UNIVERSITY OF HELSINKI
DEPARTMENT OF PHYSICS

TOOLS FOR HIGH PERFORMANCE COMPUTING

Exercise 4

Student: Caike Crepaldi

Professor: Antti Kuronen

October 2015

See the README file for further information about the exercise.

Problem 1

Item (a)

The modified (optimized) code can be seen below.

ex4p1a.f90

```
1  program optimization1
2    implicit none
3    integer :: i,m
4    integer, parameter :: n = 10000000
5    real :: a(n), b(n),t1,t2
6
7    a(:)=42.0
8    b(:)=42.0
9    call cpu_time(t1)
10
11   ! Unroll: depth=8
12   m = mod(499,8)
13   do i=1,m
14       a(i) = 4.0*b(i)+b(i+1)
15   end do
16
17   do i = m+1,499,8
18       a(i) = 4.0*b(i+0)+b(i+0+1)
19       a(i+1) = 4.0*b(i+1)+b(i+1+1)
20       a(i+2) = 4.0*b(i+2)+b(i+2+1)
21       a(i+3) = 4.0*b(i+3)+b(i+3+1)
22       a(i+4) = 4.0*b(i+4)+b(i+4+1)
23       a(i+5) = 4.0*b(i+5)+b(i+5+1)
24       a(i+6) = 4.0*b(i+6)+b(i+6+1)
25       a(i+7) = 4.0*b(i+7)+b(i+7+1)
26   end do
27
```

```
28    ! Unroll: depth=8
29    m = mod(n-1,8)
30    do i=500,m
31        a(i) = 4.0*b(i+1)+b(i)
32    end do
33
34    do i = m+1,n-1,8
35        a(i+0)=4.0*b(i+0+1)+b(i+0)
36        a(i+1)=4.0*b(i+1+1)+b(i+1)
37        a(i+2)=4.0*b(i+2+1)+b(i+2)
38        a(i+3)=4.0*b(i+3+1)+b(i+3)
39        a(i+4)=4.0*b(i+4+1)+b(i+4)
40        a(i+5)=4.0*b(i+5+1)+b(i+5)
41        a(i+6)=4.0*b(i+6+1)+b(i+6)
42        a(i+7)=4.0*b(i+7+1)+b(i+7)
43    end do
44
45  !  a(1:499) = 4.0*b(1:499)+b(2:500)
46  !  a(500:n) = 4.0*b(501:n+1)+b(500:n)
47    call cpu_time(t2)
48    print *, "Elapsed time:", t2-t1
49  end program optimization1
```

See the makefile for the compilation commands.
The output (elapsed CPU times) can be seen below.
Modified code with -O0 optimization option:

```
Elapsed time:    2.27739997E-02
```

Modified code with -O3 optimization option:

```
Elapsed time:    2.00048089E-06
```

Original code with -O0 optimization option:

```
Elapsed time:    3.27620022E-02
```

Original code with -O3 optimization option:

```
Elapsed time:    2.00048089E-06
```

Item (b)

ex4p1b.f90

```
1  program optimization2
2    implicit none
3    integer, parameter :: n=10000
4    real :: a(n,n),b(n,n),c(n),d,t1,t2
5    integer :: i,j
6    b(:,:)=23.0
7    c(:)=7.0
8
9    call cpu_time(t1)
10   a(:,:)=b(:,:)
11   do i=1,n,8
12       a(i,:)=a(i,:)/c(i)
13       a(i+1,:)=a(i+1,:)/c(i+1)
14       a(i+2,:)=a(i+2,:)/c(i+2)
15       a(i+3,:)=a(i+3,:)/c(i+3)
16       a(i+4,:)=a(i+4,:)/c(i+4)
17       a(i+5,:)=a(i+5,:)/c(i+5)
```

```
18        a(i+6,:)=a(i+6,:)/c(i+6)
19        a(i+7,:)=a(i+7,:)/c(i+7)
20      end do
21      call cpu_time(t2)
22
23      print *, "Elapsed time:", t2-t1
24
25  end program optimization2
```

See the makefile for the compilation commands.
The output (elapsed CPU times) can be seen below.
Modified code with -O0 optimization option:

```
Elapsed time:    1.42182302
```

Modified code with -O3 optimization option:

```
Elapsed time:    1.47281110
```

Original code with -O0 optimization option:

```
Elapsed time:    3.14909101
```

Original code with -O3 optimization option:

```
Elapsed time:    1.99675560E-06
```

Problem 2

The source code can be seen below.

