ECE 574 – Cluster Computing Lecture 11

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

11am, Barrows 133

20 February 2024

Announcements

- More info on the upcoming midterm (February 29th, week from Thursday) will be forthcoming.

- The final project info will be posted to the course website, more info on that soon as well

Shared Memory vs Distributed Systems

- So far we've mostly been talking about shared memory systems

Shared Memory Systems

- Has one copy of OS
- All programs see one unified memory space.

Shared Memory Limits

- OpenMP is nice to use. But what if your problem won't fit on a single machine?
- How big can a shared-memory machine be?

Biggest Shared Memory Systems

- SGI UV systems at least 4096 cores and 16TB running one Linux image

  http://www.techeye.net/hardware-2/sgi-builds-pittsburgh-4096-processor-core-16tb-shared-memory-su

- Digression about SGI
- Use special NUMA-Linux architecture to spread cache coherence across multiple machines.
- Origin TM and Onyx2 TM Theory of Operations Manual

Linux Scaling Limitations

- Linux currently maxes out to 4096 cores or so
- Somewhat dated "Scaling Linux to the Extreme" paper problems: cache contention could bring machine to halt (if a global idle counter, each thread trying to increment once a second)
  lock contention, page cache
- What are the challenges? Locking contention?

Eventually you hit the limit

What's the alternative?

Distributed System

- Communicate over a network
- Many systems each with own memory, communicate via Message passing
- Each node has own copy of operating system
- How do they communicate?

Networks – Physical Layer

- Copper Wire vs Fiber Optics
- Which is faster? Which is lower latency? Which is cheaper?
- Which can go further distance? Which can bend around corners?
- Which is lower power?
- Latency vs Bandwidth
- Speed of Light in each?

Networks – Copper

- Twisted Pair
- Fancy connectors

Networks – Fiber

- Single-mode
  - More expensive?
  - Only one frequency?
- Multi-mode
  - Shorter distance

Network Topology

- Packet-switching vs bus

- Ring, mesh, star, line, tree, fat-tree fully connected

- Cube, hypercube

- Mesh networks and routing

- Routing. Fully connected? Crossbar?

Network Types – Top 500 Nov 2022

This is constantly changing, hard to keep up year to year

interconnect	#
100GB Ethernet	70
25GB Ethernet	70
10GB Ethernet	62
Infiniband EDR	35
Infiniband HDR	35
Intel Omnipath	34
Mellanox HDR Infiniband	26
Aries	25
Infiniband HDR 100	20
Slingshot-10	17
. . .	-

Network Types – Ethernet

- Low-end (10/100/1GB/2.5GB/10GB) commodity
- Regular computers support it
- Cat-5/Cat-6 cables relatively cheap?
- How many ports on a switch? What do you do when you need more?
- The faster schemes are complex, use a lot of power

Network Types – High-end Ethernet

- 25GB
  - Introduced as 10GB not fast enough, but 40GB expensive
  - SFP28 transceivers, both optical and copper
- 40GB
  - 802.3ba – 2010 – 1m backplane, 100m multi-mode fiber, 10km single-mode fiber
  - 802.3bg –
  - 802.3bq – 2016 – 4-pair balanced twisted pair 30m

- ○ QSFP+ transceivers, quad small form-factor pluggable, four 10GB lanes
- 100GB
  - ○ 4 lanes of 25GB
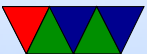
Network Types – Infiniband

- low latency, most common in supercomputers
- Often a step ahead of ethernet (but more expensive)
- QSFP+ 3m 40Gb/s cable $30?
- copper or fiber, GB/s (links aggregated, 4x common)
-

	SDR	DDR	QDR	FDR-10	FDR	EDR	HDR	NDR	XDR	GDR
4x	8	16	32	40	54	100	200	400	800	1600
12x	24	48	96	120	163	300	600	1200	2400	4800

- Note XDR/GDR just proposed (2023)
- RDMA, Ethernet over Infiniband

Network Types – HPE Slingshot

- HPE Cray Supercomputers
- Up to 200 Gb/s
- Handles lots of small packets
- 1.2 billion packets/second/port
- High Radix(?) 64-port 12.8TB/s switch
- Scale to 250k ports with max of 3 hops
- Price: "get quote" which means, a lot

Network Types – Fugaku Paper

- Hooking together 10k+ nodes?
- Mesh or 3d-torus?
- Blue-Genel L partition things so jobs run on subset partitions that are also 3d-torus. Issues if a node goes down
- Cray XT mesh and job can be scattered throughout, but an have higher latency
- Instead, 6-dimensional torus
- Tofu interconnect for K computer (torus fusion)

Network Types – Older/Other

- Cray Gemini – Mesh/torus – 64Gb/s
- Fibrechannel
- Older: custom, Myrinet

Remote DMA

- Zero copy of data from network card directly into memory of remote system
- Can lead to really fast MPI transactions
- Security?

