TDT4200 - Parallel Computing
Problem Set 3
Jørgen Faret - MTDT

1a)

Modern processor architectures include pipelining, which means that while one instruction is being performed the next is being fetched, and so on. Code branches cause problems with pipelinining because the processor doesn´t know what the next instruction to execute is when it is executing an instruction in a branch, and therefore doesn´t know which instruction to fetch.

To handle this problem processors today perform something called branch prediction. The processor predicts what the result of the branch is going to be based on past behaviour, and continues the control flow as if there is no branch. Modern processors typically have a very high successfull prediction rate, but the performance hit is still very significant in the case of a wrong prediction. Therefore, as a programmer it is important to try and take this into account and make the branch predictor´s job as easy as possible.

b)

Temporal locality refers to the reuse of the same data in memory within a relatively short time duration of when it is used. Spatial locality refers to the use of data within relatively close storage locations.

c)

Cache works based on the two principals defined in the previous subproblem: temporal locality and spatial locality. The former is the idea that if you recently used a certain chunk of data, you'll probably need it again soon. The latter means that if you recently used the data at address X, you'll probably soon need address X+1. The cache tries to accomodate this by remembering the most recently used chunks of data. It operates with cache lines, typically sized 128 byte or so, so even if you only need a single byte, the entire cache line that contains it gets pulled into the cache. So if you need the following byte afterwards, it'll already be in the cache.

2a)

The average time it took to run the two different functions based on running both 10 times:

- struct_of_arrays() = 1.289s
- array_of_structs() = 1.155s

b)

The layout “struct of arrays” is advantageous because arrays are stored sequentially in memory. Every time the program makes a jump to a new array in the structure, the pointer to the first element in the array is not likely to be cached, but once in the array then the next element for each iteration is likely to be cached because of the principal of spatial locality that the cache takes advantage of. In the case of the other layout, “array of structs”, then the pointer to the first element in each structure is likely to be in cache, but none of the elements in the structure are likely to be cached.

The layout “struct of arrays” performs better because a majority of the memory accesses are performed in the “innermost container”, that is the data container where the elements are actually located.

3

The problem with this code is that way too much memory is allocated, but only a small part of that is used. In addition there is no need to return the value 0.

4

The problem with this code is that it does unneccessart memory accesses, as is apparant in the valgrind figure. The program first iterates up to 32 and writes the current index to the address specified by i. Then the program iterates up to 32 again, and writes to the current index + 64 the value of the address at index i. There is no need for two loops to do this, and there is no need (in this situation) to access memory in the 2nd loop. A better way to perform this could be:

```
for(i = 0; i < 32; i++)
        mem[i] = i;
        mem[i+64] = i;
```

The program also mallocs unnessecary amounts of memory.

5

The problem in this program occurs because the size of strings is assumed to be max 10. In cases where strings are longer than 10, the value in the loop where the *lastChar* value is set will not be set, and it will continue with the value that it previously had. It then continues to the loop that prints the values, where it starts at the value of *lastChar* and iterates backwards.

A problem I encountered here is that my mac machine did not allow the program the complete if it saw the program was writing was overflowing the allocated memory. In addition, my machine did not give the same warnings during compliation (in fact - it gave no warnings at all) as linux.

6

In this task I also encountered a problem because of running on mac. My computer was not able to compile the program, and gave the following error message:

../sys/setparams dc S
gcc -O3 -fno-inline -g -mcmodel=medium -o ../bin/dc.S.x adc.o dc.o extbuild.o rbt.o jobcntl.o ../common/c_print_results.o ../common/c_timers.o ../common/c_wtime.o -lm
ld: in section __TEXT,__text reloc 7: length != 2 and X86_64_RELOC_GOT not supported for architecture x86_64
collect2: ld returned 1 exit status
make[1]: *** [../bin/dc.S.x] Error 1
make: *** [dc] Error 2

When executing this assignment, I was not in Trondheim until the last day before the deadline and could therefore not use the computers at Gløshaugen or borrow one from a classmate. I will speak to the TA during the recitation on Friday the 4th to discuss a possibility to do this problem set again on a machine that properly runs valgrind and is able to compile the code for problem 6.