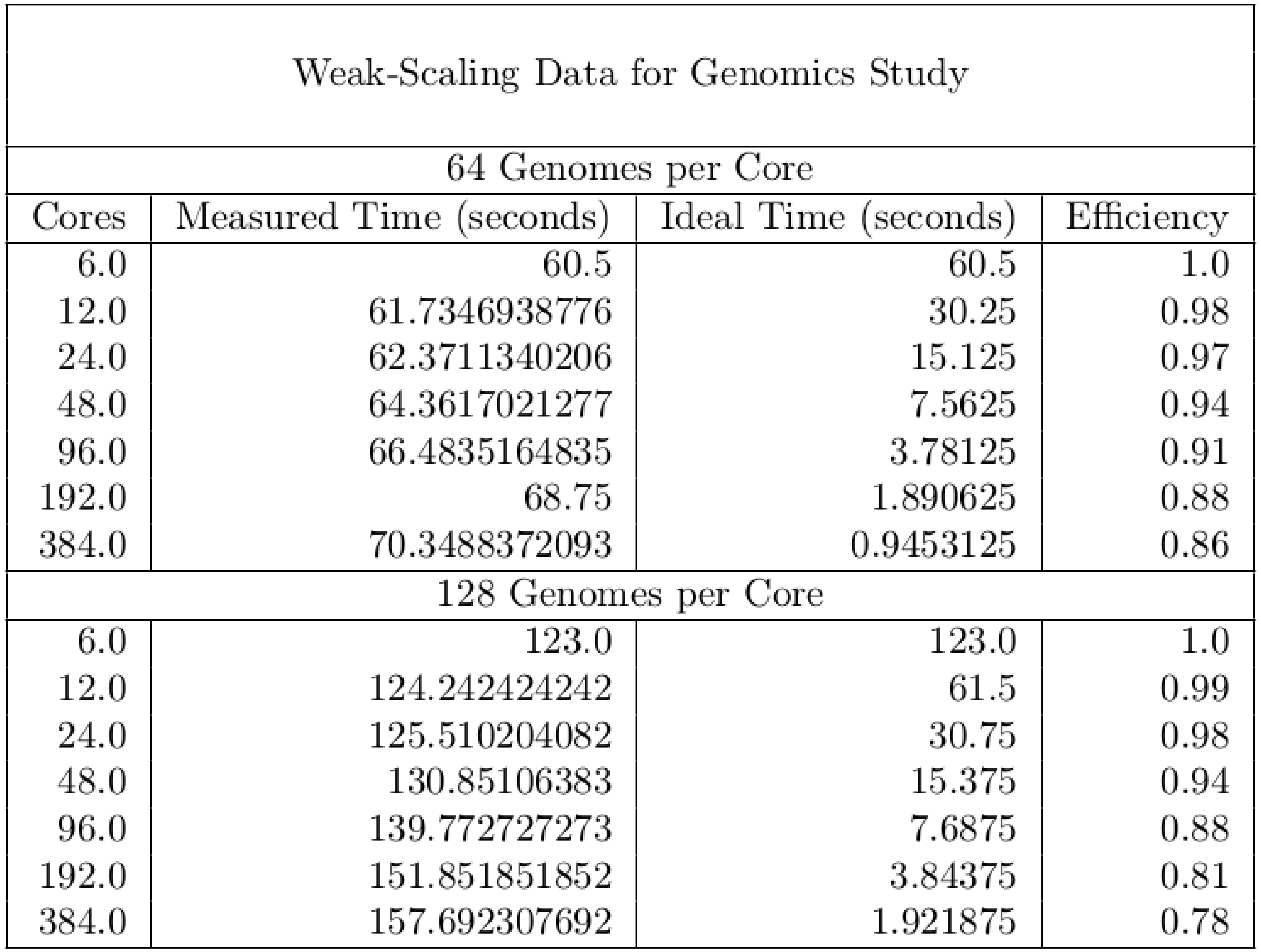




Suggestion: Run with at least 80% efficiency

