ECE 574 – Cluster Computing
Lecture 6

Vince Weaver

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

vincent.weaver@maine.edu

11 February 2025

Announcements

- Lecture delayed by last Thursday's snowstorm
- HW#3 was posted, deadline extended to Monday
- C coding. Not too horrible, (not as bad as ECE435) but some array manipulation.

Workload for future Homeworks

- Before we can write parallel code, we need some serial code as an example
- Matrix multiply is typical, but boring
- What else can we use that's embarrassingly parallel, but interesting?

Convolution

- https://en.wikipedia.org/wiki/Kernel_%28image_processing%29
- Specifically 2-D convolution
- Widely used in image processing
- Walk over every pixel in an image, convolving a matrix over it. The new value is based on some combination of the surrounding pixels.
- Usually a 3x3 grid, but can be larger

Common Convolution Matrices

- Identity $= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- Blur $= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ (need to normalize)
- Sharpen $= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
- Emboss $= \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$
- Sobel (edge detection) $= \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$

Loading Graphics into a C array

- We'll use libjpeg to do this
- Use to decompress image into an array of RGB pixels
- How JPEG works is a bit beyond this class:
  - Lossy compression format Discrete Cosine Transforms
  - It is good at compressing pictures
  - It is lossy, but deterministic, so we can use it for output in this class
- We also use libjpeg to convert the output array back to an image file

What does a framebuffer look like?

- Depends on many things
- Bits-per-pixel, 1bpp, 2bpp, 4bpp, 8bpp, 15bpp, 16bpp, 24bpp, 32bpp
- We will be using 24bpp, with RGB each being one byte
- Our image is a 3D array (x,y,color), but that's hard to do in C (especially when dynamically allocating memory) so we will just do a 1D array

Aside: modern framebuffers are a luxury

- Might seem tricky to get bytes in right place
- On old Apple II system, weird interleaved framebuffer, 14 pixels per 2-bytes, 240p resolution, color clash
- Atari 2600 worse, only 20-bit *total* framebuffer, had to "race the beam" to draw whole screen
- Even more recent, CGA / EGA/ VGA planar for bandwidth reasons, would have to bank switch in multiple planes to draw one pixel
- Famous Mode 13h nice linear array, 8bpp (palette)

One way to implement the convolution

- There are many ways you can implement this, some will be faster than others. The one shown below is definitely not the fastest.
- Below is *pseudo code*. It won't compile, as you won't be able to do the triple array access as pictured, you'll have to access the values as a 1-D array as discussed in class.

```
for(x=1;x<width-1;x++) {
    for(y=1;y<height-1;y++) {
        for(color=0;color<3;color++) {
            sum=0;
            sum+=filter[0][0]*old[x-1][y-1][color];
            sum+=filter[1][0]*old[x][y-1][color];
            sum+=filter[2][0]*old[x+1][y-1][color];
            sum+=filter[0][1]*old[x-1][y][color];
            sum+=filter[1][1]*old[x][y][color];
            sum+=filter[2][1]*old[x+1][y][color];
            sum+=filter[0][2]*old[x-1][y+1][color];
            sum+=filter[1][2]*old[x][y+1][color];
            sum+=filter[2][2]*old[x+1][y+1][color];

            /* Normalize if necessary (not needed for Sobel) */

            /* Saturate if necessary (Make sure stays in 0 to 255 range) */
            (your code here)

            /* Set the new value */
            new[x][y][color]=sum;
        }
    }
}
```

C array access

- `a[x][y][color]` should be done as `a[(y*xsize*3)+(x*3)+color]`
- You might want to write a helper function that does this for you.
- Remember in C that array indexes begin at 0, not 1.
- Why do things this way? You can't use malloc() or calloc() with a[x][y][c] syntax (or you can, but you have to have pointers to points and one malloc per row, it gets complex very quickly). Since we don't know the

size of the image in advance it's easier to do things with a 1D array

Sobel Convolution Notes – Saturating Adds

- For Sobel we do not need to normalize the result, but we do need to saturate
- If the results is greater than 255, set to 255, or if less than zero, set to zero.
- Otherwise will wrap and give odd results.

