CSci 2021, Spring 2020 Written Exercise Set 4

This assignment is not due. It is for practice.

Problem 1:

The following code snippets do not perform as well as they could. Write new optimized versions of each that will compute the same results, but more quickly.

Part A:

The following function simply calculates the range of a vector (the struct definition of the vector is also given). Recommendation: use separate accumulators to improve instruction-level parallelism.

```
struct vector {
  int length;
  int* arr;
}

int range(struct vector* v) {
  if (!v->length) {
    return NULL;
  }
  int max = v->arr[0];
  int min = v->arr[0];
  for (int i = 1; i < v->length; ++i) {
    if (v->arr[i] > max) {
      max = v->arr[i];
    }
    if (v->arr[i] < min) {
      min = v->arr[i];
    }
  }
  return max - min;
}
```

Part B:

The following function takes in an M*N matrix of ints and returns an array of just two ints, both of which correspond to a different sum. The first element of the result array should be the sum of matrix times two. The second element of the result array should be the sum of the matrix with 10 added to each element. Recommendation: improve temporal locality.

```
int* compute_sums(int** mat, int* res) {
  //zero out result array
  res[0] = 0;
  res[1] = 0;
  //loops to compute sum of mat * 2
  for (int i = 0; i < M; ++i) {
    for (int j = 0; j < N; ++j) {
      res[0] += mat[i][j] * 2;
    }
  }
  //loops to compute sum of mat + 10
  for (int i = 0; i < M; ++i) {
    for (int j = 0; j < N; ++j) {
      res[1] += mat[i][j] + 10;
    }
  }
  return res;
}
```

Problem 2:

In this problem you will practice memory access using virtual memory. The properties you need to know are:

- 14-bit virtual addresses
- 12-bit physical addresses
- 64 byte page size

Below is the TLB. It has 16 entries and is 4-way associative.

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	0D	-	0	1C	-	0	03	12	1	09	-	0
1	01	-	0	05	-	0	05	-	0	08	0A	1
2	05	2A	1	11	1D	1	01	-	0	0A	-	0
3	0B	05	1	0A	-	0	07	1C	1	02	15	1

Below is the first 16 entires of the page table.

VPN	PPN	Valid
00	-	0
01	-	0
02	5D	1
03	-	0
04	-	0
05	-	0
06	1A	1
07	06	1
08	-	0
09	-	0
0A	1B	1
0B	15	1
0C	12	1
0E	-	0
0F	-	0

Below is the cache. Its properties are also follows:

- 16 lines, 4-byte block size
- physically addressed
- direct mapped

Index	Tag	Valid	Byte Offset #0	Byte Offset #1	Byte Offset #2	Byte Offset #3
0	11	0	-	-	-	-
1	15	1	07	21	AA	3B
2	B1	0	-	-	-	-
3	19	1	07	14	B1	2A
4	3A	1	11	A1	05	17
5	21	0	-	-	-	-
6	2A	0	-	-	-	-
7	0B	1	31	B2	1A	27
8	16	0	-	-	-	-
9	B2	0	-	-	-	-
A	C1	0	-	-	-	-
B	12	1	F0	41	2A	54
C	81	0	-	-	-	-
D	35	1	00	14	A4	D1
E	21	1	06	1C	D1	17
F	AB	0	-	-	-	-

A. Draw a diagram of how the 14-bit virtual addresses will be divided into VPN and VPO. Also show how the VPN will be further divided into the TLBT and TLBI.

B. Draw a diagram of how the 12-bit physical addresses will be divided into the PPN and PPO. Also show how the PPN and PPO will be further divided into the CT, CI, CO.

For the follow virtual addresses, give the following information:

- Virtual address in binary
- VPN
- TLB index
- TLB tag
- TLB hit (hit or miss)
- Page fault (yes or no)
- PPN
- Physical address in binary
- cache offset
- cache index
- cache tag
- hit? (yes or no)
- byte

C. 0x32E

D. 0x2C5

E. 0x57B

Problem 3:

The following functions, `find_range` and `bubble_sort`, contain branches in their inner loops that might hurt performance because of misprediction. However the branching is not required: it would be possible to compute the same results using conditional moves instead. To help the compiler see how to do this, rewrite these functions without using `if` statements: instead, the code should only make choices using the `? :` ternary operator, where the second and third arguments of the ternary operator do not have any side-effects.

Comment on how effective this transformation will be as an optimization.

```
int find_range(int* arr, length) {
  int min = arr[0];
  int max = arr[0];
  for (int i = 0; i < length; ++i) {
    if (arr[i] > max) {
      max = arr[i];
    }
    else if (arr[i] < min) {
      min = arr[i];
    }
  }
  return max - min;
}
```

```
int bubble_sort(int arr[], length) {
  int i,j;
  for (int j = 0; j < lenth - 1; j++) {
    for (i = 0; i < (length - j - 2); i++) {
      if (arr[i] > arr[i+1]) {
        int temp = arr[i];
        arr[i] = arr[i+1];
        arr[i+1] = temp;
      }
    }
  }
}
```

Problem 4: (This problem is related to the textbook's 9.10)

On a Unix/Linux system, the `mmap` system call can be used to request the kernel to allocate a virtual memory region on behalf of a process. `mmap` is somewhat analogous to `malloc`, but the allocation is done by the operating system for larger memory regions (typically `mmap` is used for regions whose size is a multiple of the page size, so at least 4K bytes). There is also a system call `munmap`, which is the counterpart of `free` for a region allocated with `mmap`.

Like `malloc`, the memory allocation performed by `mmap` is subject to fragmentation: the virtual address of a region cannot change once it has been allocated, so depending on the layout of regions, it might be that it is impossible to satisfy a request, even when the amount of available virtual memory is sufficient. Explain with a detailed example how this can happen. Assume that were are on a system with a 32-bit virtual address space, and that `mmap` returns regions in the range `0x40000000` through `0xb0000000`. You can assume that before the first de-allocation with `munmap`, `mmap` will perform allocations sequentially, returning the first free region with the lowest address. Describe a sequence of `mmap` and `munmap` operations leading to a fragmented state where it is impossible for the operating system to satisfy a request to `mmap` 100MB, even though more than 1GB of virtual memory is free in

the range that `mmap` uses. (There are many such sequences possible. You might want to consider the effect of allocating regions of different sizes, and the order in which they are deallocated.)

You don't have to write code for your scenario, just describe it with enough detail that it is clear what will happen.