Shared Memory Architectures
Programming and Synchronization

Discuss paper on Cosmic Cube (message passing)

Today's Outline

- Message passing review
- Cosmic Cube discussion
  > Message passing machine
- Shared memory model
  > Communication
  > Synchronization
- Ultracomputer/RP3 discussion
  > Shared memory machine
- Shared memory programming
- Fine grain versus coarse grain parallelism
- How do caches change things
  > Improve and complicate!
  > Beehive
Review
Message Passing Parallel Programming Model

Communication: via messages
Synchronization: via messages

Message
Private Memory
Process A
Private Memory
Process B
E.g., MPI

How to Receive a Message
Beehive uses polling

Core 0
local cache
Id
stio
Wait in loop if no msg

Core 1
local cache
Id
local cache
local cache
local cache
Message
Network
The Cosmic Cube
The Earliest Message Passing Machine

- Direct network - hypercube (details later in course)
- Private memories
- Message sends by calling into OS
- Routing in software
- Sequential programming on each processor & message send/receive (much like Beehive)
- Hide comm latency by switching processes
- Simple hardware

Discuss paper
Next, Recall, Shared Memory Parallel Programming Model

Blackboard captures state

Shared memory

Threads

E.g., pthreads

Communication: via shared memory

Synchronization: shared memory locks

Ultracomputer Design

- Indirect network - Omega network (details later in course)
- Shared memory machine
- Communication/synchronization through shared memory
- Hardware routing of memory requests
- No latency hiding - wait for memory request

Concept built as IBM RP3 machine (we will see this later)
Popularized SPMD Programming
(Single-program multiple-data)

**Parallel section**
- P_A
- R_A
- P_B
- S_A
- P_C
- P_D

**Replicate section**
- Glob_C=5

**Serial section**
- Glob_Z=Glob_Z+1

Annotate sequential programs

- 9 -

Quick Detour
Barrier Synchronization

A process that executes a barrier must wait until all other processes have executed their barrier.

Discuss how to do barrier on Beehive using message passing

- 10 -
You should learn this!

Most parallel programs written for commodity multicores use this style (all commodity multicores happen to be shared memory machines!)*

All processors run a copy of the same program (commonly a slightly modified version of the sequential program)

Processor-specific behavior created using unique processor IDs

Also need to introduce synchronization as necessary

Let's do a simple example to build intuition

*Note that, in general, SPMD style of programming can be applied to either shared memory or message passing machines

```
#define LENGTH 1000000
int a[LENGTH], b[LENGTH], c[LENGTH];
int i=0;

main()
{
    /* Initializations */
    /* read in the two vectors */
    i = 0;
    while (i < LENGTH)
    {
        c[i] = a[i] + b[i];
        i = i + 1;
    }

    /* output the answer */
}
```

Sequential addition of two vectors
Parallel SPMD Version

Assume Ultracomputer model. Assume no caches, single word memory access.

```c
#define LENGTH 1000000
int a[LENGTH], b[LENGTH], c[LENGTH];
int i=0;
int L=0;

main()
{
    /* create parallel processes */
    ...
    /* Initializations */
    if (myPId == 0) ...
    /* read in the two vectors */
    if (myPID == 0) ...
    int myi;
    myi = getwork();
    while (i < LENGTH)
    {
        c[myi] = a[myi] + b[myi];
        myi = getwork();
    }
    /* output the answer */
    if (myPID == 0) ...
}

int getwork()
{
    getlock();
    i = i + 1; /* increment is atomic */
    releaselock();
    return(i);
}
```

Sequential addition of two vectors

## Pure Shared Memory

No caches, single word reads/writes

Assume each process runs the rest of the same program
Only process 0 runs this
Get an index on which to work. Example of self scheduling
Lock: Using Test-and-Set Instruction

Example of a "spin lock"

```c
void getlock()
{
    while (T&S(L) == 1); /* loop till you get the lock */
    return();
}

void releaselock()
{
    L = 0; /* release the lock */
    return();
}
```

How to implement T&S in HW? In SW?

Test-and-Set Instruction Implementation

```c
void getlock()
{
    while (T&S(L) == 1); /* loop till you get the lock */
    return();
}

void releaselock()
{
    L = 0; /* release the lock */
    return();
}
```

Problem: Lock is held for load-store cycle!
Locks out even the lock releaser.

Can we do better?
Ideas?

How to implement T&S in SW? Dekker's Alg.
void getlock()
{
    while (T&S(L) == 1) {} /* loop till you get the lock */
    return();
}

void releaselock()
{
    L = 0; /* release the lock */
    return();
}

Any other problems?

// introduce backoff here
while (L == 1) {} /* loop till you get the lock */
return();

void releaselock()
{
    L = 0; /* release the lock */
    return();
}

We engineers love to optimize!
So, getting a work item is not so cheap after all, is it?
Any ideas?

Coarse grain parallelism (versus fine grain parallelism):
Get a block of 4 or 16 or more indices each time to amortize the overhead of locking

Getwork() grabs an index to a row (e.g.)
Synchronization as before
Loads and stores to shared array
Finish row. How do I know when to start next jacobi iteration?
Use barrier after you finish your row
Lots of communication over the network
Pure Shared Memory

32-bit energy costs in 40nm

DRAM read: \(~1000\text{pJ}\)

Add: \(~1\text{pJ}\)

Send 1mm distance: \(~10\text{pJ}\)

Register read: \(~1\text{pJ}\)

Ideas?

Caches!

Cache read (small L1): \(~10\text{pJ}\)