Sobel Convolution Notes – Image Border

- What do we do for pixels on the edge of the image that don't have surrounding pixels?
- Do you wrap? Assume 0?
- For our code we will only convolve on pixels at least 1 pixel from the border, which results in the edge of the final image being 0 (black)

Sobel Convolution – Combining the Results

- We will find the horizontal edge, (sobel_x), the vertical edge, (sobel_y) and then combine the two
- To combine, for each element square the two results then take the square root (and saturate to 0..255)
- $final[x][y][c] = \sqrt{sobelx[x][y][c]^2 + sobely[x][y][c]^2}$
- Note C has a `sqrt()` function. May need to link against math library to use (`-lm`)
- Also note, you can't use ˆ2 to square things (that's XOR!) either multiply by itself or use `pow()`

How to Optimize

- ROW vs Column Major? FORTRAN vs C? Comes down to using cache in an expected way.
- Loop order? Want to access in cache friendly manner
- Loop unrolling? Avoids branch issues, etc.
- SIMD? Definitely a case where we could load all 4 channels and operate on them at once. Possibly multiple. A bit advanced for this class though.
- Compiler options? Using better compiler? Just use OpenCV?

PAPI Usage Instructions – Setup

- The code will include `papi.h` and link against the library with `-lpapi`
- Initialize with:
  `PAPI_library_init(PAPI_VER_CURRENT);`
  Check the result to see if it matches `PAPI_VER_CURRENT`
- All other functions should return `PAPI_OK` if successful.
- If using pthreads need to do:
  `PAPI_thread_init(pthread_self);`

PAPI Usage Instructions – Creating Eventsets

- Eventsets are just integers

  `int eventset=PAPI_NULL;`
- Gathered results are typically 64-bit integers

  `long long values[NUM];`

  Where NUM is the number of events you are measuring at once.
- Create an eventset:

  `PAPI_create_eventset(&eventset);`

- Available events can be seen with the `papi_avail` and `papi_native_avail` commands.
- Add an event. You can run multiple times to add multiple events.

  ```
  PAPI_add_named_event(eventset,"PAPI_TOT_INS");
  ```

PAPI Usage Instructions – Instrumenting the Code

- Before the code of interest do a `PAPI_start(eventset);`
- Afterward do a `PAPI_stop(eventset,values);` and you can print the value or save it for later.
- When printing, remember the results are 64 bits. `printf("Result:  %lld",values[0]);`

PAPI Usage Instructions – Debugging

- The functions all return errors, so it's best to check them
- If you don't check for errors, it won't crash, but you might get strange (usually really high) results
- If you get an error returned, you can use `PAPI_strerror()` to look up the meaning

Hints for Debugging

- You don't have to develop on the cluster, but I will test there
  If you run on own machine you'll have to install PAPI which might only be possible on Linux
- If your final results don't look right, you can first try dumping the jpeg of sobelx and sobely and getting those working first

Getting Results off the Server

- How can you view the results?
- You can `scp` locally (port-redirection with scp needs -P2131, note it's a capital P)

```
scp -P2131 ece574-0@weaver-lab.eece.maine.edu:output.jpg .
```

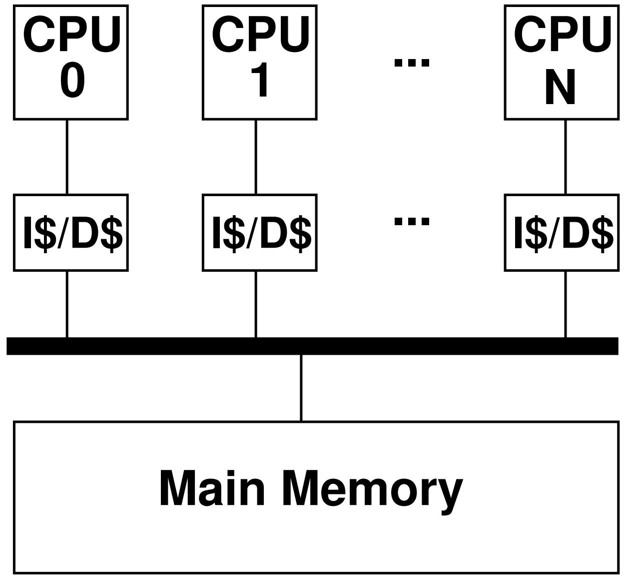
- If you're running X11 graphics on your machine, you can ssh into the server with `-Y` option to forward a graphics viewer like geeqie
- Some GUI scp/sftp clients will let you just double click on images and it will pop them up

More Computer Arch Review

Multicore Systems

- Moore's Law can't make systems faster, so the extra transistors are used for more cores
- Single Package: CMP (Chip-multiprocessor) or SMP (Symmetric-multiprocessor)
- Multi-package: Multiple CMP packages in system.
- These are called "shared memory" systems, one memory all cores write to, all cores can see reads/writes by all

