Memory hierarchy exercises

1. Exam exercises

Exercise 1 *(June 2015). Assume that you have a computer with 1 clock cycle per instruction (1 CPI) when all accesses to memory are in cache. The only accesses to data come from load and store instructions. Those accesses account for 25 % of the total number of instructions. Miss penalty is 50 clock cycles and miss rate is 5 %.*

Determine the speedup obtained when there is no cache miss compared to the case when there are cache misses.

Exercise 2 *(October 2014). Consider the following global variables definition:*

```
const unsigned int max = 1024 * 1024;
double x[max];
double y[max];
double z[max];
double vx[max];
double vy[max];
double vz[max];
```

And the following function:

```
void update_positions(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    x[i] = vx[i] * dt + x[i];
    y[i] = vy[i] * dt + y[i];
    z[i] = vz[i] * dt + z[i];
  }
}
```

Consider a system with a 4 ways set associative L1 cache with a size of 32 KB and a line size of 64 bytes. L2 cache is 8 ways set associative with a size of 1 MB and a line size of 64 bytes. Replacement policy is LRU.

Arrays are consecutively stored in memory and first of them starts at 1024 multiple address.

1. *Determine the hit rate for L1 and L2 caches for running function* ***update_positions()****. What is the global hit rate?*

2. *Modify code applying the array merging optimization.*

3. *Repeat computations from the first question for the code resulting from the second question.*

4. *Assume that a hit at L1 cache requires 4 cycles and that L2 cache requires 14 cycles. Also assume that penalty associated to bringing a block from main memory to L2 cache is 80 cycles. Which is the average access time in each case?*

Exercise 3 *(October 2014) Consider the following global variables definition:*

```
const unsigned int max = 1024 * 1024;
double x[max];
double y[max];
double z[max];
double vx[max];
double vy[max];
double vz[max];
```

And the following function:

```
void update_positions(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    x[i] = vx[i] * dt + x[i];
  }
  for (unsinged int i=0; i<max; ++i) {
    y[i] = vy[i] * dt + y[i];
  }
  for (unsinged int i=0; i<max; ++i) {
    z[i] = vz[i] * dt + z[i];
  }
}
```

Consider a system with a 4 ways set associative L1 cache with a size of 32 KB and a line size of 64 bytes. L2 cache is 8 ways set associative with a size of 1 MB and a line size of 64 bytes. Replacement policy is LRU.

Arrays are consecutively stored in memory and first of them starts at 1024 multiple address.

1. *Determine the hit rate for L1 and L2 caches for running function* ***update_positions()****. What is the global hit rate?*
2. *Modify code applying the loop merging optimization.*
3. *Repeat computations from the first question for the code resulting from the second question.*
4. *Assume that a hit at L1 cache requires 4 cycles and that L2 cache requires 16 cycles. Also assume that penalty associated to bringing a block from main memory to L2 cache is 80 cycles. Which is the average access time in each case?*

Exercise 4 *(October 2013) Given the following code fragment:*

```
struct particle {
  double x, y;
  double ax, ay;
  double vx, vy;
  double mass;
  double load;
};

void move(particle p[], int n, double t) {
  for (int i=0; i<n; ++i) {
    p[i].x += p[i].vx * t + 0.5 *p[i].ax * t * t;
  }
  for (int i=0; i<n; ++i) {
    p[i].y += p[i].vy * t + 0.5 * p[i].ay * t * t;
  }
}
```

Function ***move()*** *is executed for an array of 1000 particles in a system with a L1 data cache with a size of 32 KB and 4 ways set associative. The block size is 64 bytes. Assume that array* ***p*** *is aligned to an address which is 64 multiple.*

You are asked to determine:

1. *The number of cache misses.*

2. *The average miss rate.*
3. *Propose an alternative code using loop merging. What is the new miss rate?*
4. *Propose an alternative code, reorganizing the code in multiple parallel arrays and without loop merging. What is the new miss rate?*
5. *Do you think that it is convenient to apply loop merging in the previous section? Reason your answer.*

Exercise 5 *(October 2013) Consider the following code:*

```
for (i=0; i<64; i++)
  for (j=0; j<1024; j=j+2 )
    v[i][j] = v[i][j] * v[i][j+1] + b[i]
```

