CMSC416: Introduction to Parallel Computing

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

Announcements:

- For bug in project, use largest size 1920 x 1080 and 100, should make output file.
- Assignment 5 is only for 616 students, extra credit for 416 students

Task-based Prog. Models and Charm ++
Starting on Slide 15/20 - Hello World: Hello Class

Proxy Class:

- User does not need to know where the objects are placed
- Charm runtime knows where the objects are and it keeps track of these through the proxy class
- Using proxy we can invoke remote method calls of objects and maintain consistency

Broadcast, barrier, and reduction:

- One element of a chare array can call another element of a chare array
- chareProxy.entryMethod() is going to be called on all elements of chare array
- Charm is asynchronous so when you call a remote method it is not a blocking call, it is enqueued in the scheduler queue
- If you assume synchronous there will be bugs, remember things are asynchronous
- contribute() is a barrier call; if you call it from every element in a chare array it is a reduction
- Barrier will be blocking to make sure all elements are ran
- Reduction is a barrier with arguments- first argument is bytes, second is pointer to buffer, third is reducer type

Callback for reduction:

- If you call contribute then everyone does reduction, we get the result by setting a callback object
- Specify some function you want to be called when reduction is complete, also specify what the chare elements proxy that you want to call the function on are

2D Stencil in Charm++

- Instead of thinking about mpi processes think about charm objects
- Stencil is same as game of life; 2D board with rows and columns; want to parallelize

- Create a 2D chare array; array[2D] Stencil; assign a sub block to each element of chare array
- We assign the stencil computation to the corresponding index in the array
- For communication do remote method invocation; create a function with the data we want to send as an argument and we will call it on thisProxy[i, j+1] and so on

Load Balancing

Load imbalance:

- When unequal amounts of work get assigned to different processes
- Even if its same amount of work but more communication that is still imbalance
- Imbalance is bad because processes with extra work can slow down all other processes and hold up everything else
- Often times if program is spending extra time in barrier or reduce it is because of imbalance
- Load imbalance = max_load/mean_load
- If load imbalance is closer to 1 the less imbalance we have, further away from 1 then we have more
- We can fix load imbalance in a couple of ways:
    - Bring the maximum load as close to mean as possible
    - First determine if load balancing is needed by collecting performance data
    - If the metric is greater than 1 then we need to balance our code
    - We then need to decide where and when to do load balancing; in charm we can migrate chare objects
    - Whenever we migrate work that is on overhead so there is a cost to balancing
    - Decide what information we need to do balancing, i.e how much work is assigned to each process, do analytical modeling of code
    - Design a load balancing algorithm which decides how to move load around; greedy strategy take work from most assigned process and give to least worked process

Is load balancing needed?

- We need the distribution of work across processes; can collect through analytical tools

When/how often to load balance?

- Static load balancing is done in start of program; good when load imbalance is not dynamic
- Sometimes use first run to collect data and in second run do balancing
- Dynamic load balancing is when the differences change between each run
- How often depends on how much overhead load balancing takes

Information gathering for load balancing:

- Gather all information at one process to have a global view of data - centralized load balancing
    - Might create bottleneck
- Each process knows the load of an x number of neighbors - distributed load balancing
    - Lots of communication
    - We only have a local view of our neighborhood which can create imbalance across neighborhoods

Hierarchical load balancing:

- Bottom of tree is individual processes that we need to balance
- Arrange them into smaller groups to make it easier to balance
- Within each group do a centralized load balancing
- All the group leaders in each group also send total group load to even higher level
- This prevents neighborhood imbalances while also using the pros of centralized load balancing
- Everyone does balancing in their own neighborhood then we balance across neighborhoods

What information is used for load balancing:

- Computation load is the most common metric to record
- Keep track of how large the computations are, i.e multiplying 2 matrices together
- Communication load between processes
- If there is issue we can change our code
- Issue with communication graph then we move code to processes with minimal communication
- We can use communication graph to figure out which process communicate the most

Load balancing algorithms:

- A typical load balancing algorithm has input of amount of work assigned to each process
- Output is new assignment to different processes to balance them
- Main goals are to bring the metric closer to 1 by bringing maximum load closer to average load and minimize amount of data migration
- Secondary goals are to balance communication loads and minimize time spent on load balancing

Examples of static load balancing:

- When we have 2D stencil we statically divide the board among multiple processes

- Every cell in array was doing same amount of work, we just decided size of rows and columns
- Space filling curves is used to do linear ordering of higher dimensional space into one dimensional space
- Cosmology or n body problem a nice way to sectionalize is to use orthogonal recursive bisection
- Divide space into two so left and right side have same number of particles and then top and bottom have same number and keep going
- Helps when there is no order among the data

Simple greedy strategy:
- Sort all processes by their load so max is first
- Assign some load from the heaviest loaded process and assign to the lightly loaded process
- Only assign enough work to bring the lightest loaded to the average

Work stealing:
- Strategy to do load balancing
- Decentralized strategy - processes steal work from their neighbors when they are sitting empty
- Each process has a queue of items to work on
- When a processes queue is empty it looks at neighbors queues and steals work

Other considerations:
- In addition to balancing compute load also balance communication
- Communication aware load balancing - move work to processes that a certain process communicates with frequently
- Network topology aware load balancing - take into account how nodes are connected to optimize which nodes are moving work across each other
- Reduce time to communicate work from far away nodes

Announcement:
Issue with vs code - there is a piazza post - do not use ssh plugin in vs code