Sample Weak Scaling Data

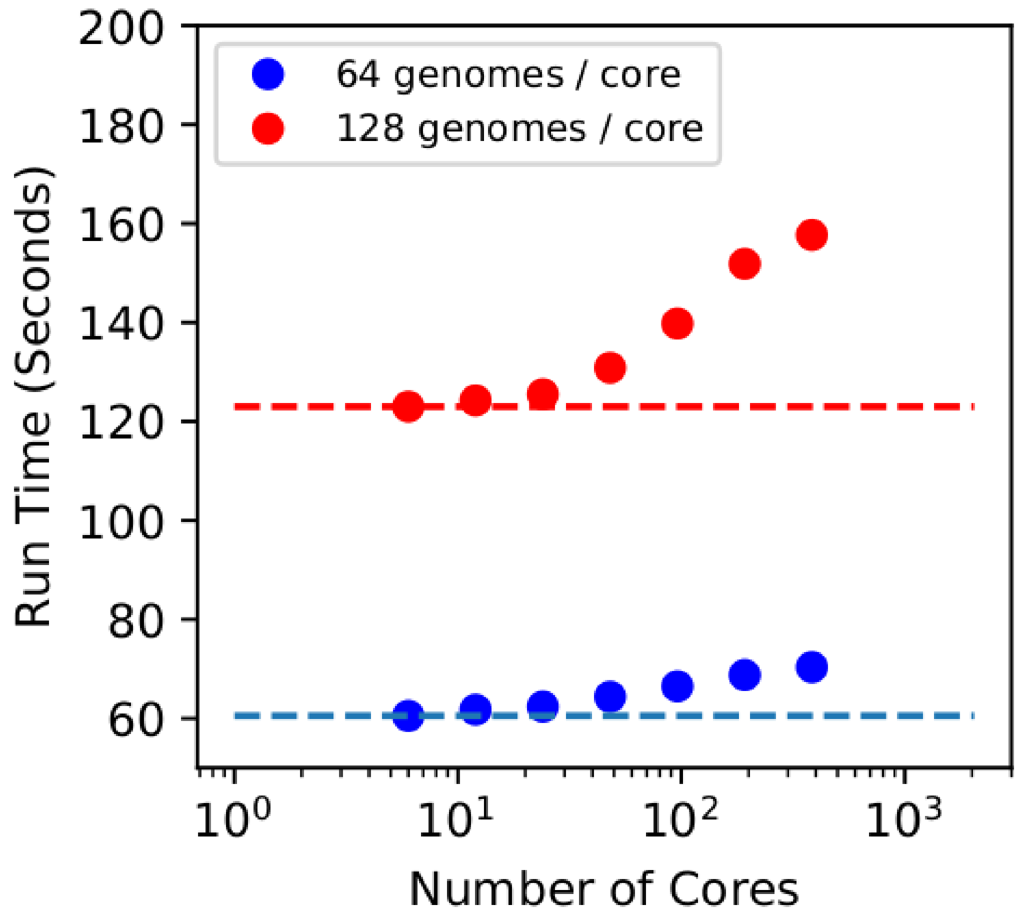
Vary problem size in proportion to core count.