Assume an architecture with 256 KB size cache and a block size of 64 bytes and word size of 8 bytes. Cache memory is fully associative and uses an LRU replacement policy. Let ***v*** *and* ***b*** *be 8 byte real number matrices stored by rows (C programming language style) and sizes* 64×1024 *(****v****),* 64 *(****b****). Assume that index variables are stored in processor registers and cache is initially empty.*

Your are asked to determine:

1. *The number of cache misses.*
2. *The average miss rate.*
3. *Reason whether there is some time a replacement of a previously loaded cache line. It is not needed to identify the replacements but it is enough to reason whether they could happen or not.*
4. *Reason if the loop exchanges optimization technique would improve the average miss rate for the cache (it is not needed to compute it).*

Exercise 6 *(October 2013) Given the following code fragment:*

```
int a[100];
int b[100];
for (i=0;i<100;i++)
  a[i]=a[i]+5;
for (i=0;i<100;i++){
  if (i>90)
    c=c+1;
  else
    b[i]=a[i]*3;
}
```

You are asked:

1. *Describe compiler cache optimizations that allow to improve memory access time for this code. Rewrite the code accordingly.*
2. *Consider a computer with cache memory with a line size of 32 bytes. Initially, cache is empty. Which is the cache miss rate when executing the optimized program obtained from the previous question? Consider only compulsory misses for vectors* ***a*** *and* ***b*** *(there are no capacity of conflict misses).*
3. *Consider now a computer with a two-level cache. Both L1 and L2 are unified caches. Clock cycle is 1ns and CPI=1.3. For L1 cache, assume a miss rate of 10 % and a miss penalty of 10ns. For L2 cache hit rate is 95 % and penalty is 80 ns. Assume access time to L1 cache is 1ns. You are requested to compute program execution time for a program with IC instructions where 50 % are read/write instructions.*

Exercise 7 *(June 2011) Consider the following code fragment:*

```
double a[256][256], b[256][256], c[256][256], d[256][256];
// ...
for (int i=0;i<256;++i)
  for (int j=0;j<256;++j) {
    a[i][j] = b[i][j] + c[i][j]
  }
}
for (int i=0;i<256;++i)
  for (int j=0;j<256;++j) {
    d[i][j] = b[i][j] - c[i][j]
  }
}
```

We want to run this code in a computer with fully associative 16 KB level 1 cache. Replacement policy is LRU with a 64 bytes line size. Level 1 cache misses require 16 clock cycles. Besides, level 2 cache always gives hits for this code.

Assume that write misses in level 1 cache are directly sent to a write buffer and do not generate stall cycles.

You are asked:

1. *Determine the hit rate for the code segment, assuming that variables* ***i*** *and* ***j*** *are assigned to processor registers and that matrices* ***a***, ***b***, ***c*** *and* ***d*** *fully reside at level 2 cache.*
2. *Determine the memory access time assuming that accesses to level 1 cache require a clock cycle.*
3. *Propose a code transformation that can be generated by a compiler to improve hit rate, showing the resulting code in the C programming language.*
4. *Determine the new hit rate and the resulting average access time to memory.*

Exercise 8 *(May 2011) Consider an architecture with two cache levels and the following characteristics:*

Memory	*Access time (ns)*	*Hit rate*
L1	*2*	*0.8*
L2	*8*	*0.9*
RAM	*100*	*1*

The computer runs a program fully residing in memory (no disk access). You are asked:

1. *Assuming that 100 % of memory accesses are write operations, calculate and justify the average access time for (a) a write through policy, and (b) a write back policy, both for L1 and L2 caches.*
2. *Consider both cache levels (L1 and L2) as a single global cache. Calculate the average access time and hit rate for this global cache.*
3. *Given the following code:*

```
for (i=0;i<1000;i=i+32){
  a[i] = a[i+8] + a[i+16];
}
```

Size for each entry in array a is 8 bytes and block size is 64 bytes. The loop index is stored in a processor register. You are asked to comment about the effect in performance of using a multi-bank cache with 4 banks. What effect could this approach have on time for each access and cache bandwidth for the given code?