DeepL
Subscribe to DeepL Pro to translate larger documents.
Visit www.DeepL.com/pro for more information.

High Performance Computing
HPC

Problems Topic 2: Vectorization and Parallelization of Loops

2.1.- The following loop has several types of dependencies. Find all true dependencies, anti-dependencies and exit dependencies, and remove the anti-dependencies and exit dependencies by renaming them.

```
for (i=0; i<100; i=i+1) {
        Y[i] = X[i] / c;        /* S1 */
        X[i] = X[i] + c;        /* S2 */
        Z[i] = Y[i] + c;        /* S3 */
        Y[i] = c - Y[i];        /* S4 */
}
```

SOLUTION:

The following dependences exist among the four statements:
1. There are true dependences from S1 to S3 and from S1 to S4 because of Y[i]. These are not loop carried, so they do not prevent the loop from being considered parallel. These dependences will force S3 and S4 to wait for S1 to complete.
2. There is an antidependence from S1 to S2, based on X[i].
3. There is an antidependence from S3 to S4 for Y[i].
4. There is an output dependence from S1 to S4, based on Y[i].

The following version of the loop eliminates these false (or pseudo) dependencies.

```
for (i=0; i<100; i=i+1 {
        T[i] = X[i] / c; /* Y renamed to T to remove output dependence */
        X1[i] = X[i] + c;/* X renamed to X1 to remove antidependence */
        Z[i] = T[i] + c;/* Y renamed to T to remove antidependence */
        Y[i] = c - T[i];
}
```

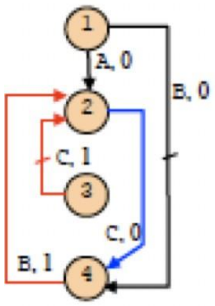
After the loop, the variable X has been renamed X1. In code that follows the loop, the compiler can simply replace the name X by X1. In this case, renaming does not require an actual copy operation, as it can be done by substituting names or by register allocation. In other

2.2.- The following loop is going to be executed in a multithread machine:

```
do i = 1, N-2
        (1) A(i) =B(i)
        (2) C(i) = A(i) + B(i-1)
        (3) D(i) = C(i+1)
        (4) B(i) = C(i) * 2
enddo
```

Generate the dependency network
Write a parallel version of such a loop and assuming that the execution time of an instruction is T, how many times faster is the execution when parallelizing on P processors. What will be the maximum speedup with infinite processors and neglecting the synchronization time.

SOL




```
doall i = 1, N-2              ; 1 y 3 se pueden ejecutar en paralelo
(1)  A(i) = B(i)
(3)  D(i) = C(i+1)
enddoall

[barrera]

do i = 1, N-2                 ; este bloque hay que ejecutarlo en serie
(2)  C(i) = A(i) + B(i-1)
(4)  B(i) = C(i) * 2
enddo
```

Serial run time Ts= Nx4xT

Run time with improvements Tp= (Nx2xT)/p + Nx2xT If p =

infinity S= 2.

2.3.- Examine the following loops and analyze their possible parallelization:

a) Do the iterations of the following loop have

dependencies? for (i=0;i<100;i++) {

```
A[i] = B[2*i+4];
B[4*i+5] = A[i];
}
```

I

SOL:

5 -4 /GCD(2,4) = ½ =0.5 not integer => No dependencies

b) Find all true dependencies, anti-dependencies and exit dependencies, and remove anti-dependencies and exit dependencies by renaming them.

```
for (i=0;i<100;i++) {
    A[i] = A[i] * B[i]; /* S1 */
    B[i] = A[i] + c; /* S2 */
    A[i] = C[i] * c; /* S3 */
    C[i] = D[i] * A[i]; /* S4 */
}
```

SOL

Output dependencies

S1 and S3 cause through A[i]

Anti-dependencies

S4 and S3 cause an anti-dependency through C[i]

Re-written code

```
for (i=0;i<100;i++) {
T[i] = A[i] * B[i]; /* S1 */
B[i] = T[i] + c; /* S2 */
A1[i] = C[i] * c; /* S3 */
C1[i] = D[i] * A1[i]; /* S4 */}
```

True dependencies

S4 and S3 through A[i]

S2 and S1 through T[i]

S4 and S3 through A[i]

S2 and S1 through T[i]

S2 and S1 through T[i]

c) Consider the following loop:

```
for (i=0;i < 100;i++) {
    A[i] = A[i] + B[i]; /* S1 */
    B[i+1] = C[i] + D[i]; /* S2 */
}
```

Are there dependencies between S1 and S2 and is it parallelizable? If not, show how to parallelize it.

