CS 470
Spring 2017

Mike Lam, Professor

Performance Analysis

Performance analysis

- Why do we parallelize our programs?

Performance analysis

- Why do we parallelize our programs?
  - So that they run faster!

Performance analysis

- How do we evaluate whether we've done a good job in parallelizing a program?

Performance analysis

- How do we evaluate whether we've done a good job in parallelizing a program?
  - Asymptotic analysis (i.e., distributed sum)
  - Empirical analysis

Empirical analysis issues

- How do you measure time-to-solution accurately?
  - CPU cycles, OS clock "ticks", wall time, etc.
- How do you compare across systems?
  - Differing CPUs, memories, OSes, etc.
- How do you compare against the original?
  - 1-core parallel version will likely be slower
- How do you assess scalability?
  - Does performance improve as you add cores?
  - How do you quantify the improvement?
  - Is there a limit to how far we can improve performance?

Empirical analysis

T_s = serial time

T_P = parallel time

p = # of processes

S = speedup = $\dfrac{T_S}{T_P}$

should increase as p grows

E = efficiency = $\dfrac{S}{p} = \dfrac{T_S}{p\,T_P}$

usually decreases as p grows

Empirical analysis

T_s = serial time

T_P = parallel time

p = # of processes

$$S = \text{speedup} = \frac{T_S}{T_P}$$

should increase as p grows

$$E = \text{efficiency} = \frac{S}{p} = \frac{T_S}{p\,T_P}$$

usually decreases as p grows

r = serial % of original program

$$T_P = \frac{(1-r)T_S}{p} + r\,T_S$$

$$S = \text{speedup} = \frac{T_S}{\frac{(1-r)T_S}{p} + r\,T_S}$$

Empirical analysis

T_s = serial time

T_P = parallel time

p = # of processes

S = speedup = $\dfrac{T_S}{T_P}$

should increase as p grows

E = efficiency = $\dfrac{S}{p} = \dfrac{T_S}{p\,T_P}$

usually decreases as p grows

r = serial % of original program

$$T_P = \frac{(1-r)\,T_S}{p} + r\,T_S$$

$$S = \text{speedup} = \frac{T_S}{\dfrac{(1-r)\,T_S}{p} + r\,T_S}$$

Amdahl's Law: $S \leq \dfrac{1}{r}$ as p increases

Amdahl's Law

p = # of processors

r = serial % of program

$$S = \text{speedup} = \frac{T_S}{\frac{(1-r)T_S}{p} + rT_S}$$

Amdahl's Law:

