Queue Assignment

Why Queue?

- Decouple sender and receiver
  - Separated in time
    - We don’t know when the other is ready
  - Unknown receiver?
    - Don’t care about who processes?

Some examples of uses

- Example: Distribute/schedule work
  - Post messages about some work/task that has to be done
  - Anyone can take it
  - Balances the load on nodes
  - Making sure everyone is working with something
- Decoupled communication
- Event-driven architectures(EDA)
  - Choreograph vs orchestrate
  - Many different micro-services, operating independently
- Stream processing
- Pipelines
  - Queues between steps

Some considerations

- One specific receiver, type of receiver, or not?
  - Rendevouz? Synchronization?
- Consumers subscribe to queues?
  - Filter?
- Does order matter?
  - If messages must be processed in a particular order
  - How to ensure that are messages are ordered correctly
    - «Happened before», relationships?
    - Some message numbering arrangement?

Example of solution for assignment

- Rank 0 can have special role
  - Can be main responsible for queue
    - Or be the keeper of things that the others need to cooperate
  - Doesn’t mean that everything has to be on rank 0
    - Memory can be spread on the nodes (global pointers)
  - Don’t need the whole queue logic on rank 0 either
    - Only the head or tail-data?
- Rank 0 can read initial work at start-up
  - Post messages to queue that the others can pick up
- Nodes read initial chunks of shared, large data
  - E.g. a very large matrix, partitioned into different parts, and kept in shared memory (PGAS)
- Nodes pick up work
  - Their work results in more work
  - Post this to queue for other to process

A general scenario

- On start-up
  - Data is read (from somewhere) and partitioned between processes
- Rank 0 posts initial work-messages to queue
  - Because we want to calculate something
- Nodes dequeue messages and perform calculations involving use of shared memory
- The processing done by the nodes may create various amounts of new work that has to be done
  - Unpredictable where more work must be done, and what is «finished» or an unfruitful path. Maybe we are searching for something, or gradually focusing on some promising paths
  - Iterative/loop
  - Post this new work to the queue as messages

Examples of implementation alternatives

- We mainly want non-blocking, unbounded versions
- **Keep queue at one node**
  - Use RPC
  - Single-threaded, which solves many problems
  - Bottleneck?
- **Distribute queue operations**
  - Nodes directly do queue operations
  - Lock-free, but use of **atomic operations on integers**
  - Rank 0 has an array of pointers (to head and tail)? Atomic stamped references? What to use? Research this.
  - A current-index is updated with atomic domains
    - The index is an integer manages through atomic domains
  - Messages can be located on any node
  - More complicated to manage. Think about concurrency.

Be aware of:

- ABA-problem
  - Chapter 10
- Use of atomic operations
  - Not C++ version, but UPC++ versions. *Atomic domains!*
  - Compare and set (CAS) and similar types of operations
  - To prevent data races, where to processes changes or use values at the same time, and «messes up» the data.
  - Ensures that the changes are visible to all nodes
  - Remember:
    - Hardware support
    - Remote direct memory access (RDMA) – Fast buses (Infiniband and similar)
    - Doest't involve OS! Much lower level …

Questions?

- Assignment text