SOL

There is an anti-dependence between iteration i and i+1 for array B. This can be avoided by renaming the B array in S2

2.4. Represent the dependency network of the following loops:

a)

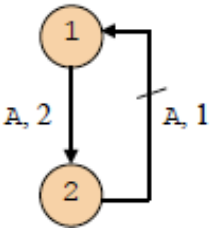
```
do i = 1, N-2
        (1) A(i) =B(i) +2
        (2) C(i) = A(i-2) +A(i+1)
enddo
```

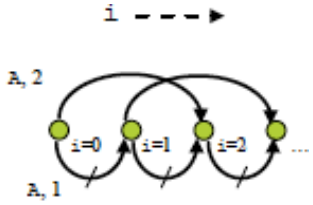
b)

```
do i = 1, N-2
do j = 1, N-2
        (1) A(i,j) =A(i,j-1) *2
        (2) C(i,j) = A(i-2.j+1) +1
Enddo
enddo
enddo
```

SUN a.-

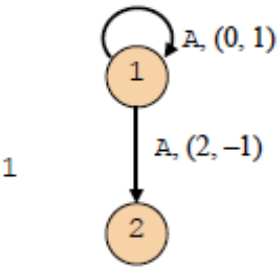
```
do i = 2, N-2
1:  A(i) = B(i) + 2
2:  C(i) = A(i-2) + A(i+1)
enddo
```



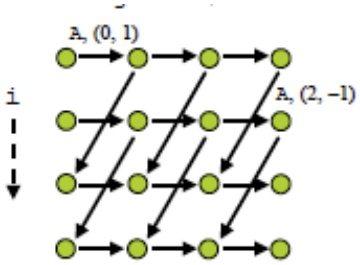


SUN b.-

```
do i = 2, N-1
do j = 1, N-2
1:  A(i,j) = A(i,j-1) * 2
2:  C(i,j) = A(i-2,j+1) + 1
enddo
enddo
```



grafo de dependencias



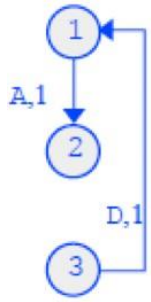
espacio de iteraciones

2.5.- The following loop is going to be executed in a 4 processors SMP machine:

```
do i = 1, 994
      (1) A(i) = D(i) - 10
      (2) B(i+1) = B(i+1) * A(i-1)
      (3) D(i+1) = D(i+1) + D(i+1)
Enddo
```

a) Generate the dependency network and iteration space for the loop. Write a parallel version of the loop that can run on P processors without having to use synchronization functions.

SOL




```
A(1) = D(3) - 10
B(2) = B(2) * A(0)
B(3) = B(3) * A(1)
doall i = 1, 992
    D(i+1) = D(i+1) + D(i+1)
    A(i+1) = D(i+1) - 10
    B(i+3) = B(i+3) + A(i+1)
enddoall
D(994) = D(994) + D(994)
A(994) = D(994) - 10
D(995) = D(995) + D(995)
```

b) For the case of this 4-processor machine, write the code that each processor (the parallel loop) will execute if the distribution of iterations between the processors is (1) static consecutive; and (2) dynamic by chunks, with chunks of 50 iterations. How many iterations will processor P1 execute in the first case? and in the second case?

The doall loop of 992 iterations can be distributed as you wish. **1.-**
consecutive static: each processor executes 992/4 = 128 P0:
i=1,248
P1: i 249.496
P2: i = 497.744
P3: i = 745.942

1.- dynamic each processor executes 50 iterations
P1 can't know 250 if he is the hardest worker and 200 or 242 in other cases.

