# Exemples d utilisation de PIPS chez HPC Project

Khadija Immadouedine
Pierre Villalon

# Agenda

- Analyses de code C produit par Simulink
- Transformations de boucles et mesures de performances
- Optimisation du calcul de Stencil
- Conclusion

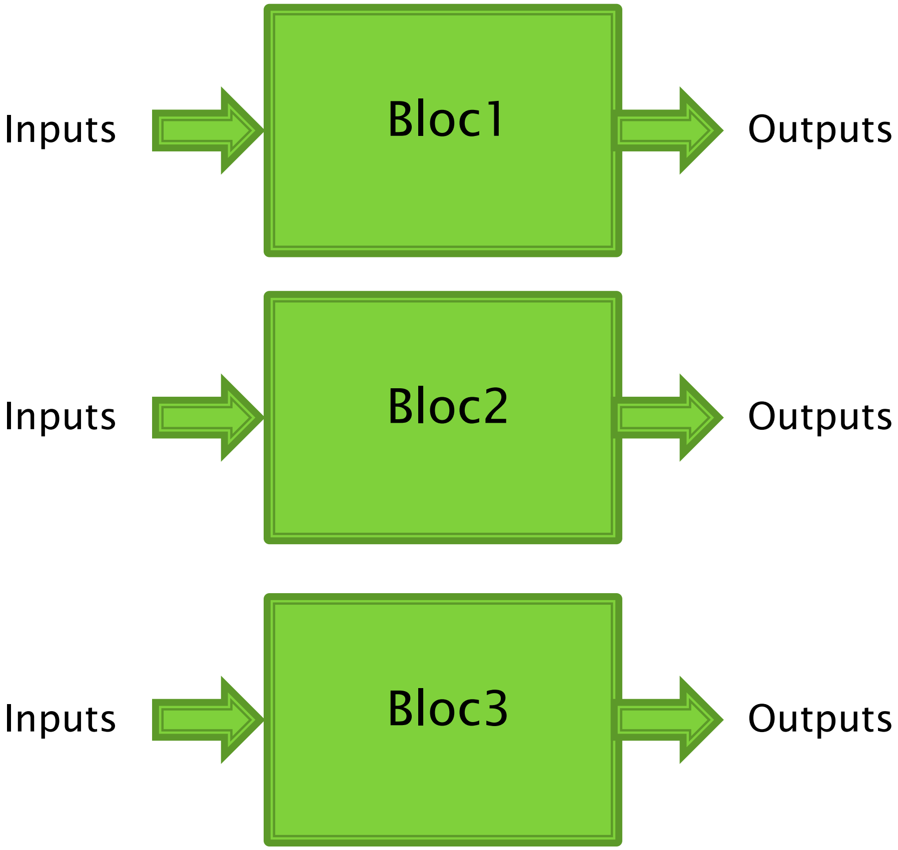
# Agenda

- Analyses de code C produit par Simulink
- Transformations de boucles et mesures de performances
- Optimisation du calcul de Stencil
- Conclusion

