FINISHING LAST CLASS'S NOTES ON MPI:

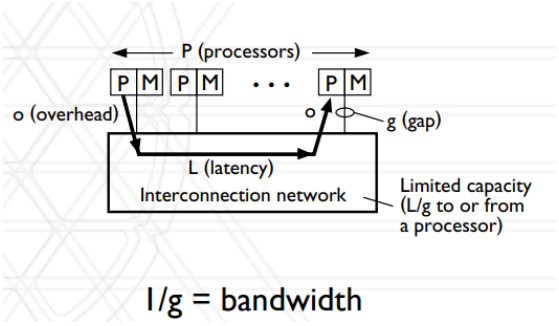
- Protocols for Sending a Message
    - Eager
        - Message sent assuming the destination can store
        - Used when there are very small messages
        - Assumes that receiving destination has enough space for the message
    - Rendezvous
        - Message only sent after handshake
        - Runtime does not assume that there is enough space
        - 2 step process
            - First step is handshake, and when it is posted or an ack is received:
            - Second step happens: this is where the actual message is sent
        - Opaque to programmer, happens inside MPI
    - Short
        - Data sent with the message envelope
        - Message envelope: NPI runtime creates envelope that has all the info and this is what is sent
    - Can switch between them with the MPI Environment variable
    - MPI Makes things portable
- Other MPI_send Modes
    - Basic mode: MPI_Send
    - Buffered Mode: MPI_Bsend
    - Synchronous Mode: MPI_Ssend
    - Ready Mode: MPI_Rsend
    - Last 3 not going to be used too much in class

Performance Modeling, Analysis, and Tools

- Weak Versus Strong Scaling
    - Strong Scaling: Fixed total problem size as we run on more processes
        - Ex:
            - 2n numbers on 1 process
            - 2n numbers on 2 processes
            - 2n numbers on 4 processes
            - 2n numbers on 8 processes
            - …

    - Weak Scaling: Fixed problem size per process but increasing total problem size as we run on more processes
        - Ex:
            - N numbers on 1 process
            - 2n numbers on 2 processes
            - 4n numbers on 4 processes
            - 8n numbers on 8 processes
            - …
        - Constant scaling
- Amdahl's Law
    - Very simple
    - If you have a parallel program, if you parallelize some parts of the program, that part will take a very short amount of time but the remaining portion is what takes more time and effort.
    - Speedup = $$\text{Speedup} = \frac{1}{(1-f)+f/p}$$
    - If you keep large portion of code as serial code, it will become a bottleneck
- Performance Analysis
    - Parallel performance of program may not be what the developer wants
    - Performance analysis = process of studying performance of parallel code
    - Why may performance be slow:
        - Serial performance
        - Serial bottlenecks when running in parallel
        - Communication overheats
    - Performance Analysis methods:
        - Analytical techniques: use algebraic formulae
            - Figure out if func derived is able to scale well or not
        - Time complexity analysis
            - Similar to analytical
        - Scalability analysis (isoEfficiency)
        - Model performance of various operations
            - Analytical models: LogP, alpha-beta model
        - Empirical performance analysis using tools
- Parallel Prefix Sum for n >> p
    - N = size of data, p = number of processes
    - Assign n/p elements (block) to each process
    - Perform prefix sum on these blocks on each process locally

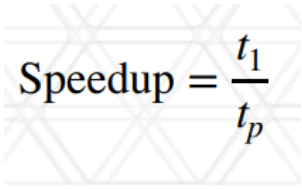
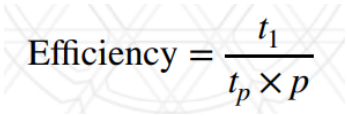
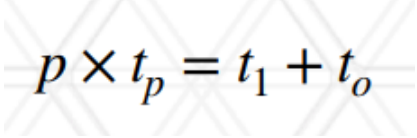
        - Serial part
        - Number of calculations: n/p
    - Then we do parallel algorithm with partial prefix sums
        - We determine the number of phases: log(p)
        - Total number of calculations: log(p) X (n/p)
            - The reason for this is that we have n/p calculations across log(p) phases on one process
            - Every process should be finishing around same amount of time since they are a very similar size
        - Communication: log(p) X 1 X c
            - 1 = number of processes
            - c = constant that is being used
        - Most of the time, there is no overlap between computation and communication
- Modeling Communication: LogP Model
    - One of the simpler models for communication on an interconnection network