2.6- For each of the following loops, indicate in which cases there is dependence between the two instructions given. Justify your answer.

a)

```
do i = 1, 100
      A(2*i) = ...
      ... = A(2*i+1)
enddo
```

SOL:
(1-0)/GCD(2,2) =1/2=0.5 is not integer => There are no dependencies.

b)

```
do i = 5, 100
      A(i-5) = ...
      ... = A(2*i+90)
enddo
```

SOL:
(90-(-5))/GCD(2,1) =95/1 = 95 is integer => There may be dependencies, but there are not because the write intervals are from A(0) to A(95) and the read interval is from A(100) to A(290).

c)

```
do i = 1, 100
      A(3*i+100) = ...
      ... = A(2*i-1)
enddo
```

SOL:
(-1-100)/GCD(3,2) =-101/1 = -101 is integer => There may be dependencies.
Intersection of the write intervals are from A(103) to A(400) and the read interval is from A(1) to A(199).
For write with i= 1 in A(103) depends on read with i =52 in A(103)

d)

```
do i = 1, 100
      A(6*i+3) = ...
      ... = A(3*i+81)
enddo
```

SOL:
(81-3)/GCD(6,3) =78/3 = 26 is integer => There may be dependencies.
Intersection of the write intervals are from A(9) to A(603) and the read interval is from A(84) to A(181).
For writing with i= 27 in A(165) depends on reading with i=28 in A(165)
In i=2 A(87) is read to be written in i= 14 A(87) anti-dependence

2.7.- Indicate how to parallelize in the most effective way the following loop for a machine with 40 processors. Write the pseudocode for the case of consecutive static scheduling and estimate the speedup that will be obtained.

```
do i = 0, 19
        do j = 0, 199
                A(i, j) = A(i, j) + 1
                B(2i+1, j) = B(2i, j) * 2
        enddo
enddo
```

Sol= The speedup will be 40 if the creation of threads is not taken into account.

The Loop is fully parallelizable

2.8.- Consider the following loop to be executed in a parallel machine:

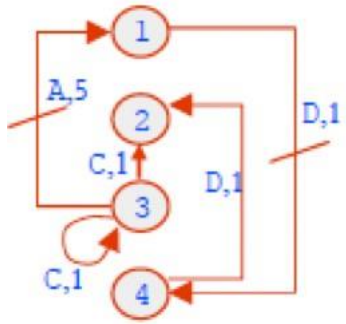
```
do i = 1, 24
     do j = 1, i
          (1) A(i,j) = B(i,j) + 1
          (2) C(i,j) = A(i-1,j) * 2
     enddo
enddo
enddo
```

a) Generate the dependency network and decide how to parallelize the code as efficiently as possible. Write the code corresponding to the case of a 4-processor system, where the task scheduling is dynamic by fixed-size chunks, 4 iterations in particular.
b) State the number of scheduling tasks required and make a reasoned estimate of the speed-up that can be achieved by executing the program in this way. Assume that the cost of executing an instruction such as those in the loop is 1, and that a synchronization operation has a cost of 10.

SOL: b → 6 planning tasks $S = 3$, efficiency = 0.75

2.9.- Given the following loop, indicate how to execute it in parallel in the most efficient way possible. Write the resulting code. If the execution time of each instruction is T, estimate the speedup and efficiency that will be achieved by using 10 processors. Justify the results.

```
do i = 2, 93
     (1) A(i) = D(i) - 10
     (2) B(i+1) = C(i-1) * D(i-2)
     (3) C(i) = C(i-1) + A(i+5)
     (4) D(i-1) = E(i+1) * 5
enddo
```




```
do i = 2, 93                          ; en serie (un procesador)
    C(i) = C(i-1) + A(i+5)
enddo

D(1) = E(3) * 5                       ; prólogo
B(3) = C(1) * D(0)
B(4) = C(2) * D(1)

doall i = 2, 91                       ; dos iteraciones menos
    A(i) = D(i) - 10
    D(i) = E(i+2) * 5
    B(i+3) = C(i+1) * D(i)
enddoall

A(92) = D(92) - 10                    ; epílogo
A(93) = D(93) - 10
D(92) = E(94) * 5
```

S = 368 T / 125 T = 2,94

2.10.- Given the following loop, generate the dependency graph and indicate how to parallelize the code without the need for synchronization (peeling.)

