

# Tools for High Performance Computing - Exercise 1

Caike Crepaldi

September 2015

This exercise was done using a ssh remote login through my pangolin (pangolin.it.helsinki.fi) university linux account while using my macbook pro with OS X.

To do such a thing, the following command was run in the OS X Yosemite terminal.

```
Macbook-Pro:~ crepaldi$ ssh username@pangolin.it.helsinki.fi
```

The editor I used to edit files and write codes was **emacs**.

## 1 First Problem

Using the shell, the following code was used:

```
username@tktl-pangolin:~$ cat /proc/cpuinfo
```

This command was called many times in a row, so that the system wouldn't adjust the CPU frequency.

The following text was then printed in the shell.

```
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 42
model name    : Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz
stepping       : 2
microcode     : 0x70d
cpu MHz        : 2000.000
cache size    : 20480 KB
physical id   : 0
siblings       : 1
core id        : 0
cpu cores     : 1
apicid         : 0
initial apicid: 0
fpu            : yes
fpu_exception  : yes
cpuid level   : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr
                 sse sse2 ss syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology tsc_reliable nonstop_tsc
                 aperfmpf perf_pni pclmulqdq ssse3 cx16 sse4_1 sse4_2 popcnt aes hypervisor lahf_lm ida arat epb pln pts dtherm
bogomips       : 4000.00
clflush size   : 64
cache_alignment : 64
address sizes  : 40 bits physical, 48 bits virtual
power management:
```

We can see in the text above the cpu MHz information. The clock frequency of this Linux system is 2.0 GHz.

## 2 Second Problem

The Makefile that describes the dependencies shown in the assignment can be seen below.

Makefile

```
1 # -*-makefile-*-
2
3 target: dep_a2 dep_b2
4     cat dep_a2 dep_b2 >target
5
6 dep_a2: dep_a1
7     cp dep_a1 dep_a2
```

```

8
9 dep_b2: dep_b1
10      cp dep_b1 dep_b2

```

After creating the files dep\_a1 and dep\_b1, we can run the Makefile with the command **make** in the terminal.

```

username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ make
cp dep_a1 dep_a2
cp dep_b1 dep_b2
cat dep_a2 dep_b2 >target
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$

```

As an exemple, using a file dep\_a1 which contains the string “Helsingin” and a file dep\_b1 which contains the string “Yliopisto”, the command **make** produces the archives dep\_a2 (copy of dep\_a1), dep\_b2 (copy of dep\_b1) and target (concatenation of the archives dep\_a2 and dep\_b2). Therefore the target file will contain the string “HelsinginYliopisto”.

After running the command **make** for the first time, if we try to run it again immediately after that the shell will return the following message.

```

username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ make
make: ‘target’ is up to date.
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$

```

However if we use the command **touch** in one of those files, the result is different, since we touched one of the archives that the target file depends. This is shown bellow.

```

username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ touch dep_b1
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ make
cp dep_b1 dep_b2
cat dep_a2 dep_b2 >target
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$

```

We can see that the commands that depended on the touched file were run once again in order to update the files that depend on the touched file.

### 3 Third Problem

To test that strange circular dependence lets create the Makefile first.

Makefile

```

1 # -*-makefile -*-
2
3 target1: target3
4
5 target3: target2
6
7 target2: target1

```

After creating the three target files (in my case I just let them as blank files), we can use the command **make** and see what happens.

```

username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ make
make: Circular target2 <- target1 dependency dropped.
make: Nothing to be done for ‘target1’.
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$

```

### 4 Fourth Problem

First we have to download the gauss.tgz file and unpack it. After that we have to compile the program without optimization.

I will use the fortran code to compile the program and run it in the **gdb**.

The commands are shown bellow.

```

username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ wget "http://www.courses.physics.helsinki.fi/fys/stltk/exercises/gauss.tgz"
2015-09-10 19:51:27 (149 MB/s) - ‘gauss.tgz’ saved [1502/1502]
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ tar zxvf gauss.tgz
gauss/
gauss/matrix
gauss/gauss.c
gauss/gauss.f90
username@tktl-pangolin:~/Tools-for-HPC/exercise-1$ cd gauss

```

```
username@tktl-pangolin:~/Tools-for-HPC/exercise-1/gauss$ gfortran -O0 -g -o gauss gauss.f90
```

```
username@tktl-pangolin:~/Tools-for-HPC/exercise-1/gauss$ gdb gauss
```

```
The fortran source code for the gauss program is shown bellow.
```

