Discussion 10

Open MPI, Caches

Aditya Balasubramanian

`aditbala [at] berkeley [dot] edu`

Slides available at `teaching.aditbala.com`

Announcements 

Agenda

- OpenMPI
- Caches

Credits to Rosalie Fang and Andrew Liu

OpenMPI

Credits to Rosalie Fang and Andrew Liu

OpenMPI

- Run one program on multiple processes at once
- Threads
  - share memory
  - Hive Mind, multiple things being done at same time, still same entity
- Processes
  - Processes have distinct memory spaces and each run a copy of the program on different nodes
  - group of people doing work independently
- Manager-worker approach where a manager will assign tasks to different processes when processes finish executing

OpenMPI general workflow

```
int main(int argc, char** argv) {
        …
        // set up OpenMPI
        if (manager process) {
        // manager node code
}
        else {
        // worker node code
}
        // terminate OpenMPI
}
```

Credits to Rosalie Fang and Andrew Liu

OpenMPI Setup Procedure

```
int main(int argc, char** argv) {
        …
        MPI_Init(&argc, &argv); // initialize
        // get process ID of this process and total num of processes
        int procID, totalProcs;
        MPI_Comm_size(MPI_COMM_WORLD, &totalProcs);
        MPI_Comm_rank(MPI_COMM_WORLD, &procID);
        if (procID == 0) {
        // manager node code
}
        else {
        // worker node code
}
        MPI_Finalize();
}
```

Credits to Rosalie Fang and Andrew Liu

OpenMPI Manager Node Pseudo Code

```
While there's work to do:
    Wait until a worker is ready for work (recv from all)
    Find the next task to do
    Send to the worker what task to do
Repeat #Worker times:
    Wait until a worker is ready for work (recv from all)
    Send to the worker "All work done"
```

Credits to Rosalie Fang and Andrew Liu

OpenMPI Manager Node Code

```
if (procID == 0) {
    // manager node code
    MPI_Status status;
    ____ message;
    while there are more tasks {
        MPI_Recv(&message, <count>, <datatype>, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
        int sourceProc = status.MPI_SOURCE;
        // assign next task
        message = _______;
        MPI_Send(&message, <count>, <datatype>, sourceProc, 0, MPI_COMM_WORLD);
    }
}
```

Credits to Rosalie Fang and Andrew Liu

OpenMPI Worker Node Pseudo Code

```
While True:
    Send to the manager "I'm ready for more work"
    Receive message from manager
    If message is "Here's more work":
        Do the work
    Else if message is "All work done":
        break
```

OpenMPI Worker Node Code

```
else {
    // worker node code
    _____ message;
    while (true) {
        // request more work
        message = READY;
        MPI_Send(&message, <count>, <datatype>, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        // receive message from manager
        MPI_Recv(&message, <count>, <datatype>, 0, 0, MPI_COMM_WORLD);
        if (message == TERMINATE) break;
        // do work
    }
}
```

Credits to Rosalie Fang and Andrew Liu

Caches

Credits to Rosalie Fang and Andrew Liu

Caches

- The closer we get to the CPU, the faster we can load data
- Registers
  - On the CPU, blazingly fast, super tiny
- Memory
  - On the motherboard, still pretty fast, pretty big
- Between them, we have the Cache
  - On the CPU, faster than memory, small-ish
- We turn memory addresses into TIO bits to place it into the cache

Data that's close

- Spatial Locality
  - Close with respect to location in memory
  - Data that “lives” +/- a certain amount from the accessed address might be accessed soon
- Temporal Locality
  - Close with respect to time
  - Data that’s accessed recently might be accessed soon again

Cache Terminology

- Cache line
  - a chunk of data that is loaded into the cache from main memory upon an access (also known as block)
- Set
  - groups of cache lines to form a larger group indexed by the index
- Associativity
  - number of cache lines needed to form a set (4-way associative cache means each set contains 4 blocks)

TIO Breakdown

tag	index	offset

- Tag
  - log (Memory size in bytes) - I - O
- Index
  - log (Number of blocks / associativity) = log2(Number of indices)
- Offset
  - log (Size of block in bytes)

Thank you!

Credits to Rosalie Fang and Andrew Liu