CMOR 421/521: Why parallelize?

M	W	F	
			1
			2
			3
			4
			5
			6
			7
			8
			9
			10
			11
			12
			13
			14
			15

Topics

- What problems do we parallelize?
- Faster workers vs more workers
- Theoretical benefits of parallelization
  - Strong and weak scaling
  - Strong scaling: Amdahl's Law
  - Weak scaling: Gustafson's Law
  - Parallel efficiency
  - Divisibility of problem size
- Different parallelization paradigms
  - Shared memory and message passing

Why Parallelize?

- Memory
  - Individual computers may not have the memory needed
  - Even if they did, the caching effects could be limiting
- Parallelizing allows slow programs to be run within their projects’ time constraints
- Parallelization also allows some programs to be run at all (agonizingly slowly, but they’ll run)
- **Feasibility before convenience**

Quality vs Quantity: Faster vs More

Why not faster computers?

- Faster computers work on higher frequencies
- Higher frequencies generate more heat and consume more power
    - Supercomputers consume huge amounts of energy
    - Some of that energy is from computing, some is from cooling
    - Industry is experimenting with hot server rooms, water cooling, etc because of this

The Caveat:

9 women can’t make a baby in 1 month.

Parallel Performance

- **Not all parts of a program can be parallelized**
- The benefits of parallelization are limited.
- Define the work fractions:
  - $p =$ The parallelizable portion of the work
  - $s =$ The sequential portion of the work
  - $1 = s + p$

Parallel Performance

- Consider a program that takes $T(n)$ time to run on n processors. The time can be expressed as:

$$T(n) = T_s(n) + T_p(n)$$

where

- $T_s(n) =$ Time spent on the sequential portion

- $T_p(n) =$ Time spent on the parallelizable portion

Parallel performance

- From before:

$$T(n) = T_s(n) + T_p(n)$$

Assuming that serial runtime T_s is independent of n:

- $T_s(n) = T_s = sT_1$ (s is the sequential portion of work)
- $T_p(n) = \frac{1}{n} pT_1$

Strong Scaling: Amdahl’s Law

- Parallel speed-up is defined as:

$$S(n) = \frac{T(1)}{T(n)}$$

- There are different ways of measuring speed-up related to scaling
- *Strong scaling*: The amount of work done is fixed
- *Weak scaling*: The amount of work done scales with n

Strong Scaling: Amdahl's Law

- Speed-up for strong scaling is then given by:

$$S(n) = \frac{T(1)}{T(n)} = \frac{1}{s + \frac{p}{n}} < \frac{1}{s} = \lim_{n \to \infty} S(n)$$

Note:

- The speed-up is asymptotic wrt n!
- You cannot get arbitrarily large speed-up
- As $n \to \infty$, the time needed to do the sequential part remains the same

Parallel Efficiency

- Parallel efficiency is a measure of how much each processer contributes to the parallel speed-up

$$E(n) = \frac{S(n)}{n}$$

- A problem can have an optimal number of processers
- Using more than that number of processers will see diminishing returns on speed-up

Weak Scaling

- Often the size of the problem N is proportional to the number of processers
- Additionally, the sequential part of many programs, does not scale with the problem size (or scales minimally); this means $s \rightarrow s_N$, $p \rightarrow p_N$
- It makes more sense for some applications to discuss their *weak scaling*
  - i.e., how their performance scales when we **hold the work per processor constant**

Weak Scaling vs Strong Scaling

- Consider the amount of work being done for a problem of size N:

$$W(N) = W_s(N) + W_p(N)$$

- As with time, we can divide the work in terms of the part that can only be done sequentially and the parallelizable part

Weak Scaling vs Strong Scaling

- For a given amount of work, we can define the work fractions s and p as:

$$s_N = \frac{W_s(N)}{W(N)}, \qquad p_N = \frac{W_p(N)}{W(N)}$$

- Strong scaling assumed N was fixed, so it did not matter how s and p change with the problem size
- Weak scaling assumes $W_s(N)$ is actually constant, which means s and p change with N

Weak Scaling: Gustafson's Law

- Assume without loss of generality that $T(n,N) = 1$

$$S(n,N) = \frac{T(1,N)}{T(n,N)} = \frac{s_N T(n,N) + np_N T(n,N)}{1} = s_N + np_N$$

- (Weak) speed-up is linear; it can be arbitrarily large!
- Weak speed-up measures how much time it would take one processer to do n processers amount of work

Weak Scaling vs Strong Scaling

- Strong scaling may make more sense intuitively, but often weak scaling is more practical
- When benchmarking, both may be reported
- Amdahl’s and Gustafson’s laws provide theoretical bounds on speed-up; actual performance may vary due to the assumptions made + real world considerations

Real World Considerations

- Divisibility limits of the problem
  - Work is often defined discretely
  - This is why weak scaling makes more sense for some problems
  - Example: The number of entries in a matrix

- Parallelization can impact the sequential time
  - Often minimally though
  - The sequential work may also grow with problem size

Parallelization Paradigms

- Parallel work can be split into two sections: computation and communication
- Parallel computation requires multiple workers; it looks the same in most circumstances
  - Special consideration may need to be made for how computations are actually done
- There are multiple ways to implement communication however

Parallelization Paradigms

Different paradigms exist for handling communication:

- Shared memory
  - Communication is handled via shared memory
  - Appropriate for individual, multi-core CPUs
  - OpenMP: Open Multiple Processing
- Message passing
  - Communication is handled via explicit messages
  - Appropriate for multiple CPUs, GPUs
  - MPI: Message Passing Interface