ex4p2.f90

```
1  program matrixproducts
2    implicit none
3    real :: a(1000,1000),b(1000,1000),c(1000,1000)
4    integer :: i,j
5
6    a(:,:)=42.0
7    b(:,:)=a(:,:)
8
9
10   call product1(a,b,c)
11
12   call product2(a,b,c)
13
14   call product3(a,b,c)
15
16 end program matrixproducts
17
18 subroutine product1(a,b,c)
19   implicit none
20   integer, parameter :: dp = selected_real_kind(15, 307)
21   real :: a(1000,1000),b(1000,1000),c(1000,1000)
22   real(kind=dp) :: ti,tf
23   integer :: i,j,k
24   call cpu_time(ti)
25   c(:,:)=0.0
26   do i=1,1000
27      do j=1,1000
28         do k=1,1000
29            c(i,j)=c(i,j)+a(i,k)*b(k,j)
30         end do
```

```
31        end do
32     end do
33     call cpu_time(tf)
34     print *, "time:", tf-ti
35  end subroutine product1
36
37  subroutine product2(a,b,c)
38     implicit none
39     integer, parameter :: dp = selected_real_kind(15, 307)
40     real :: a(1000,1000),b(1000,1000),c(1000,1000)
41     real(kind=dp) :: ti,tf
42     integer :: i,j,k
43     call cpu_time(ti)
44     c(:,:)=0.0
45     do i=1,1000
46        do j=1,1000
47           do k=1,1000
48              c(i,j)=c(i,j)+a(j,k)*b(k,j)
49           end do
50        end do
51     end do
52     call cpu_time(tf)
53     print *, "time:", tf-ti
54  end subroutine product2
55
56  subroutine product3(a,b,c)
57     implicit none
58     integer, parameter :: dp = selected_real_kind(15, 307)
59     real :: a(1000,1000),b(1000,1000),c(1000,1000)
60     real(kind=dp) :: ti,tf
61     integer :: i,j,k
62     call cpu_time(ti)
63     c(:,:)=0.0
64     do i=1,1000
65        do j=1,1000
66           do k=1,1000
67              c(i,j)=c(i,j)+a(i,k)*b(k,i)
68           end do
69        end do
70     end do
71     call cpu_time(tf)
72     print *, "time:", tf-ti
73  end subroutine product3
```

See the makefile for the compilation commands.

The output (elapsed CPU times) can be seen below.

-O0 optmization option:

```
time:    4.7683210000000003
time:    6.4779070000000001
time:    4.7146000000000008
```

-O3 optmization option:

```
time:    1.0720490000000000
time:    1.2282850000000001
time:    1.0395799999999999
```

We can see that it takes more time to calculate the $A^T B$ product. The other 2 matricial products spend similar execution time (≈ 4.7 s).

The optimization (-O3) increases the speed of all product types. In that case, the execution times becomes much more similar (≈ 1 s). Still, the $A^T B$ product have the biggest execution time (≈ 1.22 s).

Problem 3

The source code can be seen below.

ex4p3.f90

```
1  program unroll_test
2    implicit none
3    integer :: m,n=10000,i
4    integer(kind=4), dimension(10000) :: a
5    real(kind=16) :: t1,t2
6
7    ! Unroll: depth=1
8    call cpu_time(t1)
9   m = mod(n,1)
10   do i=1,m
11      a(i) = i
12   end do
13
14   do i = m+1,n,1
15      a(i+0) = i+0
16   end do
17   call cpu_time(t2)
18   print *, "Elapsed time:", t2-t1
19   ! Unroll: depth=2
20   call cpu_time(t1)
21  m = mod(n,2)
22   do i=1,m
23      a(i) = i
24   end do
25
26   do i = m+1,n,2
27      a(i+0) = i+0
28      a(i+1) = i+1
29   end do
30   call cpu_time(t2)
31   print *, "Elapsed time:", t2-t1
32   ! Unroll: depth=4
33   call cpu_time(t1)
34  m = mod(n,4)
35   do i=1,m
36      a(i) = i
37   end do
38
39   do i = m+1,n,4
40      a(i+0) = i+0
41      a(i+1) = i+1
42      a(i+2) = i+2
43      a(i+3) = i+3
44   end do
45   call cpu_time(t2)
46   print *, "Elapsed time:", t2-t1
47   ! Unroll: depth=8
48   call cpu_time(t1)
49  m = mod(n,8)
```

```
50    do i=1,m
51       a(i) = i
52    end do
53
54    do i = m+1,n,8
55       a(i+0) = i+0
56       a(i+1) = i+1
57       a(i+2) = i+2
58       a(i+3) = i+3
59       a(i+4) = i+4
60       a(i+5) = i+5
61       a(i+6) = i+6
62       a(i+7) = i+7
63    end do
64    call cpu_time(t2)
65    print *, "Elapsed time:", t2-t1
66    ! Unroll: depth=16
67    call cpu_time(t1)
68    m = mod(n,16)
69    do i=1,m
70       a(i) = i
71    end do
72
73    do i = m+1,n,16
74       a(i+0) = i+0
75       a(i+1) = i+1
76       a(i+2) = i+2
77       a(i+3) = i+3
78       a(i+4) = i+4
79       a(i+5) = i+5
80       a(i+6) = i+6
81       a(i+7) = i+7
82       a(i+8) = i+8
83       a(i+9) = i+9
84       a(i+10) = i+10
85       a(i+11) = i+11
86       a(i+12) = i+12
87       a(i+13) = i+13
88       a(i+14) = i+14
89       a(i+15) = i+15
90    end do
91    call cpu_time(t2)
92    print *, "Elapsed time:", t2-t1
93    ! Unroll: depth=32
94    call cpu_time(t1)
95    m = mod(n,32)
96    do i=1,m
97       a(i) = i
98    end do
99
100   do i = m+1,n,32
101      a(i+0) = i+0
102      a(i+1) = i+1
103      a(i+2) = i+2
104      a(i+3) = i+3
105      a(i+4) = i+4
106      a(i+5) = i+5
```

```
107         a(i+6) = i+6
108         a(i+7) = i+7
109         a(i+8) = i+8
110         a(i+9) = i+9
111         a(i+10) = i+10
112         a(i+11) = i+11
113         a(i+12) = i+12
114         a(i+13) = i+13
115         a(i+14) = i+14
116         a(i+15) = i+15
117         a(i+16) = i+16
118         a(i+17) = i+17
119         a(i+18) = i+18
120         a(i+19) = i+19
121         a(i+20) = i+20
122         a(i+21) = i+21
123         a(i+22) = i+22
124         a(i+23) = i+23
125         a(i+24) = i+24
126         a(i+25) = i+25
127         a(i+26) = i+26
128         a(i+27) = i+27
129         a(i+28) = i+28
130         a(i+29) = i+29
131         a(i+30) = i+30
132         a(i+31) = i+31
133       end do
134       call cpu_time(t2)
135       print *, "Elapsed time:", t2-t1
136
137 end program unroll_test
```

