Exercícios - vetorização

Escrever uma função para multiplicação de matrizes.

Exemplo artificial: toda linguagem tem uma função (ou operador) para isso em alguma biblioteca (padrão ou não) .

Equivalente a (ex. IDL):

```
function pp_para_on_exe_1,mat1,mat2
sz1=size(mat1,/dimensions)
sz2=size(mat2,/dimensions)
if sz1[0] ne sz2[1] then begin
  message,'Second dimension of second array must match first dimension
of first array'
  return,ret
endif
ret=dblarr(sz2[0],sz1[1])
for i=0L,sz1[1]-1 do begin
 for j=0L,sz2[0]-1 do begin
   for k=0L,sz1[0]-1 do begin
     ret[j,i]+=mat1[k,i]*mat2[j,k]
   endfor
 endfor
endfor
return,ret
end
```

Uma solução em http://www.ppenteado.net/ast/pp_para_on/pp_para_on_sol_4.pdf

Exercícios – vetorização – solução

Exemplo de uso das soluções diferentes:

```
d0=n_elements(d0) eq 1 ? d0 : 100
d1=n_elements(d1) eq 1 ? d1 : 200
d2=n_elements(d2) eq 1 ? d2 : 300
a=dindgen(d0,d1)
b=dindgen(d2,d0)
t0=systime(/seconds)&ab0=pp_para_on_exe_0(a,b)&t1=systime(/seconds)
tt0=t1-t0
t0=systime(/seconds)&ab1=pp_para_on_exe_1(a,b)&t1=systime(/seconds)
tt1=t1-t0
t0=systime(/seconds)&ab2=pp_para_on_exe_2(a,b)&t1=systime(/seconds)
tt2=t1-t0
t0=systime(/seconds)&ab3=pp_para_on_exe_3(a,b)&t1=systime(/seconds)
tt3=t1-t0
print,'time with pp_para_on_exe_0: ',tt0
print,'time with pp_para_on_exe_1: ',tt1
print,'time with pp_para_on_exe_2: ',tt2
print,'time with pp_para_on_exe_3: ',tt3
IDL> @pp_para_on_exe_4
time with pp_para_on_exe_0:      0.024621010
time with pp_para_on_exe_1:        2.7856679
time with pp_para_on_exe_2:       0.23254895
time with pp_para_on_exe_3:       0.19247580
```

Usando o operador (##), para comparação.

Exercícios – vetorização – solução

Vetorização simples:

```
function pp_para_on_exe_2,mat1,mat2
sz1=size(mat1,/dimensions)
sz2=size(mat2,/dimensions)
if sz1[0] ne sz2[1] then begin
  message,'Second dimension of second array must match first dimension
of first array'
  return,ret
endif
ret=dblarr(sz2[0],sz1[1])
for i=0L,sz1[1]-1 do begin
 for j=0L,sz2[0]-1 do begin
   ret[j,i]=total(mat1[*,i]*mat2[j,*])
 endfor
endfor
return,ret
end
```

Exercícios – vetorização – solução

Vetorização (talvez desnecessariamente) complexa:

```
function pp_para_on_exe_3,mat1,mat2
sz1=size(mat1,/dimensions)
sz2=size(mat2,/dimensions)
if sz1[0] ne sz2[1] then begin
  message,'Second dimension of second array must match first dimension
of first array'
  return,ret
endif
tmp1=rebin(reform(transpose(mat1),1,sz1[1],sz1[0]),sz2[0],sz1[1],sz2[1])
tmp2=rebin(reform(mat2,sz2[0],1,sz2[1]),sz2[0],sz1[1],sz2[1])
ret=total(tmp1*tmp2,3)
return,ret
end
```

Exercícios – vetorização, OpenMP

Escrever uma função para multiplicação de matrizes.

Exemplo artificial: toda linguagem tem uma função (ou operador) para isso em alguma biblioteca (padrão ou não) .

Equivalente a (ex. Fortran):

```
module pp_para_on_exe_1
implicit none
interface pp_para_on_exe_1_f
  module procedure pp_para_on_exe_1_ff
end interface
contains
function pp_para_on_exe_1_ff(mat1,mat2) result(mat12)
implicit none
integer :: i,j,k,sz1(2),sz2(2)
double precision :: mat1(:,:),mat2(:,:)
double precision,allocatable :: mat12(:,:)
sz1=(/size(mat1,dim=1),size(mat1,dim=2)/)
sz2=(/size(mat2,dim=1),size(mat2,dim=2)/)
if (sz1(2) .ne. sz2(1)) then
  print *,'Second dimension of second array must match first dimension
of first array'
  return
endif
```