Review – Setting up a Cluster

- Setting up machines (power/cooling)
- Installing operating system
- Setting up network
- Setting up head node?
- Setting up authentication
- Setting up shared storage (network file system)
- Setting up MPI
- Optimizing MPI (Infiniband?)

Programming a distributed System

Could you Open Code by Hand?

- Sort of how you can use pthread directly?
  - Need to have copy of executable on each node (network filesytem helps)
  - Need to launch executable on remote system (ssh can do this) authentication is a challenge
  - Then write custom network code to open sockets and communicate among them all
- Network code is a pain
- Just crying out for abstraction

Distributed Programming

- Parallel Virtual Machine (PVM) early implementation
- MPI was competitor
- MPI won somehow? Other proposals never took off?

Message Passing Interface (MPI)

- Abstraction for sending data between separate processes
- The interesting part is these processes can be on different nodes in a cluster
- You can put together an array of 100 floats, and say "send this to process Y" and like magic it appears there.
- You don't have to worry about setting up networks, network addresses or ports, or decoding streams of bytes
- The code is also portable, so can move to another cluster without a total rewrite

MPI (Message Passing Interface) History

- MPI 1.0 – 1994. MPI 3.0 – 2012
- MPI 1.2 widely used. MPI2.0 is complicated and adoption not as high as it could be.
- MPICH – CH stands for Chameleon – Argonne and Mississippi State
- MVAPICH – from Ohio State, based on MPICH
- OpenMPI – merger of 3 MPI implementations:
    - FT-MPI from the University of Tennessee,
    - LA-MPI from Los Alamos National Laboratory, and

- ○ LAM/MPI from Indiana University
- Any other options? PVM was a predecessor
- Python Bindings, Java bindings, Matlab

MPI

Some references

`https://hpc-tutorials.llnl.gov/mpi/`

`http://moss.csc.ncsu.edu/~mueller/cluster/mpi.guide.pdf`

Writing MPI code

- `#include "mpi.h"`
- Over 430 routines
- `MPI_Init()` called before anything else
  can pass in command line arguments that get sent to all, but this isn't always supported
- `MPI_Finalize()` at the end
- Error handling – most errors just abort

Compiling/Running MPI Code

- use `mpicc` to compile
  gcc or other compiler underneath, just sets up includes
  and libraries for you.
- `mpirun -n 4 ./test_mpi`
- `mpiexec` is possibly preferred over mpirun
- tools like slurm can handle running things for you

Communicators

- You can specify communicator groups, and only send messages to specific groups.

- `MPI_COMM_WORLD` is the default, means all processes.

Rank

- Rank is the process number
- You can find out a processes rank

```
MPI_Comm_rank(MPI_Comm comm, int *rank);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

- You can find the number of ranks (processes):