$$S \leq \frac{1}{r} \quad \text{as } p \text{ increases}$$

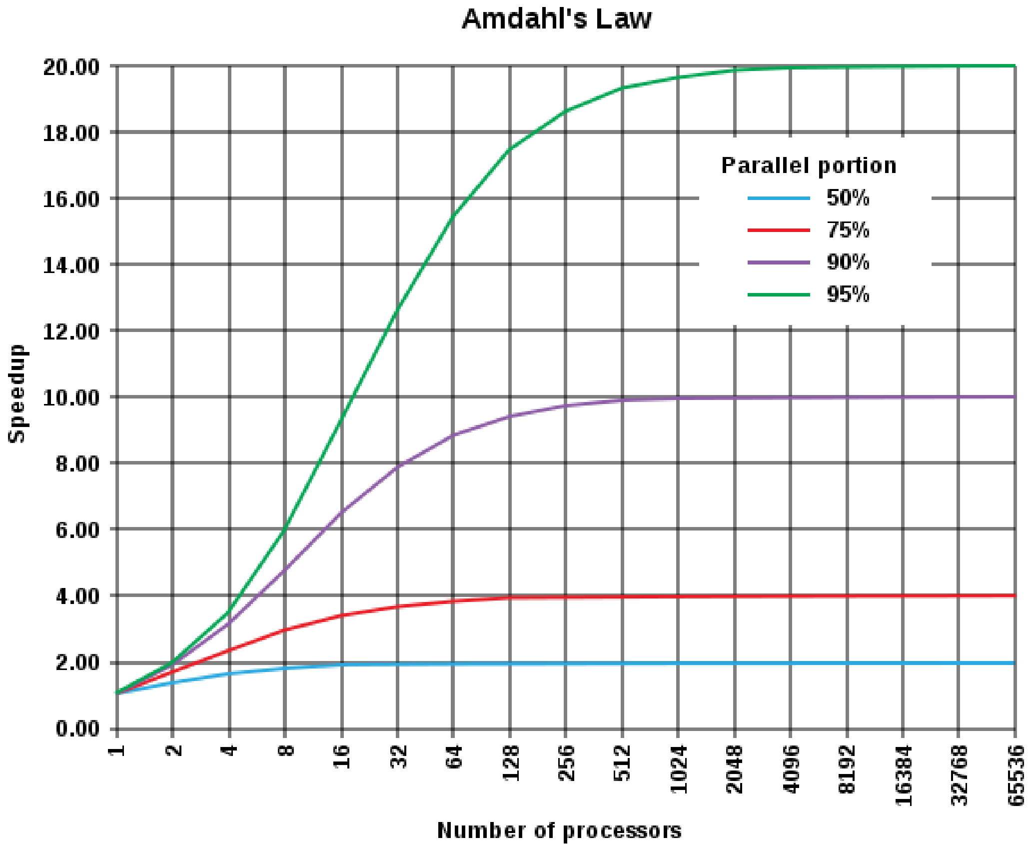
r = 50% → speedup limited to 2x

r = 25% → speedup limited to 4x

r = 10% → speedup limited to 10x

r = 5% → speedup limited to 20x

Speedup limited inversely proportionally by serial %

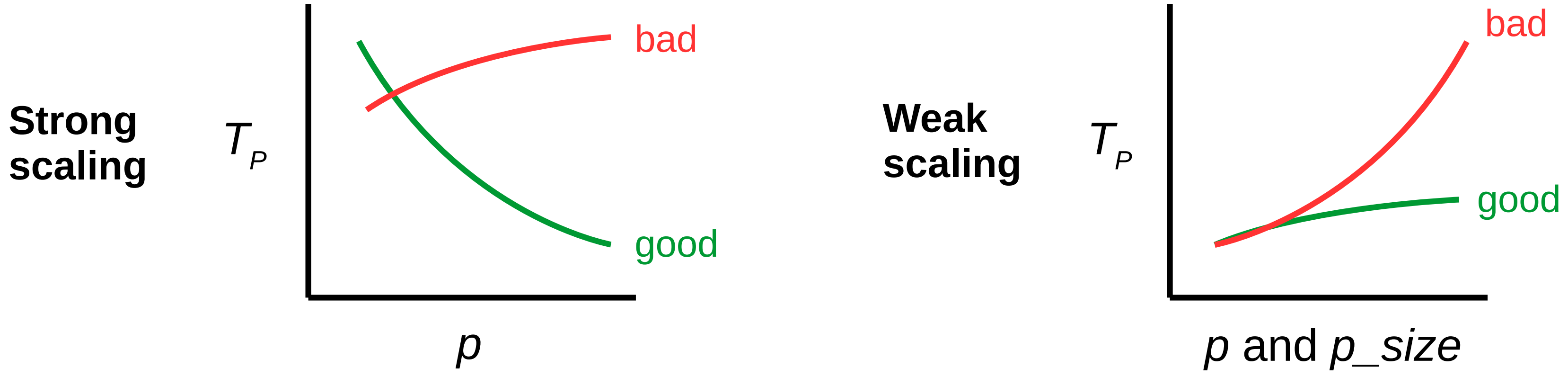


Scaling

- Generally, we don't care about any particular T_P
  - Or with how it compares to T_S (except as a sanity check)
- We care more about how T_P , S, and E change as p increases
  - And/or as the problem size increases
  - In general, a program is scalable if E remains fixed as p and the problem size increase at fixed rates
- Strong scaling: as p increases, T_P decreases
  - Linear speedup: same rate of change (2x procs → half time)
- Weak scaling: as p increases AND the problem size increases proportionally, T_P stays roughly the same

Scaling

- Strong scaling: as p increases, T_P decreases
  - Linear speedup: same rate of change
  - Sublinear (most common) / superlinear (exceedingly rare) speedup
- Weak scaling: as p increases AND the problem size increases proportionally, T_P stays roughly the same



Cluster access

- Detailed instructions online:
  `w3.cs.jmu.edu/lam2mo/cs470/cluster.html`
- Connect to login node via SSH
  - Hostname: `login.cluster.cs.jmu.edu`
  - User/password: *(your e-ID and password)*
- Recommended conveniences
  - Set up public/private key access from `stu`
  - Set up `.ssh/config` entries
  - Install Spack for access to more software

Cluster access

- Things to play with:
  - "`squeue`" or "`watch squeue`" to see jobs
  - "`srun <command>`" to run an interactive job
    - Use "`-n <p>`" to launch *p* processes
    - Use "`-N <n>`" to request *n* nodes (defaults to *p*/8)
    - The given "`<command>`" will run in every process
  - "`salloc <command>`" to run an interactive MPI job
    - Use "`-n <p>`" to launch *p* MPI processes

```
srun hostname
srun -n 4 hostname
srun -n 16 hostname
srun -N 4 hostname
srun sleep 5
srun -N 2 sleep 5
```

```
salloc -n 1 mpirun /shared/mpi-pi/mpipi
salloc -n 2 mpirun /shared/mpi-pi/mpipi
salloc -n 4 mpirun /shared/mpi-pi/mpipi
salloc -n 8 mpirun /shared/mpi-pi/mpipi
salloc -n 16 mpirun /shared/mpi-pi/mpipi
(etc.)
```

What's the max *n*?