CMSC416: Introduction to Parallel Computing

Topic: Task-based Prog. Models and Charm++, Load Balancing
Date: April 16, 2024

Proxy:

- Users don't know where the chares are placed, but someone has to know.
- Proxy objects know and keep track of where the real objects are

Broadcast, barrier, and reduction:

- So far, we have been looking at point to point communication. It's possible to do collective operations, but still this is a C++ library.
- entryMethod() is called on all elements of the chare array, thereby providing the capability of a broadcast.
- All of the method invocations on chare are asynchronous, so things won't happen in a certain order.
- It's on the programmer to synchronize and make sure things happen in a certain order.
- Calling the contribute() method on all elements of the chare array serves as a barrier/reduction call
- Chare doesn't have two separate functions for barrier or reduction.
- The barrier call is blocking.
- The contribute() can take parameters, which serves as a reduction with arguments. The three arguments are bytes, data, and reducer type. Bytes is the amount of data that is being sent in bytes. Data refers to a pointer to the buffer. ReducerType refers to the type of reduction operation.
- In order to store the final value of the reduction, the output is stored in a callback object called the reduction client. This callback argument is now specified in the contribute method.

2D stencil in charm++

- Stencil is the same as the game of life.
- The first step is to create a 2D chare array (array[2D] stencil).
- Then, each element of the chare array, similar to how it's done in MPI, is assigned a sub-block.
- Each element needs to communicate with its neighbors. This operation is done with a remote entry method, where the put would be put as an argument to the entry method. This method will be called on the neighboring elements.

Load Balancing

- Load imbalance is a performance issue.

- It's defined as unequal amounts of "work" gets assigned to different processes/threads. The work imbalance could be in computation or communication or both.
- Load imbalance = max_load / mean_load. Max_load is the largest load of any process. Mean_load is the average load across all processes. 1 means that there's no imbalance while greater values means there's more imbalance. The goal is to minimize the number by bringing the maximum load as close to average as possible.
- Load imbalance can slow down all processes.
- The first step in load balancing is determining its necessity. This can be done using profiling tools or through timers.
- The second step is to determine when and how to load balance. If the load imbalance is static, it can be addressed with static load balancing at program startup. The load balancing is dynamic load balancing, where the load distribution evolves over time. One example is scientific simulations, where weather simulations change over time. The change is dependending on each use case though.
- The third step is to decide what information is needed for load balancing.
- The fourth step is to design a load balancing algorithm, which decides how to move data around. One example is the greedy approach, which takes extra load from the most overloaded processes and assigns the work to the least overloaded processes. However, the design depends on what you want to load balance.

Information gathering for load balancing

- Centralized load balancing is a technique which involves gathering all load information to one process. It has a global view of the data and makes all decisions. Some potential issues are communication bottlenecks (sending all to one process) and load balancing bottlenecks (making one process decide for all other processes, which also make sit not scalable).
- Distributed load balancing is a technique, where every process knows the load of its near neighbors, which is user defined. One issue is that within neighborhoods, there is balancing, but across different neighborhoods, there may be imbalance.
- Hybrid/hierarchical load balancing is the best of both worlds. Hybrid scheme requires combining centralized with distributed load balancing. Balancing is done in each clique/group. Then group leaders for each neighborhood sends to a leader process. The leader processes then help fix imbalance across neighborhoods/groups.

Load balancing algorithms

- The input is the current amount of work assigned to each process.
- The output is new assignments of work units to different processes.
- The goals are to bring maximum load close to average as possible and minimize the amount of data migration.

- There is overhead of load balancing, which as figuring out how to balance data and then migrating the data. It's necessary to make sure the benefits outweigh the overhead.
- The secondary goals of this algorithm are to balance and potentially reduce communication load and keep the time for doing load balancing to a minimum.

Examples of static load balancing

- With a 2D stencil, for example, with the game of life, every cell does the same amount of work but needs to decide how many rows to provide to each process.
- Another example is space-filling curves. This algorithm is used to do linear ordering of higher dimensional space into 1 dimensional space. For example, objects nearby in 2D space will be nearby in 1D space
- Another example is orthogonal recursive bisection (ORB). This algorithm partitions data until each partition has the same amount of data (for example, same number of particles in cosmology). It's best for non-uniform data.

Simple greedy strategy:

- The greedy strategy is to sort all processes by their load.
- Then, take some load from heavily overloaded processes and move them to processes with most lightly loaded processes.

Work Stealing:

- Any time a process is out of work (no items in their own queue), it steals work from other processes' queue.

Other considerations:

- Communication-aware load balancing tries to move work to processes that this current process communicates with frequently as opposed to communicating with far away processes,
- Network topology-aware load balancing takes into account how different nodes are connected to one another in order to minimize some metrics like number of hops, average link load, etc.