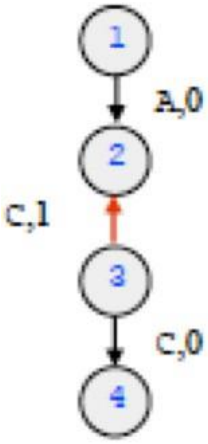
```
do i= 1, 198
        A(i) = B(i-1) * 3
        B(i+1) = C(i-1) + 1
        C(i) = C(i) * 0.5
        enddo
```

SOL:

```
B(2) = C(0) + 1
A(1) = B(0) * 3
A(2) = B(1) * 3
A(3) = B(2) * 3
doall i = 1, 195
   C(i) = C(i) * 0,5
   B(i+2) = C(i) + 1
   A(i+3) = B(i+2) * 3
enddo
C(196) = C(196) * 0,5
C(197) = C(197) * 0,5
B(198) = C(196) + 1
C(198) = C(198) * 0,5
B(199) = C(197) + 1
```

2.11.- Given the following loop, generate the dependency graph and indicate how to parallelize the code without synchronizing the processes.

```
do i= 1, 1000
        A(i) = A(i) +1
        B(i+1) = C(i)* A(i)
        C(i+1) = C(i+1) +D(i)
        F(i) =C(i+1)/F(i)
enddo
```




```
A(1) = A(1) + 1
B(2) = C(1) * A(1)

doall i = 1, 999
    A(i+1) = A(i+1) + 1
    C(i+1) = C(i+1) + D(i)
    B(i+2) = C(i+1) * A(i+1)
    F(i) = C(i+1) / F(i)
enddoall

C(1001) = C(1001) + D(1000)
F(1000) = C(1001) / F(1000)
```

2.12.- The following loop is executed on a 10-processor machine. Analyze how to parallelize the loop in the most effective way and write the code to be executed by each processor for the case of consecutive static scheduling.
Note: each processor is identified by the private variable pid = 0..9.

```
do i = 2, 101
        do j = 0, 199
                A(i,j) = A(i,j+2) * 2
                B(i,j) = B(i-2,j) * 3
        enddo
enddo
```

SOL

```
doall i = 2, 101
   do j = 0, 199
      A(i,j) = A(i,j+2) * 2
   enddo
enddoall

doall j = 0, 199
   do i = 2, 101
      B(i,j) = B(i-2,j) * 3
   enddo
enddoall
```

2.13.- The following loop is executed in a 4 processors parallel system:

```
do i = 1, 40
     <code>
enddo
```

All iterations are independent of each other and the execution time of the iterations is known to be T (i)) = i (i.e., first iteration 1, second iteration 2, etc.).
Calculates the parallel execution time of the program, and the speedup factor with respect to the serial case, if the task scheduling is:

a) 1. consecutive; and 2. intertwined.

b) Self-planning with: Z = 1 and Z = 5
2. guided self-planning, Zs = ceil [(N - i) / P].
3. trapezoidal, Z1 = 9, k = 1

Consider that the time of each dynamic planning operation is 1.

Sun:
a1 → *S*= 2.3 *efficiency* = 0.58
a2 → *S*= 3.8 *efficiency* = 0.94
b1 → *S* = 3,6 *efficiency* = 0,9 and S = 2,9 *efficiency* =
0,73 b2 → S = 3,6 efficiency = 0,9
b3 → *S* = 3,2 efficiency = 0,79

2.14.- 100 iterations of a certain loop are executed in parallel, between 4 processors, independently from each other.
The execution time of each iteration is 1 s, except for one of the iterations, which has an execution time of 20 s.
Calculate, approximately, the speed-up and efficiency achieved in the following cases:

(a) the distribution of iterations is static consecutive (with chunks of the same size).
(b) the distribution of iterations is static interleaved (with chunks of the same size).
(c) iteration allocation is dynamic, 1 to 1 (calculates worst case and best case); cost of a tasking operation, 50 ms.

SOL:

a) S= 119/44 = 2.7 efficiency 0.68

b) S= 119/44 = 2.7 efficiency 0.68

c) Best case S= 119/31 = 3.8 efficiency 0.96 Best case S= 119/45.25 = 2.6 efficiency 0.66