UNIVERSITY OF HELSINKI
DEPARTMENT OF PHYSICS

TOOLS FOR HIGH PERFORMANCE COMPUTING

Exercise 5

Student: Caike Crepaldi

Professor: Antti Kuronen

September 2015

Problem 1

See the commands and output below.

Command 1

```
$ tar -zxvf mmc.tgz
$ cd ./mmc/src
$ time make F90=gfortran F90FLAGS='-O0 -ffree-line-length-none'

real    0m3.079s
user    0m2.214s
sys     0m0.628s

$ make clean
$ time make F90=gfortran F90FLAGS='-O3 -ffree-line-length-none'

real    0m7.165s
user    0m6.553s
sys     0m0.491s

$ make clean
$ time make F90=gfortran F90FLAGS='-O3 -fbounds-check -ffree-line-length-none'

real    0m9.929s
user    0m9.254s
sys     0m0.569s

$ make clean
```

We can easily notice in the output above that bigger the optimization bigger the compilation time. The execution time, however, decreases for bigger optimizations. This is exacly what we would expect from theory.

Problem 2

The source code can be seen below.

ex5p2.f90

```
1 program command_line
2   implicit none
3   integer :: i
```

```
4     real :: t1,t2
5     integer, parameter :: sp = selected_real_kind(6, 37), n=10000000
6     real(kind=sp), dimension(n) :: a
7
8     a(:)=42.0
9
10    open(2,file='un1.txt',form='unformatted',access='stream',status='replace')
11
12    call cpu_time(t1)
13    do i=1,n
14       write(2) a(i)
15    end do
16    call cpu_time(t2)
17
18    write(6,*) "Elapsed time:", t2-t1
19
20    close(unit=1,status='keep')
21
22    open(2,file='un2.txt',form='formatted',access='stream',status='replace')
23
24    call cpu_time(t1)
25    do i=1,n
26       write(2,'(g0.10)') a(i)
27    end do
28    call cpu_time(t2)
29
30    write(6,*) "Elapsed time:", t2-t1
31
32 end program command_line
```

See the shell commands and output below.

Command 2

```
$ gfortran ex5p2.f90
$ ./a.out
 Elapsed time:    1.46100605
 Elapsed time:    6.28715706
```

With the information in the output above, we can notice that it takes more time for the program to write formatted output in a file than write unformatted one. The difference in the elapsed time is quite noticeable.

Problem 3

The source code can be seen below.

ex5p3.f90

```
1  program command_line
2     implicit none
3     integer :: i
4     real :: t1,t2
5     integer, parameter :: sp = selected_real_kind(6, 37), n=10000000
6     integer :: count1,count2, count_rate, count_max
7     real(kind=sp), dimension(n) :: a
8
9     a(:)=42.0
10
```

```
11    open(2,file='un1.txt',form='unformatted',access='stream',status='replace')
12    call system_clock(count1, count_rate)
13    do i=1,n
14       write(2) a(i)
15    end do
16    call system_clock(count2, count_rate)
17    write(6,*) count2-count1
18    close(unit=1,status='keep')
19
20    open(2,file='un2.txt',form='formatted',access='stream',status='replace')
21    call system_clock(count1, count_rate)
22    do i=1,n
23       write(2,'(g0.10)') a(i)
24    end do
25    call system_clock(count2, count_rate)
26    write(6,*) count2-count1
27    close(unit=1,status='keep')
28
29 end program command_line
```

See the shell commands and output below.

Command 3

```
$ gfortran ex5p3.f90
$ ./a.out
         1470
         6303
```

The two types of time-outputs, in exercise 2 and 3, are different. The system_call function returns the milliseconds of wall clock since the Epoch. Note that the time is not given in seconds but milliseconds. Both values (from exercise 2 and 3) are very close but the cpu_time permits a result with more decimal digits to analyse. Since the system_call uses milliseconds, we get a result with limited decimal digits (when passing the values to seconds).

Problem 4

See below the commands and outputs.

Command 4

```
$ gfortran -o a.out omit_frame_pointer.f90
$ ./a.out
 5.71948886E-16 0.691670954
$ -fomit-frame-pointer -o b.out omit_frame_pointer.f90
$ ./b.out
 5.71948886E-16 0.575793982
$ gfortran -O2 -o c.out omit_frame_pointer.f90
$ ./c.out
 5.71948886E-16 1.91520005E-02
$ gfortran -O2 -fomit-frame-pointer -o d.out omit_frame_pointer.f90
$ ./d.out
 5.71948886E-16 1.98280010E-02
```

In respect of the execution times of the compilated program we can notice that: $(t_a > t_b > t_d > t_c)$, where the index represents the kind of compilation optimization (items a-d in the assignment).

Note that, in our case, $t_d > t_c$. However the difference between t_c and t_d is very small. By searching about this option in the internet, we can assume that this happens because this flag is turned on in all levels of -O (execept for -O0), therefore it is useless to use it with the -O2 option.[1]

[1] `https://wiki.gentoo.org/wiki/GCC_optimization#-fomit-frame-pointer`