See the makefile for the compilation commands.

The output (elapsed CPU times) can be seen below.

-O0 optmization option:

```
Elapsed time:    5.09999999999999999999999999999999448E-0005
Elapsed time:    1.39999999999999999999999999999999443E-0005
Elapsed time:    8.99999999999999999999999999999992388E-0006
Elapsed time:    8.99999999999999999999999999999992388E-0006
Elapsed time:    6.99999999999999999999999999999987810E-0006
Elapsed time:    9.00000000000000000000000000000011196E-0006
```

-O3 optmization option:

```
Elapsed time:    1.10000000000000000000000000000001577E-0005
Elapsed time:    0.00000000000000000000000000000000000
Elapsed time:    1.99999999999999999999999999999985770E-0006
Elapsed time:    0.00000000000000000000000000000000000
Elapsed time:    0.00000000000000000000000000000000000
Elapsed time:    1.00000000000000000000000000000011693E-0006
```

Problem 4

The source code can be seen below.

ex4p4.f90

```
1 program vectorize_test
2   implicit none
3   integer, parameter :: n=10000
4   real :: a(n,n),b(n,n),c(n),d=255,t1,t2
```

```
  integer :: i,j

  call cpu_time(t1)
  do i=1,n
     b(i,:)=(i*20)/17.0
     c(i)=i*23
  end do


  do i=1,n
     do j=1,n
        a(i,j)=c(i)*b(j,i)+d
     end do
  end do
  call cpu_time(t2)

  print *, "Elapsed time:", t2-t1

end program vectorize_test
```

See the makefile for the compilation commands.
The output (elapsed CPU times) can be seen below.
-O2 -ftree-vectorize -ftree-vectorizer-verbose=1 optmization option:

```
Elapsed time:   0.495373994
```

-O2 -fno-tree-vectorize optmization option:

```
Elapsed time:    1.57867610
```

A Makefiles

Makefile.ex4p1

```
# -*-makefile-*-

F90=gfortran
PROBLEMS=ex4p1a ex4p1a-original ex4p1b ex4p1b-original

.PHONY: all clean print

all: $(PROBLEMS) print

ex4p1a: ex4p1a.f90
        $(F90) -O0 -o ex4p1a $<
        ./ex4p1a > ex4p1a-O0.output
        $(F90) -O3 -o ex4p1a $<
        ./ex4p1a > ex4p1a-O3.output
        rm -f ex4p1a

ex4p1a-original: ex4p1a-original.f90
        $(F90) -O0 -o ex4p1a-original $<
        ./ex4p1a-original > ex4p1a-original-O0.output
        $(F90) -O3 -o ex4p1a-original $<
        ./ex4p1a-original > ex4p1a-original-O3.output
        rm -f ex4p1a-original

ex4p1b: ex4p1b.f90
        $(F90) -O0 -o ex4p1b $<
```

```
26          ./ex4p1b > ex4p1b-O0.output
27          $(F90) -O3 -o ex4p1b $<
28          ./ex4p1b > ex4p1b-O3.output
29          rm -f ex4p1b
30
31  ex4p1b-original: ex4p1b-original.f90
32          $(F90) -O0 -o ex4p1b-original $<
33          ./ex4p1b-original > ex4p1b-original-O0.output
34          $(F90) -O3 -o ex4p1b-original $<
35          ./ex4p1b-original > ex4p1b-original-O3.output
36          rm -f ex4p1b-original
37
38  print:
39          cat *.output | less
40
41  clean:
42          rm -f *.output *.o
```