Exercícios – vetorização, OpenMP

(continuação)

```
allocate(mat12(sz1(1),sz2(2)))
do i=1,sz2(2)
  do j=1,sz1(1)
    do k=1,sz2(1)
      mat12(j,i)=mat12(j,i)+mat1(j,k)*mat2(k,i)
    enddo
  enddo
enddo
return
end function pp_para_on_exe_1_ff
```

Uma solução em http://www.ppenteado.net/ast/pp_para_on/pp_para_on_sol_4.pdf

Exercícios – vetorização, OpenMP - solução

Exemplo de uso das soluções diferentes:

```
program pp_para_on_exe_5
use pp_para_on_exe_4
!$ use omp_lib
implicit none
integer,parameter :: d0=1000,d1=500,d2=600
double precision :: a(d1,d0),b(d0,d2),ab0(d1,d2),ab1(d1,d2),ab3(d1,d2),ab4(d1,d2)
character(len=12) :: date(3)
integer :: t0(8), t1(8)
integer :: i,j,k

forall (i=1:d0,j=1:d1) a(j,i)=j-1+(i-1)*d1
forall (i=1:d2,j=1:d0) b(j,i)=j-1+(i-1)*d0

call date_and_time(date(1),date(2),date(3),t0)
call pp_para_on_exe_0_f(a,b,ab0)  ----------> matmul
call date_and_time(date(1),date(2),date(3),t1)
print *,'pp_para_on_exe_0_f: ',t1(5)*36d2+t1(6)*6d1+t1(7)+t1(8)*1d-3-
(t0(5)*36d2+t0(6)*6d1+t0(7)+t0(8)*1d-3)

call date_and_time(date(1),date(2),date(3),t0)
call pp_para_on_exe_1_f(a,b,ab1)  ----------> loops
call date_and_time(date(1),date(2),date(3),t1)
print *,'pp_para_on_exe_1_f: ',t1(5)*36d2+t1(6)*6d1+t1(7)+t1(8)*1d-3-
(t0(5)*36d2+t0(6)*6d1+t0(7)+t0(8)*1d-3)
```

Exercícios – vetorização, OpenMP - solução

(continuação)

```
call date_and_time(date(1),date(2),date(3),t0)
call pp_para_on_exe_3_f(a,b,ab3)
call date_and_time(date(1),date(2),date(3),t1)
print *,'pp_para_on_exe_3_f: ',t1(5)*36d2+t1(6)*6d1+t1(7)+t1(8)*1d-3-
(t0(5)*36d2+t0(6)*6d1+t0(7)+t0(8)*1d-3)
```

→ vetorização simples

```
call date_and_time(date(1),date(2),date(3),t0)
call pp_para_on_exe_4_f(a,b,ab4)
call date_and_time(date(1),date(2),date(3),t1)
print *,'pp_para_on_exe_4_f: ',t1(5)*36d2+t1(6)*6d1+t1(7)+t1(8)*1d-3-
(t0(5)*36d2+t0(6)*6d1+t0(7)+t0(8)*1d-3)
```

→ OpenMP

```
end program pp_para_on_exe_5
```

Exercícios – vetorização, OpenMP - solução

Exemplo de uso (d0=100,d1=4000,d2=300, Core 2 Duo, Compilador Intel)

```
[user@computer dir]$ ifort -g -traceback -openmp -O3
pp_para_on_exe_4.f90 pp_para_on_exe_5.f90 -o pp_para_on_exe_5.out

[user@computer dir]$ export OMP_NUM_THREADS=2

[user@computer dir]$ time ./pp_para_on_exe_5.out
 pp_para_on_exe_0_f:   0.165000000000000
 pp_para_on_exe_1_f:   0.147000000000000
 pp_para_on_exe_3_f:   0.144000000000000
 pp_para_on_exe_4_f:   9.80000000000000E-002

real    0m0.675s
user    0m0.659s
sys     0m0.035s
```

Exercícios – vetorização, OpenMP - solução

Exemplo de uso (d0=1000,d1=4000,d2=3000, 2 Xeon 6560 (12 núcleos), Compilador PGI)

```
[user@computer dir]$pgfortran -fast -Minfo=all -g -mp
pp_para_on_exe_4.f90 pp_para_on_exe_5.f90 -o pp_para_on_exe_5.out
pp_para_on_exe_4.f90:
pp_para_on_exe_1_f:
      8, Loop unrolled 2 times (completely unrolled)
      9, Loop unrolled 2 times (completely unrolled)
     16, Generated 2 alternate versions of the loop
         Generated vector sse code for the loop
         Generated a prefetch instruction for the loop
pp_para_on_exe_3_f:
     28, Loop unrolled 2 times (completely unrolled)
     29, Loop unrolled 2 times (completely unrolled)
     36, sum reduction inlined
         Generated 2 alternate versions of the loop
         Generated vector sse code for the loop
         Generated a prefetch instruction for the loop
pp_para_on_exe_0_f:
     46, Loop unrolled 2 times (completely unrolled)
     47, Loop unrolled 2 times (completely unrolled)
pp_para_on_exe_4_f:
     60, Loop unrolled 2 times (completely unrolled)
     61, Loop unrolled 2 times (completely unrolled)
     66, Parallel region activated
     68, Parallel loop activated with static block schedule
     70, sum reduction inlined
         Generated 2 alternate versions of the loop
         Generated vector sse code for the loop
         Generated a prefetch instruction for the loop
     73, Barrier
         Parallel region terminated
pp_para_on_exe_5.f90:
pp_para_on_exe_5:
     11, Loop not vectorized: mixed data types
         Unrolled inner loop 4 times
     12, Loop not vectorized: mixed data types
         Unrolled inner loop 4 times
```