CMP Diagram



Cache Coherency

- How do you handle data being worked on by multiple processors, each with own cache of main memory?
- Cache coherency protocols.
- Many and varied. MESI is a common one
- Directory vs Snoopy

MESI (Modified, Exclusive, Shared, Invalid)

- If just reading, all cores are fine (memory is same) so can live in shared state
- Writing is the problem. Before writing, need to request "exclusive" state, which will invalidate the copies on all other cores
- Once exclusive, can write which leads to "modified"
- Once write back to memory, can return to "shared" state
- This is all done magically in hardware so in theory software doesn't need to worry about it

Barriers and Ordering

- On modern out-of-order execution, memory accesses can happen out-of-order `https://arangodb.com/2021/02/cpp-memory-model-migrating-from-x86-to-arm/`
- Sequential consistency – all happen in order
- Strong consistency – stores
- Weak consistency – can be arbitrarily reordered, only barriers protect you

- A memory barrier instruction makes sure all previous loads/stores finish before moving on

- Most important for things like locks, as well as memory-mapped I/O

Ordering Example

```
Core1:                   Core2:
y1=0                     x1=y1
y2=0                     x2=y2
y1=3
y2=4
```

What values of x1 and x2 can you get?

```
Strong:
        x1=0,x2=0
        x1=3,x2=0
        x1=3,x2=4
Weak:
        x1=0,x2=4
```

Hardware Multi-Threading

- Idea is to re-use a pipeline to execute multiple threads at once, *without* fully replicating the entire CPU (so less than multicore)
- You will have to replicate some things (program counter for each, etc)
- Usually they appear to the CPU as full separate processors even though they are not.
- Various ways to do this:

- Fine-grained – rotate threads every cycle
- Coarse-grained – rotate threads only if long latency event happens (cache miss)
- Simultaneous – issue from any combination of threads, to maximize use of pipeline (have to be superscalar)

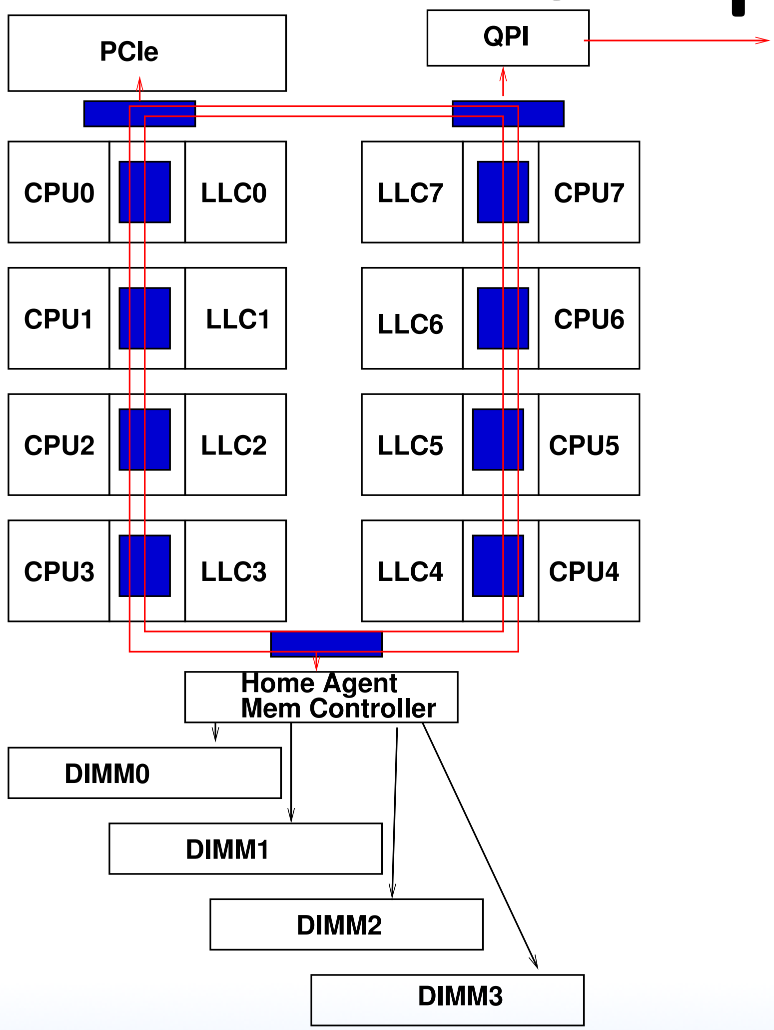
- Why do this? Often on superscalar running only one thread will leave parts idle, try to make use of these.
- Bad side effects?