gauss.f90

```
1 module sizes
2   integer , parameter :: rk=selected_real_kind(10,40)
3   integer , parameter :: MATSIZE=4
4 end module sizes
5
6 subroutine gauss(a,b,x)
7   use sizes
8   implicit none
9   real (rk), intent(inout) :: a(MATSIZE,MATSIZE),b(MATSIZE)
10  real (rk), intent(out) :: x(MATSIZE)
11  integer :: p(MATSIZE)
12  real(rk) :: s(MATSIZE)
13  integer :: i,j,k,pk
14  real(rk) :: smax,rmax,r,z,ss
15
16 do i=1,MATSIZE
17   p(i) = i
18   smax = 0.0
19   do j=1,MATSIZE
20     smax = max(smax,abs(a(i,j)))
21   enddo
22   s(i) = smax
23 enddo
24
25 do k=1,MATSIZE-1
26   rmax = 0.0
27   do i=k,MATSIZE
28     r = abs(a(p(i),k))/s(p(i))
29     if (r > rmax) then
30       j = i
31       rmax = r
32     endif
33   enddo
34
35   pk = p(j)
36   p(j) = p(k)
37   p(k) = pk
38
39   do i=k+1,MATSIZE
40     z = a(p(i),k)/a(p(k),k)
41     a(p(i),k) = z
42     do j=k+1,MATSIZE
43       a(p(i),j) = a(p(i),j) - z*a(p(k),j)
44     enddo
45   enddo
46 enddo
47
48 do k=1,MATSIZE-1
49   do i=k+1,MATSIZE
50     b(p(i)) = b(p(i)) - a(p(i),k)*b(p(k))
51   enddo
52 enddo
53
54 backsubst: do i=MATSIZE,1,-1
55   ss = b(p(i))
56   do j=i+1,MATSIZE
```

```

57      ss = ss - a(p(i),j)*x(j)
58      enddo
59      x(i) = ss/a(p(i),i)
60  enddo backsubst
61
62  return
63 end subroutine gauss
64
65
66 program gaussmain
67  use sizes
68  implicit none
69  real(rk) :: a(MATSIZE,MATSIZE), b(MATSIZE), x(MATSIZE)
70  integer :: i, j
71
72  do i=1,MATSIZE
73    read(5,*) (a(i,j),j=1,MATSIZE)
74  enddo
75  read(5,*) (b(i),i=1,MATSIZE)
76
77  write(6,'(/ a)') ' Before '
78  write(6,'(/ a)') 'A'
79  do i=1,MATSIZE
80    write(6,'(20g16.8)') (a(i,j),j=1,MATSIZE)
81  enddo
82  write(6,'(/ a)') 'b'
83  write(6,'(20g16.8)') b
84
85  call gauss(a,b,x)
86
87  write(6,'(/ a)') ' After '
88  write(6,'(/ a)') 'A'
89  do i=1,MATSIZE
90    write(6,'(20g16.8)') (a(i,j),j=1,MATSIZE)
91  enddo
92  write(6,'(/ a)') 'x'
93  write(6,'(20g16.8)') x
94
95 end program gaussmain

```

## 4.1 Item a

Now, in the assignment, it is said to “Print out the value of the arguments  $b$  and  $x$  in the beginning of routine gauss”. However I’m not really sure about what “routine gauss” it refers to. It could be talking about the whole program (by calling it a routine) or the subroutine gauss. I will suppose that it refers to the subroutine. In other words, I must print out the value of both arguments immediately before the program calls the subroutine gauss. To do so in the **gdb** we must use a breakpoint in the line 84. See commands bellow.

```

(gdb) b 84
Breakpoint 1 at 0x40135d: file gauss.f90, line 84.
(gdb) run < matrix
Breakpoint 1, gaussmain () at gauss.f90:85
85 call gauss(a,b,x)
(gdb) display b
1: b = (0.428892365341, 0.30461736686939, 0.18965374754716999, 0.19343115640521999)
(gdb) display x
2: x = (3.9525251667299724e-323, 0, 3.1245590879850029e-317, 6.953349068398127e-310)

```

## 4.2 Item b

In order to monitor the value of the variable ss during all iterations in the backsubstitution loop, we will need to set a breakpoint in the line 53 and use the command **watch**. See commands and the result bellow.

```

(gdb) b 53
Breakpoint 1 at 0x400daa: file gauss.f90, line 53.
(gdb) run < matrix
Breakpoint 1, gauss (a=..., b=..., x=...) at gauss.f90:54
54 backsubst: do i=MATSIZE,1,-1
(gdb) watch ss
Hardware watchpoint 2: ss
(gdb) cont
Old value = 6.9533558075061542e-310
New value = -0.36924162090909646
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = -0.36924162090909646
New value = -0.20807911638869767
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = -0.20807911638869767
New value = -0.028288423962739256
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = -0.028288423962739256
New value = -0.10772584337145805
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = -0.10772584337145805
New value = -0.090112438506848122
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = -0.090112438506848122
New value = -0.0069926683737533274
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = -0.0069926683737533274
New value = 0.428892365341
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = 0.428892365341
New value = 0.43178716855547911
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = 0.43178716855547911
New value = 0.44082869712412048

```

```
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.
Hardware watchpoint 2: ss

Old value = 0.44082869712412048
New value = 0.44743905642244847
gauss (a=..., b=..., x=...) at gauss.f90:56
56      do j=i+1,MATSIZE
(gdb)
Continuing.

Watchpoint 2 deleted because the program has left the block in
which its expression is valid.
```

We can easily see the values of the variable *ss* in the results shown above. Those values are shown in the “Old value” and “New value” fields that are updated in each iteration.

## Note: About this exercise

All files created by the student and used in this exercise (in this case only the Makefiles) were packed together with this PDF documentation and shall be available to the assistant for further analysis in its own problem folder.