# Introduction

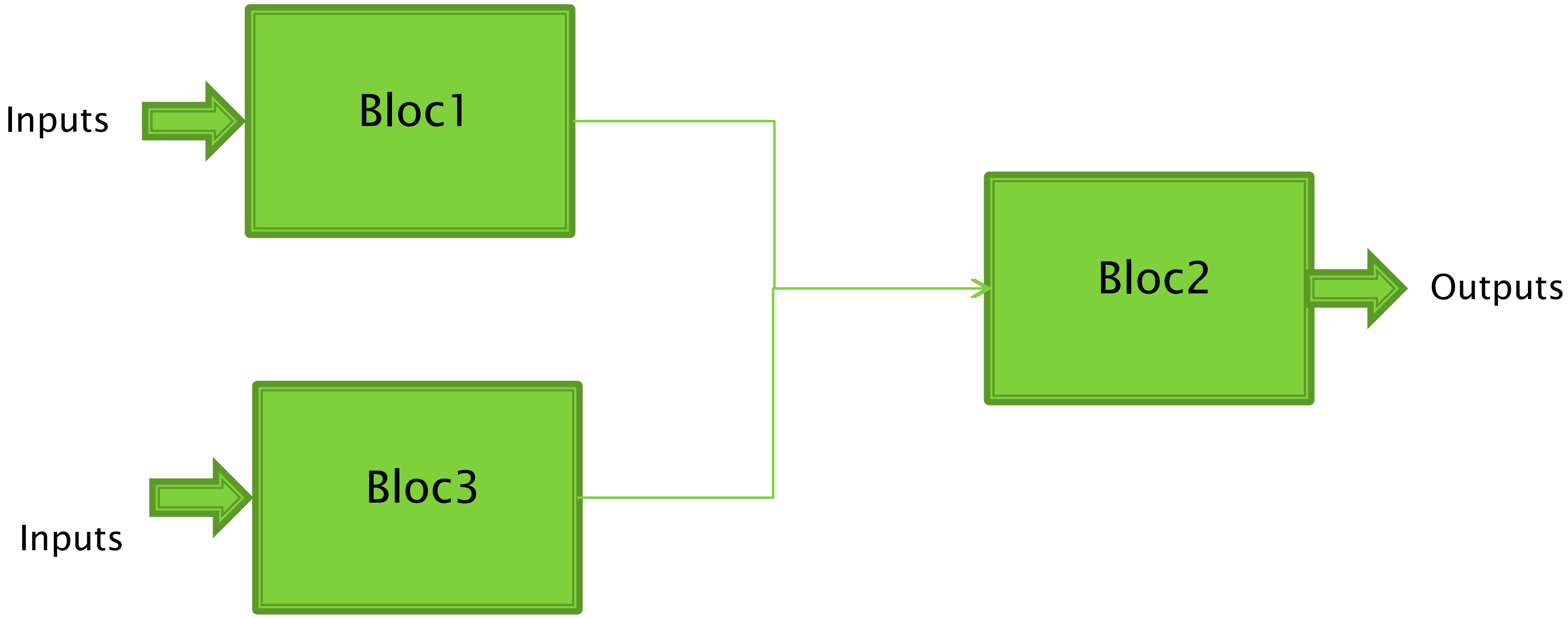
- HPC participe au développement d'une plateforme de simulation pour des applications temps réelle (Cobra).
  - Définition du simulateur via un GUI
  - Exécution distribué
  - Possibilité d interfacer le simulateur avec des capteurs physiques (« Hardware in the loop »)

# GUI



Inputs
Bloc1
Outputs
Inputs
Bloc2
Outputs
Inputs
Bloc3
Outputs

# GUI



# Basic Bloc


Bloc1.xml

```
<Signature>
  <Input>
    …
  </Input>
  <Output>
    …
  </Output>
</Signature>
```

Bloc1.c

```
Bloc1 () {
…
…
…
…

}
```

# Interfacage

- Cobra doit être une plateforme d acceuil -> il faut intégrer un écosystème:
  - Matlab/Simulink
  - Scilab/Xcos
  - Esterel/Scade
  - AdaCore