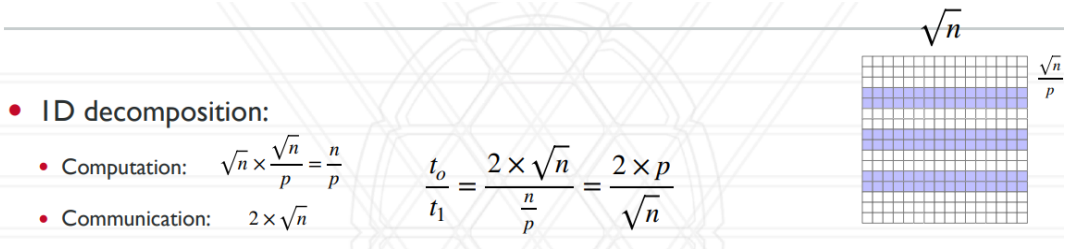


    - 1/g = bandwidth
    - L: Latency
        - The delay
    - O: Overhead:
        - When sending a message to processor, how long it is busy
    - G: Gap
        - Time between successive sends/receives
        - Depends on bandwidth
    - P:
        - number of processes
- Alpha + n 8 beta model
    - Another model for communication
    - Slightly easier than the one before

$$T_{\text{comm}} = \alpha + n \times \beta$$

    - a: latency

    - n: size of message
    - l/B: bandwidth cost
- Isoefficiency
    - What is it?
    - Relationship between problem size and number of processors to maintain a certain level of efficiency
    - Takes speed up equations and derives another eq that shows how much we need to increase the problem size with respect to number of processors in order to keep efficiency constant
- Speedup and efficiency
    - Speedup: ratio of time spent on 1 process to that on p processes
        - 
        -
        - For example:
            - With 1 process it takes 100 secs
            - With 4 process it takes 25 secs
            - Speedup = 100/25 = 4
    - Efficiency: speedup PER process
        - 
        -
        - You want number to be higher
        - Usually between 0 and 1
        - For previous ex:
            - Efficiency = (100) / (25 * 4) = 1.0
        - Possible that efficiency could be greater than 1
            - Called SUPERLINEAR SPEEDUP
                - Mostly caused due to cache effects
        - Now let's use efficiency to calculate isoefficiency
- Efficiency in terms of overhead
    - Total time spent in all processes = computation + overhead
        - Computation = useful stuff, stuff that would've been done if code was done sequentially
        - Overhead = extra computation + communication + idle time
        - 
        -
        - T1 = useful computation

        - T0 = overhead
        - P = number of processors
        - Tp = execution time using p processors
    - $$\text{Efficiency} = \frac{t_1}{t_p \times p} = \frac{t_1}{t_1 + t_o} = \frac{1}{1 + \frac{t_o}{t_1}}$$
- Isoefficiency function:
    - We have the efficiency equation from above:
    - $$\frac{1}{1 + \frac{t_o}{t_1}}$$
    - In order for it to be constant, we need t0/t1 to be constant
    - t0 = K x t1
        - K= constant
- Isoefficiency analysis
    - We will try to determine if 1d Decomposition is better than 2D decomposition here:
    - 1D Decomposition:
    - 

        - SQRT(N) long and wide on table
        - SQRT(N) x SQRT(N)/P = n/p
            - What is done on 1 process
        - Computation = useful work, t1
        - Communication = overhead, t0
        - K = t0/t1 so we plug the values into the formula since we want it to be constant
    - 2D Decomposition:
        - WILL TALK ABOUT IN NEXT CLASS