Makefile.ex4p2

```
1  # -*-makefile-*-
2
3  F90=gfortran
4
5  .PHONY: clean
6
7  ex4p2: ex4p2.f90
8          $(F90) -O0 -o ex4p2 $<
9          ./ex4p2 > ex4p2-O0.output
10         $(F90) -O3 -o ex4p2 $<
11         ./ex4p2 > ex4p2-O3.output
12         rm -f ex4p2
13         cat *.output | less
14
15 clean:
16         rm -f *.o *.output
```

Makefile.ex4p3

```
1  # -*-makefile-*-
2
3  F90=gfortran
4
5  .PHONY: clean
6
7  ex4p3: ex4p3.f90
8          $(F90) -O0 -o ex4p3 $<
9          ./ex4p3 > ex4p3-O0.output
10         $(F90) -O3 -o ex4p3 $<
11         ./ex4p3 > ex4p3-O3.output
12         rm -f ex4p3
13         cat *.output | less
14
15 clean:
16         rm -f *.o *.output
```

Makefile.ex4p4

```
1  # -*-makefile-*-
```

```
2
3 F90=gfortran
4
5 .PHONY: clean
6
7 ex4p4: ex4p4.f90
8         $(F90) -O2 -ftree-vectorize -ftree-vectorizer-verbose=1 -o ex4p4 $<
9         ./ex4p4 > ex4p4-vectorized.output
10        $(F90) -O2 -fno-tree-vectorize -o ex4p4 $<
11        ./ex4p4 > ex4p4-no-tree-vectorized.output
12        rm -f ex4p4
13        cat *.output | less
14
15 clean:
16        rm -f *.o *.output
```

B Unroll script

For manually unroll the do-loops, I used the m4 macro processor language in order to generate the fortran code. See below an exemple.

unroll–vec–fort.m4

```
1 '! unroll fortran do-loops with a desired depth'
2 '! syntax unroll(index1, nmax, depth, calculation, index2)'
3 '! index1 and 2 must be different but the output will be with index1 only'
4 define('forloop',
5        'pushdef('$1', '$2')_forloop('$1', '$2', '$3', '$4')popdef('$1')')
6 define('_forloop',
7        '$4''ifelse($1, '$3', ,
8                    'define('$1', incr($1))_forloop('$1', '$2', '$3', '$4')')')dnl
9 define('unroll','define('var',$1)dnl
10 m = mod($2,$3)
11 do var=1,m
12    a(var) = var
13 end do
14 dnl
15 do var = m+1,$2,$3
16 forloop('j',0,decr($3),'  define($5, var+'j')$4
17 ')dnl
18 end do')dnl
19 define(NMAX, 'n')dnl
20 '! Unroll: depth=1'
21 'call cpu_time(t1)'
22 unroll('i','n',1,'a(k) = k','k')
23 'call cpu_time(t2)'
24 'print *, "Elapsed time:", t2-t1'
25
26 '! Unroll: depth=2'
27 'call cpu_time(t1)'
28 unroll('i','n',2,'a(k) = k','k')
29 'call cpu_time(t2)'
30 'print *, "Elapsed time:", t2-t1'
31
32 '! Unroll: depth=4'
33 'call cpu_time(t1)'
34 unroll('i','n',4,'a(k) = k','k')
```

```
35 'call cpu_time(t2)'
36 'print *, "Elapsed time:", t2-t1'
37
38 '! Unroll: depth=8'
39 'call cpu_time(t1)'
40 unroll('i','n',8,'a(k) = k','k')
41 'call cpu_time(t2)'
42 'print *, "Elapsed time:", t2-t1'
43
44 '! Unroll: depth=16'
45 'call cpu_time(t1)'
46 unroll('i','n',16,'a(k) = k','k')
47 'call cpu_time(t2)'
48 'print *, "Elapsed time:", t2-t1'
49
50 '! Unroll: depth=32'
51 'call cpu_time(t1)'
52 unroll('i','n',32,'a(k) = k','k')
53 'call cpu_time(t2)'
54 'print *, "Elapsed time:", t2-t1'
```