Outline

Schedule

Memory organization

Caching
- Different types of locality
- Cache organization

Cachelab
- Part (a) Building Cache Simulator
- Part (b) Efficient Matrix Transpose
- Bro do you even C
Before we Begin…

Encouraging Female Reverse Engineers

- Contest to reverse engineer malicious software
  - And document it with utmost understanding

Prize

- Ticket to Symposium on Security for Asia Network (Syscan)

Details

- [http://addxorrol.blogspot.de/2013/01/encouraging-female-reverse-engineers.html](http://addxorrol.blogspot.de/2013/01/encouraging-female-reverse-engineers.html)
- Only women can submit (sorry guys)
- One of the judges went to CMU!
- Deadline: 24th of March 2013, 23:59 GMT+1
Outline

Schedule
Memory organization
Caching
  ■ Different types of locality
  ■ Cache organization
Cachelab
  ■ Part (a) Building Cache Simulator
  ■ Part (b) Efficient Matrix Transpose
  ■ Bro do you even C
Schedule

Buflab
- Due Tomorrow, midnight

Cachelab
- Out tomorrow!
- Due Thursday, February 28
- 10 days
Outline

Schedule

Memory organization

Caching

- Different types of locality
- Cache organization

Cachelab

- Part (a) Building Cache Simulator
- Part (b) Efficient Matrix Transpose
- Bro do you even C
Memory Hierarchy

Registers

SRAM

DRAM

Local Secondary storage

Remote Secondary storage

We will discuss this interaction
SRAM vs DRAM tradeoff

**SRAM (cache)**
- Faster (L1 cache: 1 CPU cycle)
- Smaller (Kilobytes (L1) or Megabytes (L2))
- More expensive and “energy-hungry”

**DRAM (main memory)**
- Relatively slower (hundreds of CPU cycles)
- Larger (Gigabytes)
- Cheaper
Outline

Schedule

Memory organization

Caching

- Different types of locality
- Cache organization

Cachelab

- Technical Questions
- Part (a) Building Cache Simulator
- Part (b) Efficient Matrix Transpose
Caching

Temporal locality
- A memory location accessed is likely to be accessed again multiple times in the future.
- After accessing address X in memory, save the bytes in cache for future access.

Spatial locality
- If a location is accessed, then nearby locations are likely to be accessed in the future.
- After accessing address X, save the block of memory around X in cache for future access.
Memory Address

64-bit on shark machines

memory address

tag set index block offset

Block offset: $b$ bits
Set index: $s$ bits
Cache

A cache is a set of $2^s$ cache sets

A cache set is a set of $E$ cache lines
- $E$ is called associativity
- If $E=1$, it is called “direct-mapped”

Each cache line stores a block
- Each block has $2^b$ bytes
Outline

Schedule

Memory organization

Caching

- Different types of locality
- Cache organization

Cachelab

- Part (a) Building Cache Simulator
- Part (b) Efficient Matrix Transpose
- Bro do you even C
Cachelab

Part (a) Building a cache simulator

Part (b) Optimizing matrix transpose

Bro do you even C (helpful C stuff)
Part (a) Cache simulator

A cache simulator is NOT a cache!
- Memory contents NOT stored
- Block offsets are NOT used
- Simply counts hits, misses, and evictions

Your cache simulator need to work for different s, b, E, given at run time.

Use LRU replacement policy
Cache simulator: Hints

A cache is just 2D array of cache lines:

- struct cache_line cache[S][E];
- $S = 2^s$, is the number of sets
- $E$ is associativity

Each cache_line has:

- Valid bit
- Tag
- LRU counter
Part (b) Efficient Matrix Transpose

Matrix Transpose (A $\rightarrow$ B)

Matrix A

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Matrix B

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

How do we optimize this operation using the cache?
Part (b)

Cache:
- You get 1 kilobytes of cache
- Directly mapped (E=1)
- Block size is 32 bytes (b=5)
- There are 32 sets (s=5)

Test Matrices:
- 32 by 32, 64 by 64, 61 by 67
Part (b)

Things you’ll need to know:

- Warnings are errors
- Header files
- Useful functions
Warnings are Errors

Strict compilation flags

Reasons:

- Avoid potential errors that are hard to debug
- Learn good habits from the beginning

Add “-Werror” to your compilation flags
Missing Header Files

Remember to include files that we will be using functions from

If function declaration is missing

- Find corresponding header files
- Use: man <function-name>

Live example

- man 3 getopt
getopt

getopt() automates parsing elements on the unix command line if function declaration is missing

- Typically called in a loop to retrieve arguments
- Its return value is stored in a local variable
- When getopt() returns -1, there are no more options

To use getopt, your program must include the header file unistd.h
getopt

A switch statement is used on the local variable holding the return value from getopt()

- Each command line input case can be taken care of separately
- “optarg” is an important variable – it will point to the value of the option argument

Think about how to handle invalid inputs
Example

```c
int main(int argc, char** argv){
    int opt, x;
    while(-1 != (opt = getopt(argc,argv,"x:")) ){//looping over arguments
        switch(opt) {//determine which argument it's processing
            case 'x':
                x=atoi(optarg);
                break;
            default:
                printf("wrong argument\n");
        }
    }
}
```

Suppose the program executable was called “foo”. Then we would call “./foo -x 1 “ to pass the value 1 to variable x.
fscanf

The `fscanf()` function is just like `scanf()` except it can specify a stream to read from (`scanf` always reads from `stdin`)

- **Parameters:**
  - file pointer,
  - format string with information on how to read file,
  - the rest are pointers to variables to storing data from file

- Typically want to use this function in a loop until it hits the end of file

`fscanf` will be useful in reading from the trace files
Example

FILE * pFile; //pointer to FILE object

pFile = fopen ("myfile.txt","r"); //open file for reading

int x, y;
char c;
//read two ints and a char from file

while(fscanf(pFile, "%d %d %c", &x, &y, &c) > 0){
  // Do stuff
}

fclose(pFile); //remember to close file when done
Use malloc this to allocate memory on the heap

Always free what you malloc, otherwise may get memory leak

- Some_pointer_you_malloced = malloc(sizeof(int));
- Free(some_pointer_you_malloced);

Don’t free memory you didn’t allocate
Tutorials

**getopt:**

**fscanf:**
- [http://crasseux.com/books/ctutorial/fscanf.html](http://crasseux.com/books/ctutorial/fscanf.html)

**Google is your friend**
Style

Read the style guideline
- But I already read it!
- Good, read it again.

Pay special attention to failure and error checking
- Functions don’t always work
- What happens when a syscall fails??

Start forming good habits now!
Questions