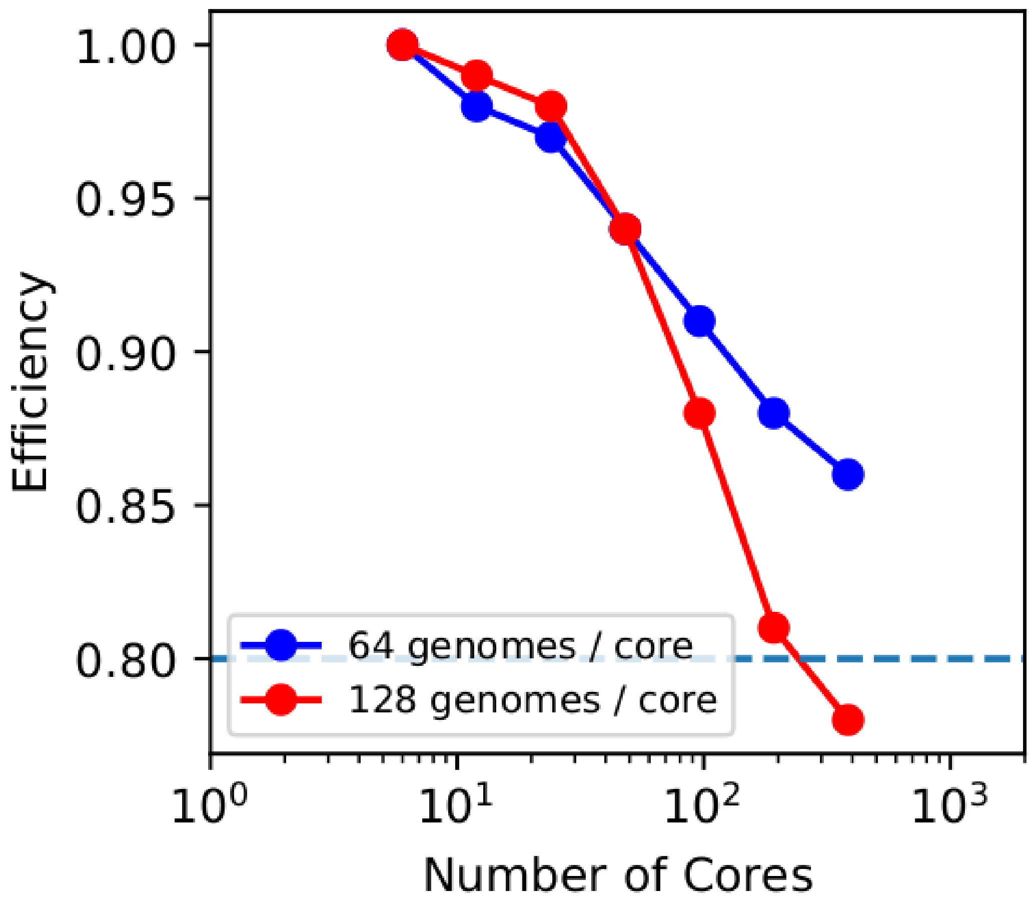


Weak-Scaling Data for Genomics Study

Cores	Measured Time (seconds)	Ideal Time (seconds)	Efficiency
64 Genomes per Core			
6.0	60.5	60.5	1.0
12.0	61.7346938776	30.25	0.98
24.0	62.3711340206	15.125	0.97
48.0	64.3617021277	7.5625	0.94
96.0	66.4835164835	3.78125	0.91
192.0	68.75	1.890625	0.88
384.0	70.3488372093	0.9453125	0.86
128 Genomes per Core			
6.0	123.0	123.0	1.0
12.0	124.242424242	61.5	0.99
24.0	125.510204082	30.75	0.98
48.0	130.85106383	15.375	0.94
96.0	139.772727273	7.6875	0.88
192.0	151.851851852	3.84375	0.81
384.0	157.692307692	1.921875	0.78

Weak Scaling Plots





Let's Analyze a Sample Program

- …./Scaling_Hands_On/scale.f90
- 2-D, iterative smoothing operation with nearest neighbor communication
- Grid dimensions (nx,ny) and number of iterations (nt) can be controlled via command-line arguments.
- We are going to run (but not edit this program)

Compilation and Running

- To compile, type:
  - module purge
  - module load intel/psxe-2018u1
  - make scale
- Running the code:
  - sbatch job.sh
- Grid dimensions (nx,ny) and number of iterations (nt) can be controlled via command-line arguments:

```
mpiexec –np 16 ./scale.out –nx 128 –ny 256 –nt 100
```

Exercise

- Generate strong and weak scaling data for scale.exe and plot the timing and efficiency results. Indicate ideal scaling and 80% efficiency levels on these plots.

- Running the code:
  - sbatch job.sh

- Code outputs: n_{cores}, time, nxglobal, nyglobal, niter

- Grid dimensions (nx,ny) and number of iterations (nt) can be controlled via command-line arguments:

```
mpiexec –np 16 ./scale.out –nx 128 –ny 256 –nt 100
```