2.20.2024

MPI Wrap Up

Protocols for sending messages:

- Depending on the size of the message sent, MPI uses different algorithms/protocols.
- **Eager:** The sender assumes that the destination has enough space to store the message.
- **Rendezvous:** Runtime does not assume destination process has enough space to store – before the message, the sender and receiver does a handshake and sender tells the user to allocate X bytes for the messages.
- Note that as a programmer, you don't interact with them directly – they're all abstracted by MPI_Send/MPI_Recv/etc.
    - However, you can force MPI_Send to use Eager/Rendezvous by setting environment variables (e.g. increase cutoff for Rendezvous)
- This is **not** TCP/IP – the MPI runtime handle drops/disconnects for you. **Assume things will just work.**
    - Runtime will ensure things work correctly.
- **Short:** Wraps data sent in an envelope. The MPI runtime creates an envelope to send the message with runtime data and then puts the user data after that.

In terms of nodes, if we have a source node send and a destination node receive, under the hold, the runtime wraps the data in an envelope and sends it to the NIC (network interface card). The NIC sends the data down the physical network cables, routers, switches, etc. Message gets copied multiple times in transit.

The purpose of having short is to save space and maybe an extra underlying send.

MPI_Send will fail if the sends don't work.

Other MPI send modes (optional):

- MPI_BSend
- MPI_SSend

Networks understand **packets**, not messages. The lowest building block is a flit (width of the wire, represents amount of data that can be sent simultaneously).

Performance Modeling, Analysis, and Tools

Strong scaling: Fixed *total* problem size as we run on more processes.

- Sorting n numbers on 1, 2, 4, etc. processes\
- Note: Same n numbers being sorted on all – this problem does *not* change

Weak scaling: Fixed problem size *per process* but increasing total problem size as we run on more processes.

- Sorting n numbers on 1 process
- Sorting 2n numbers on 2 processes

- Sorting 4n numbers on 4 processes

Recall: Amdahl's Law

- Speedup is limited by the serial (sequential) portion of the code
- Let's say only a fraction f of the code can be parallelized on p processes
- $speedup = \frac{1}{(1-f)+\frac{f}{p}}$
    - The part we **can** parallelize is f/p
    - The part we **cannot** parallelize is 1-f
    - The limit as p goes to infinity is 1/(1-f)

Performance Analysis

- Sometimes, the performance of the parallel program may not be what the developer expects.
- Process of studying performance of code and figuring out where the performance issues happen.
- Some portions of the program may not have been parallelized. In the parallel part, there is code executing in parallel, but it's still sequential on each part.
    - Bottlenecks may happen in both serial parts and less optimal parallel code
    - Communication overheads can exist when communicating between processes

Performance Analysis Methods

- One extreme: Performance Modeling using analytical techniques.
    - In terms of data size (n), number of processes (p)
    - Time complexity analysis
    - Isoefficienty (scalability) analysis
    - Performance models
        - LogP
        - Alpha-beta model
        - Other pre-existing models exist
- Another extreme: Using graphical tools to measure performance
- How can we do a time complexity analysis for prefix sum?
    - Step 1: Assign n/p elements (block) to each process
    - Perform prefix sum on these blocks on each process locally
        - Number of calculations: n/p
    - Then do parallel algorithm with partial prefix sums:
        - Number of phases: log(p)
        - Total number of calculations: log(p) * n/p (recall we take the element from the previous process and add it to every element in the array)
        - Communication: log(p) * n/p * **1** * **c** (each phase sends 1 message of time c)
            - People often compute time complexity for computation, and then add in communication. In each phase, each process sends one element to a neighboring process.
            - From POV of one process: it's just log(p) * 1 * c
    - We'll do similar things to compute the time complexities of algorithms

LogP model

- L: Latency or delay (time spent over network)
- O: Overhead (processor busy in communication)
- G: gap (between successive sends/recvs)
- P: number of processors/processes

Alpha-beta model

- Alpha: Latency (time it takes to **start** sending messages over the network – not to be confused with LogP)
- N: size of message
- 1/beta: bandwidth
- T_comm = alpha + n * beta
- Alpha is a fixed-time startup cost
- The n*beta is the network cost

Isoefficiency

- Relationship between the problem size and number of processors to maintain a certain level of efficiency
- At what rate should we increase the problem size with respect to the number of processors to keep efficiency constant
- Iso refers to maintaining a certain level of efficiency
- If we can increase problem size, we might be able to maintain said level of efficiency

Speedup: ratio of execution time on one process to that on p processes

$$\frac{t_1}{t_p}$$

Efficiency: speedup per process

$$\frac{t_1}{t_p * p}$$

It's difficult to write parallel programs with perfect efficiency.

Efficiency in terms of overhead

Total time spent in all processes = useful computation (p*t_p) + overhead (extra computation (t_1) + (communication + idle time) (t_0))

$$p * t_p = t_1 + t_0$$

$$Efficiency = \frac{t_1}{t_p * p} = \frac{1}{1 + \frac{t_0}{t_1}}$$

Efficiency is constant of the t_0/t_1 term is constant

If so, we get

$$t_0 = K t_1$$

We want to keep serial bottlenecks as a constant multiple of the total problem size

How do I need to increase the problem size?

We're doing an Isoefficienty analysis on game of life

- In 1D composition
  - Assume an n by n board
  - Each process gets sqrt(n) by sqrt(n)/p rectangle of the board
  - Computation: $\sqrt{n} * \frac{\sqrt{n}}{p} = \frac{n}{p}$
  - Communication: Each process sends two messages and receives two messages $2 * \sqrt{n}$ – we're sending $\sqrt{n}$ numbers twice – for more detailed analysis use the Alpha-Beta model
- In 2D composition
  - To come, next class