  Can actually slow down code (especially if both threads trying to use same functional units, also if both using memory heavily as cache is often shared)

- Sometimes see it talked about as SMT (Simultaneous Multithreading), Intel Hyperthreading is more or less the same thing
- Modern security issues, leak info between threads

SMT Diagram

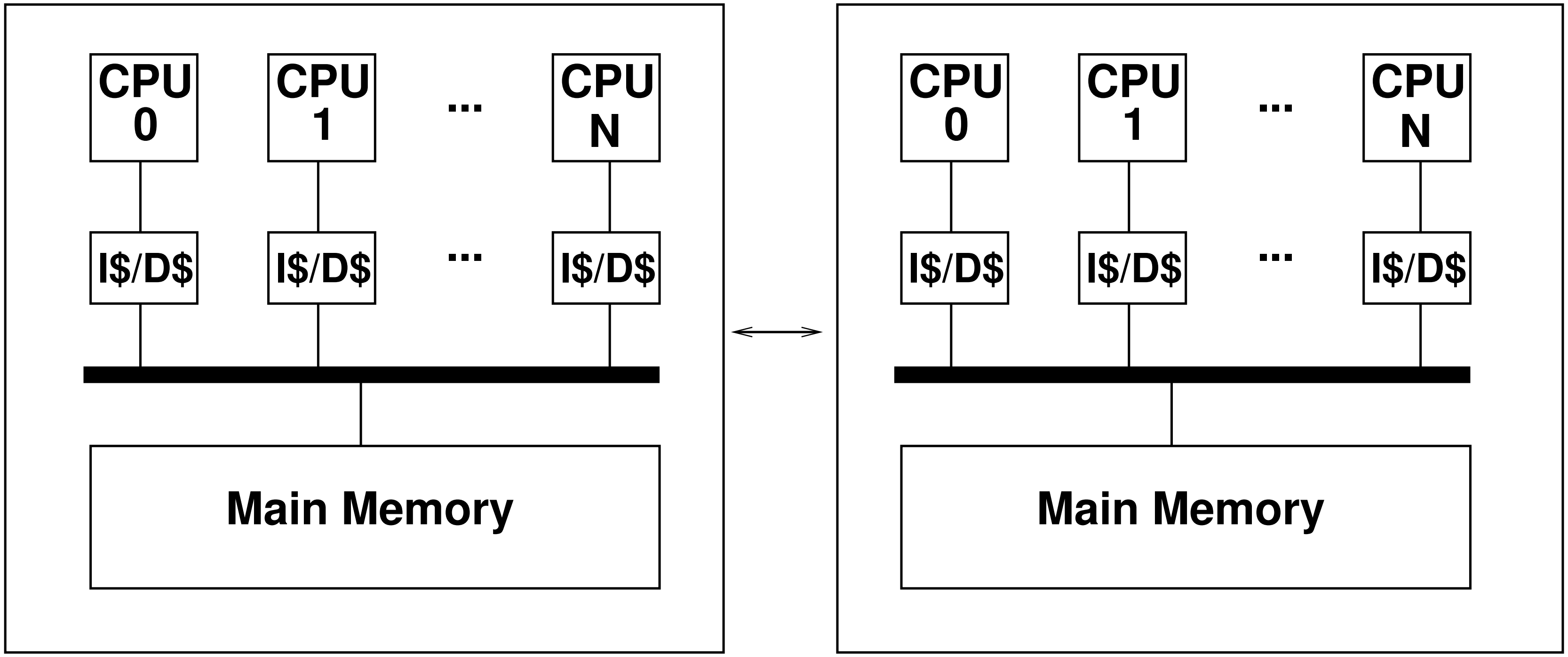
Haswell EP Setup



Non-Uniform Memory Access (NUMA)

- Random Access Memory – arbitrary memory accesses take same amount of time
- Is that true anymore/
- NUMA: some accesses will have to cross to other processors, causing extra delay
- How can you optimize this?

Traditional NUMA Layout



Types of Clusters

- Shared-memory
  - many CPUs, but one shared memory address space.
  - Usually one copy of operating system.
  - When write to memory, all CPUs can see it.
- Distributed
  - Many systems spread across network
  - Each has own memory
  - For other CPUs to see data have to send message across network.

Types of Clusters / Programming

- We'll find shared memory is easier to program
  Biggest ever? 8k SGI machines?

- Larger systems forced to use message-passing