Exercícios – vetorização, OpenMP - solução

Exemplo de uso (d0=1000,d1=4000,d2=3000, 2 Xeon 6560 (12 núcleos), Compilador PGI)

```
[user@computer dir]$ export OMP_NUM_THREADS=12
[user@computer dir]$penteado@gina-n2:~/pp_para_on> time
./pp_para_on_exe_5.out
 pp_para_on_exe_0_f:     6.362000000000990
 pp_para_on_exe_1_f:     125.0249999999942
 pp_para_on_exe_3_f:     124.9370000000054
 pp_para_on_exe_4_f:     10.92899999999645

real    4m27.297s
user    6m26.996s
sys     0m0.308s
```

Exercícios – vetorização, OpenMP - solução

Vetorização simples:

```
module pp_para_on_exe_4
contains

subroutine pp_para_on_exe_1_f(mat1,mat2,mat12)
implicit none
integer :: i,j,k,sz1(2),sz2(2)
double precision :: mat1(:,:),mat2(:,:),mat12(:,:)
sz1=(/size(mat1,dim=1),size(mat1,dim=2)/)
sz2=(/size(mat2,dim=1),size(mat2,dim=2)/)
if (sz1(2) .ne. sz2(1)) then
  print *,'Second dimension of second array must match first dimension
of first array'
  return
endif
do i=1,sz2(2)
  do j=1,sz1(1)
    do k=1,sz2(1)
      mat12(j,i)=mat12(j,i)+mat1(j,k)*mat2(k,i)
    enddo
  enddo
enddo
return
end subroutine pp_para_on_exe_1_f
```

Exercícios – vetorização, OpenMP - solução

(continuação)

```
subroutine pp_para_on_exe_3_f(mat1,mat2,mat12)
implicit none
integer :: i,j,sz1(2),sz2(2)
double precision :: mat1(:,:),mat2(:,:),mat12(:,:)
sz1=(/size(mat1,dim=1),size(mat1,dim=2)/)
sz2=(/size(mat2,dim=1),size(mat2,dim=2)/)
if (sz1(2) .ne. sz2(1)) then
  print *,'Second dimension of second array must match first dimension
of first array'
  return
endif
do i=1,sz2(2)
  do j=1,sz1(1)
      mat12(j,i)=sum(mat1(j,:)*mat2(:,i))
  enddo
enddo
return
end subroutine pp_para_on_exe_3_f
```

Exercícios – vetorização, OpenMP - solução

(continuação)

```
subroutine pp_para_on_exe_0_f(mat1,mat2,mat12)
implicit none
integer :: i,j,sz1(2),sz2(2)
double precision :: mat1(:,:),mat2(:,:),mat12(:,:)
sz1=(/size(mat1,dim=1),size(mat1,dim=2)/)
sz2=(/size(mat2,dim=1),size(mat2,dim=2)/)
if (sz1(2) .ne. sz2(1)) then
  print *,'Second dimension of second array must match first dimension
of first array'
  return
endif
mat12=matmul(mat1,mat2)
return
end subroutine pp_para_on_exe_0_f
```

Exercícios – vetorização, OpenMP - solução

(continuação)

```
subroutine pp_para_on_exe_4_f(mat1,mat2,mat12)
implicit none
integer :: i,j,sz1(2),sz2(2)
double precision :: mat1(:,:),mat2(:,:),mat12(:,:)
sz1=(/size(mat1,dim=1),size(mat1,dim=2)/)
sz2=(/size(mat2,dim=1),size(mat2,dim=2)/)
if (sz1(2) .ne. sz2(1)) then
  print *,'Second dimension of second array must match first dimension
of first array'
  return
endif
!$omp parallel shared(mat1,mat2,mat12,sz1,sz2) private(i,j)
!$omp do
do i=1,sz2(2)
  do j=1,sz1(1)
      mat12(j,i)=sum(mat1(j,:)*mat2(:,i))
  enddo
enddo
!$omp end do
!$omp end parallel
return
end subroutine pp_para_on_exe_4_f

end module pp_para_on_exe_4
```