CacheLab

10/10/2011
By Gennady Pekhimenko
Outline

• Memory organization

• Caching
  – Different types of locality
  – Cache organization

• Cachelab
  – Warnings are errors
  – Part (a) Building Cache Simulator
  – Part (b) Efficient Matrix Transpose

• Blocking
Outline

- **Memory organization**
- **Caching**
  - Different types of locality
  - Cache organization
- **Cachelab**
  - Warnings are errors
  - Part (a) Building Cache Simulator
  - Part (b) Efficient Matrix Transpose
- **Blocking**
Memory Hierarchy

- Registers
- SRAM
- DRAM
- Local Secondary storage
- Remote Secondary storage

Today: we study this interaction to give you an idea how caching works
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

• Memory organization

• **Caching**
  – Different types of locality
  – Cache organization

• Cachelab
  – Technical Questions
  – Part (a) Building Cache Simulator
  – Part (b) Efficient Matrix Transpose

• Blocking
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

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

• Memory organization

• Caching
  – Different types of locality
  – Cache organization

• Cachelab
  – Warnings are errors
  – Part (a) Building Cache Simulator
  – Part (b) Efficient Matrix Transpose

• Blocking
Cacheland

• Warnings are errors!

• Include proper header files

• Part (a) Building a cache simulator

• Part (b) Optimizing matrix transpose
Warnings are Errors

• Strict compilation flags

• Reasons:
  – Avoid potential errors that are hard to debug
  – Learn good habits from the beginning
Missing Header Files

• If function declaration is missing
  – Find corresponding header files
  – Use: man <function-name>

• Live example
  – man 3 getopt
Getopt function

NAME

g getopt - Parse command-line options

SYNOPSIS

```
#include <unistd.h>

int getopt(int argc, char * const argv[],
            const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;

#define GNU_SOURCE
#include < getopt.h>

int getopt_long(int argc, char * const argv[],
                const char *optstring,
                const struct option *longopts, int *longindex);

int getopt_long_only(int argc, char * const argv[],
                      const char *optstring,
                      const struct option *longopts, int *longindex);
```

DESCRIPTION

The `getopt()` function parses the command-line arguments. Its arguments `argc` and `argv` are the argument count and array as passed to the `main()` function on program invocation. An element of `argv` that starts with `'- '` (and is not exactly `"-"` or `"--"`) is an option element. The characters of this element (aside
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 -> B)

<table>
<thead>
<tr>
<th>Matrix A</th>
<th>Matrix B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2  3  4</td>
<td></td>
</tr>
<tr>
<td>5  6  7  8</td>
<td></td>
</tr>
<tr>
<td>9 10 11 12</td>
<td></td>
</tr>
<tr>
<td>13 14 15 16</td>
<td></td>
</tr>
</tbody>
</table>
Part (b) Efficient Matrix Transpose

- Matrix Transpose (A -> B)
- Suppose block size is 8 bytes (2 ints)

Matrix A

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<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>

Access A[0][0] cache miss
Access B[0][0] cache miss
Access A[0][1] cache hit
Access B[1][0] cache miss

Matrix B

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Question: After we handle 1&2. Should we handle 3&4 first, or 5&6 first?
Blocking

• What inspiration do you get from previous slide?
  – Divide matrix into sub-matrices
  – This is called **blocking** (CSAPP2e p.629)
  – Size of sub-matrix depends on
    • cache block size, cache size, input matrix size
  – Try different sub-matrix sizes

• We hope you invent more tricks to reduce the number of misses!
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
The End

• Good luck!