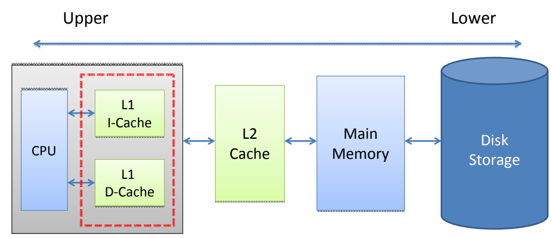
COMP4611 Tutorial 9
Cache Probing

Nov. 12 – Nov. 16

Levels in memory hierarchy (Tutorial 8)



Memory hierarchy (Tutorial 8)

Level	1	2	3	4
Name	Registers	Cache	Main Memory	Disk Storage
Typical size	<1KB	<16MB	<64GB	>1TB
Access time (ns)	0.25-0.5	0.5-25	50-250	5,000,000
Managed by	Compiler	Hardware	Operating System	Operating System/ Operator
Speed	Fastest			Slowest
Capacity	Smallest			Biggest
Cost/bit	Highest			Lowest

Cache size inference

Cache configuration has an impact on memory access performance

For example, on one computer, a memory read may take

- **0.4 tick**
- **3 ticks**
- **14 ticks**
- **25 ticks**

A tick is a processor cycle

Cache size inference

Probe the characteristics of the cache system

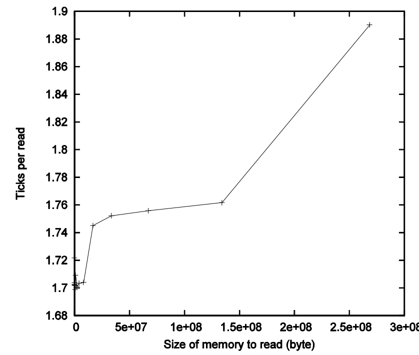
- Deliberately access memory in specific patterns, such as
  - mem[0], mem[1], mem[2], …
  - mem[82], mem[903], mem[400], … (random)
  - mem[99], mem[98], mem[97], …, mem[50], mem[199], mem[198], mem[197], …, mem[150], mem[299], mem[298], mem[297], …, mem[250], …
- Expose some aspects of the memory access performance

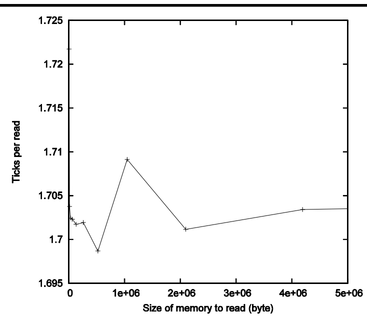
Cache size inference (Homework 3)

Use programs to probe the size of the cache on a real computer only by measuring the memory access time

First attempt: *csweep.c*

- The *csweep.c* as provided in Homework 3
  - measures the average memory access time when sweeping through memory areas from 8K to 256M
- On a PC
  - CPU model: Intel(R) Core(TM) i5 CPU 760 @ 2.80GHz





The cache does have an impact on the memory access performance.

However, probing the cache is a complicated problem, which is affected by various aspects, such as the compiler.

Cache size inference with sophisticated control

- *asmstride.c*
  - adds more sophisticated control on how we access the memory
  - uses inline assembly
- More programs designed by you

Several important functions in asmstride.c

- measure_access_time()
- measure1()
- overhead_measure1()

measure_access_time()

```
double measure_access_time(int32_t low_index,
                           int32_t high_index,
                           int32_t stride_value,
                           int64_t total_accesses) {
  ...
  // Perform memory accesses
  register int32_t count = 0;
  while (count++ < total_loops) {
    low = low_index;
    measure1();
  }
  ...
  // Mock the iteration overhead
  count = 0;
  while (count++ < total_loops) {
    low = low_index;
    overhead_measure1();
  }
  ...
```

Access memory by calling measure1()

Do all the operations as in the previous part except accessing the memory to measure the "overhead".

measure1()

Effect of the inline assembly code in C:

```
while (high > low) {
  i64a += *(long*)(p+896);
  i64a += *(long*)(p+768);
  i64a += *(long*)(p+640);
  i64a += *(long*)(p+512);
  i64a += *(long*)(p+384);
  i64a += *(long*)(p+256);
  i64a += *(long*)(p+128);
  i64a += *(long*)(p);

  t8 += 1; // for counting # of mem accesses
  low += stride;
  p += stride;
}
```


Overhead_measure1()

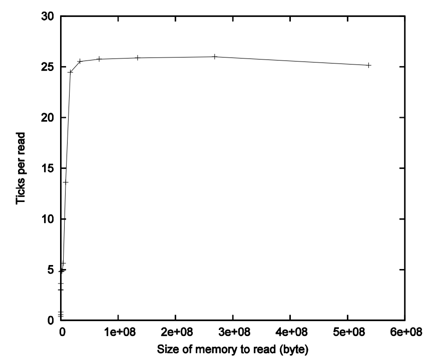
Effect of the inline assembly code in C:

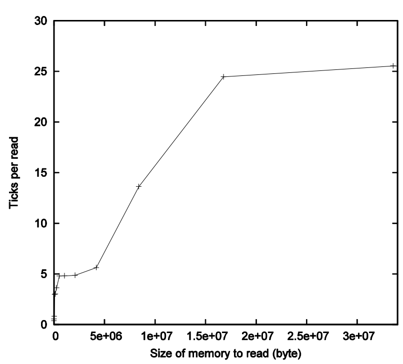
```
while (high > low) {

  t8 += 1;
  low += stride;
  p += stride;
}
```

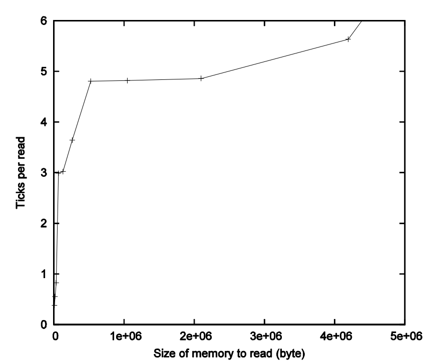
But implementation of these functions in the C code may not be compiled to the instructions equivalent to the inline assembly code

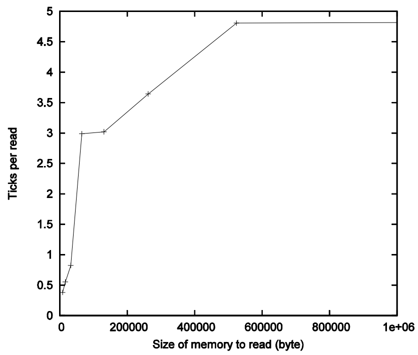
Example of results with *asmstride.c*

- A modified version of *asmstride.c*
  - low_index = 0
  - increasing high_index
  - the same stride_value
- On the PC
  - CPU model: Intel(R) Core(TM) i5 CPU 760 @ 2.80GHz









Homework 3: what to submit

A report named “report.doc” with the following contents:

- experimental environment, including the type and model of the CPU
- plot of the experimental data
- analysis of the result you get, and your estimation of the cache size of the CPU
- the program source code

Submit to HW3 on CASS (https://course.cs.ust.hk/cass).

Thank you!