```
MPI_Comm_size(MPI_Comm comm, int *size);
```

This gets all the processes in a communicator, but if you use `MPI_COMM_WORLD` it gets all of them

How does Rank relate to Threads?

- Each process is given a rank number, 0 .. N
- The processes can be on different machines/nodes, but don't have to be
- Ranks are *not* threads. They are processess. They do not share memory and communicate via distributed messages.
- To confuse things, there no reason you can't program things so the ranks can contain multiple threads. MPI is not involved with this and the threads would not get

separate rank numbers

Other Possibly Useful Commands

- `MPI_Get_processor_name()` – gets name of machine running on

- `MPI_Get_version()` – gets MPI version

- `MPI_Initialized()` – see if MPI has been initialized (useful if you have optional modules)

Error Handling

- `MPI_SUCCESS` means good result

- By default it aborts if any sort of error

- Can override this

Timing

- `MPI_Wtime();` wallclock time in double floating point. For PAPI-like measurements

- `MPI_Wtick();`

Point to Point Operations

- Buffering – what happens if we do a send but receiving side not ready?

- Blocking – blocking calls returns after it is safe to modify your send buffer. Not necessarily mean it has been sent, may just have been buffered to send. Blocking receive means only returns when all data received

- Non-blocking – return immediately. Not safe to change buffers until you know it is finished. Wait routines for

this.

- Order – messages will not overtake each other. Send $\#1$ and $\#2$ to same receive, $\#1$ will be received first

- Fairness – no guarantee of fairness. Process 1 and 2 both send to same receive on 3. No guarantee which one is received

MPI_Send – send data block

- blocking – `MPI_Send(buffer,count,type,dest,tag,comm);`
- non-block – `MPI_Isend(buffer,count,type,dest,tag,comm,request);`
- Parameters
  - buffer – pointer to the data buffer
  - count – number of items to send
  - type – MPI predefines a bunch. `MPI_CHAR`, `MPI_INT`, `MPI_LONG`, `MPI_DOUBLE`, etc.
    can also create own complex data types
  - destination – rank to send it to

○ Tag – arbitrary integer uniquely identifying message. Can pick yourself. 0-32767 guaranteed, can be higher.

○ Communicator – can specify subgroups. Usually use `MPI_COMM_WORLD`

○ request – on non-blocking this is a handle to the request that can be queried later to see that status

MPI_Recv – receive data block

- block – `MPI_Recv(buffer,count,type,source,tag,comm,status);`
- non-block – `MPI_Irecv(buffer,count,type,source,tag,comm,request);`
- Parameters
  - buffer – pointer to the data buffer
  - count – number of items to send
  - type – MPI predefines again
  - source – rank to receive from. Also can be `MPI_ANY_SOURCE`
  - Tag – arbitrary integer uniquely identifying message.

○ Communicator – can specify subgroups. Usually use MPI_COMM_WORLD

○ status – status of the receive, a struct in C
   has the source, tag, and info on bytes received

○ request – on non-blocking this is a handle to the request that can be queried later to see that status

Handling non-blocking

- `MPI_Wait()` — wait until done
- `MPI_Test()` — test to see if done
- Use the `request` vaule in conjunction with them

Fancier blocking send/receives

- Lots, with various type of blocking and buffer attaching and synchronous/asynchronous

- For example `MPI_Ssend()` will wait until it receives confirmation that the remote process got the data

Sample code

```c
/* MPI Send Example */
#include <stdio.h>
#include "mpi.h"

#define ARRAYSIZE 1024*1024

int main(int argc, char **argv) {

    int numtasks, rank;
    int result,i;
    int A[ARRAYSIZE];
    MPI_Status Stat;
    int count;

    result = MPI_Init(&argc,&argv);
    if (result != MPI_SUCCESS) {
        printf ("Error starting MPI program!.\n");
        MPI_Abort(MPI_COMM_WORLD, result);
    }

    MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
```

```c
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

printf("Number of tasks= %d My rank= %d\n",
    numtasks,rank);

if (rank==0) {
    /* Initialize Array */
    printf("Initializing array\n");
    for(i=0;i<ARRAYSIZE;i++) {
        A[i]=1;
    }

    for(i=1;i<numtasks;i++) {
        printf("Sending %d ints to %d\n",
            ARRAYSIZE,i);
        result = MPI_Send(A, /* buffer */
                    ARRAYSIZE, /* count */
                    MPI_INT, /* type */
                    i,          /* destination */
                  13,     /* tag */
                   MPI_COMM_WORLD);
    }
}
else {
```

```
        result = MPI_Recv(A, /* buffer */
                ARRAYSIZE, /* count */
                MPI_INT, /* type */
                0,   /* source */
                13,       /* tag */
                MPI_COMM_WORLD,
                &Stat);
        result = MPI_Get_count(&Stat, MPI_INT, &count);
        printf("\tTask %d: Received %d ints from task %d with tag %d \n",
            rank, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
}

int sum=0,remote_sum=0;

for(i=rank*(ARRAYSIZE/numtasks);i<(rank+1)*(ARRAYSIZE/numtasks);i++) {
    sum+=A[i];
}

if (rank==0) {

    for(i=1;i<numtasks;i++) {
        result = MPI_Recv(&remote_sum, /* buffer */
                1, /* count */
                MPI_INT, /* type */
```
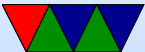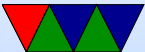
```c
                MPI_ANY_SOURCE , /* source */
                13,       /* tag */
                MPI_COMM_WORLD ,
                &Stat );
        result = MPI_Get_count (& Stat , MPI_INT , &count );
        printf ("\tTask %d: (%d) Received %d int from task %d with tag %d \n",
            rank , remote_sum , count , Stat.MPI_SOURCE , Stat.MPI_TAG );
        sum += remote_sum ;

    }
    printf ("Total: %d\n",sum );

}
else {
    printf ("\tRank %d Sending %d\n",rank ,sum );
    result = MPI_Send (& sum , /* buffer */
            1, /* count */
            MPI_INT , /* type */
            0,      /* destination */
            13,     /* tag */
            MPI_COMM_WORLD );

}
MPI_Finalize ();
```

}