# Interfacage

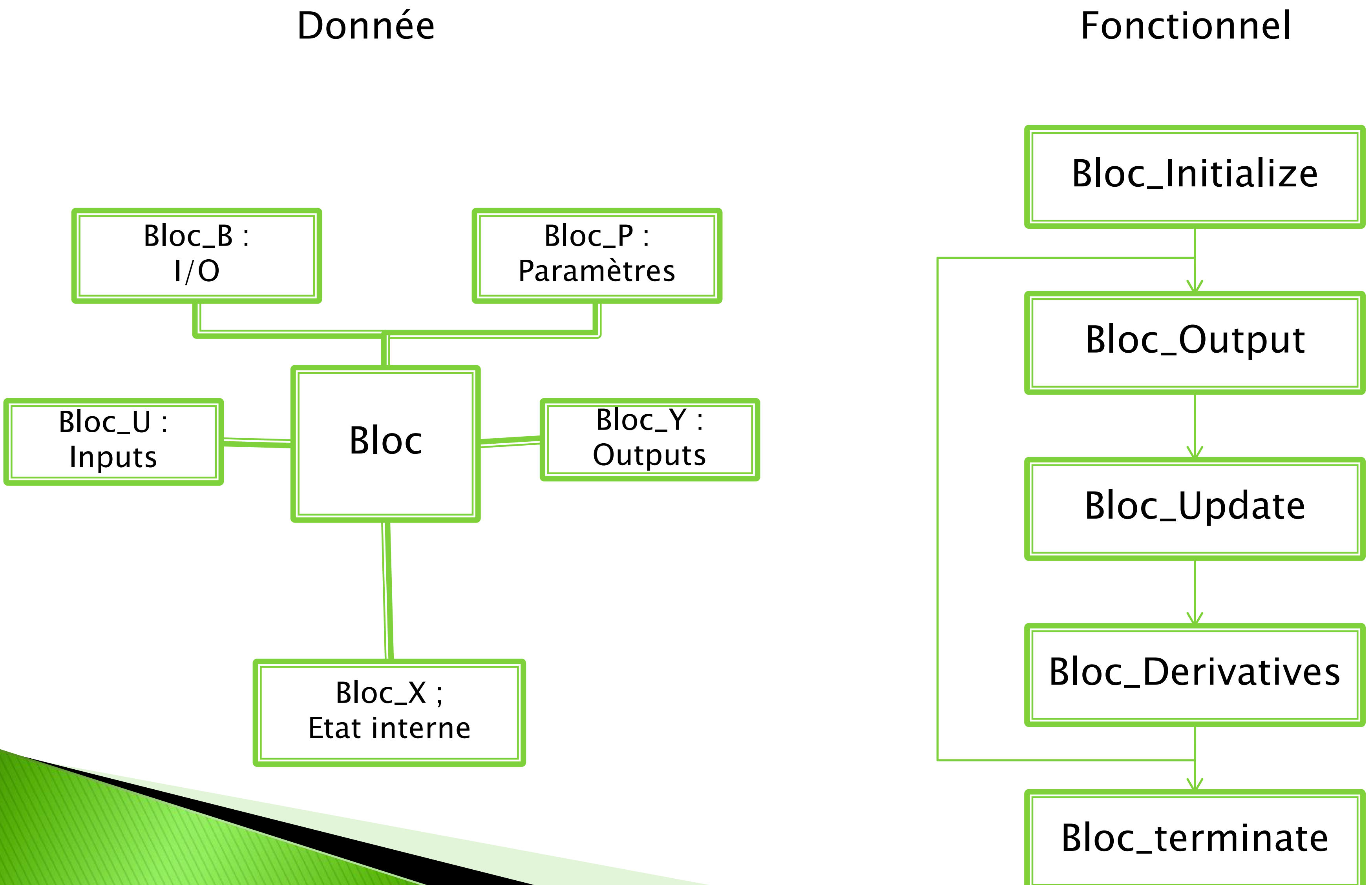
- Cobra doit être une plateforme d acceuil -> il faut intégrer un écosystème:
  - Matlab/Simulink
  - Scilab/Xcos
  - Esterel/Scade
  - AdaCore

# Interfacage

- Cobra doit être une plateforme d acceuil -> il faut intégrer un écosystème:
  - Matlab/Simulink
  - Scilab/Xcos
  - Esterel/Scade
  - AdaCore

Importation via PIPS

# Génération de code C



# Génération de code C

- Allocation statique des données ☺
- Un fichier C par Bloc (+bcp d'includes)

- Table des symboles -> Type des données
- Effects -> Input/Outputs des blocs

# Conclusion

- Embedded coder génère du code compatible avec PIPS
- PYPS fournit un langage de haut niveau pour l analyse (du code C) et la génération (C & xml)
- Tester sur 20 blocs pas de bugs de PIPS. (ca se complique avec « eliminate dead code »)

# Agenda

- Analyses de code C produit par Simulink
- Transformations de boucles et mesures de performances
  - icc VS manual opt VS PIPS
  - Interchange
  - Distribution
  - Unrolling
  - Tiling
  - Stripmining
- Optimisation du calcul de Stencil
- Conclusion

# Loop transformation

icc VS manual opt VS PIPS

- Intel Fortran Compiler 11.1
- Optimization Options: O0, O1, O2 and O3
- 3 tests:
  - program before optimization
  - program after manual optimization
  - program after optimization with PIPS

# Loop transformation

Interchange

```
subroutine interchange (n)
   integer n,i,j,k
   real a(1:n,1:n), b(1:n,1:n), c(1:n,1:n)

   !loop before loop interchange
   do 300 k=1,n
     do 200 j=1,n
       do 100 i=1,n
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
       100 continue
     200 continue
   300 continue
end
```

# Loop transformation

Interchange

```
subroutine interchange (n)
   integer n,i,j,k
   real a(1:n,1:n), b(1:n,1:n), c(1:n,1:n)

   !loop before loop interchange
   do 300 k=1,n
     do 200 j=1,n
       do 100 i=1,n
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
       100 continue
     200 continue
   300 continue
end
```

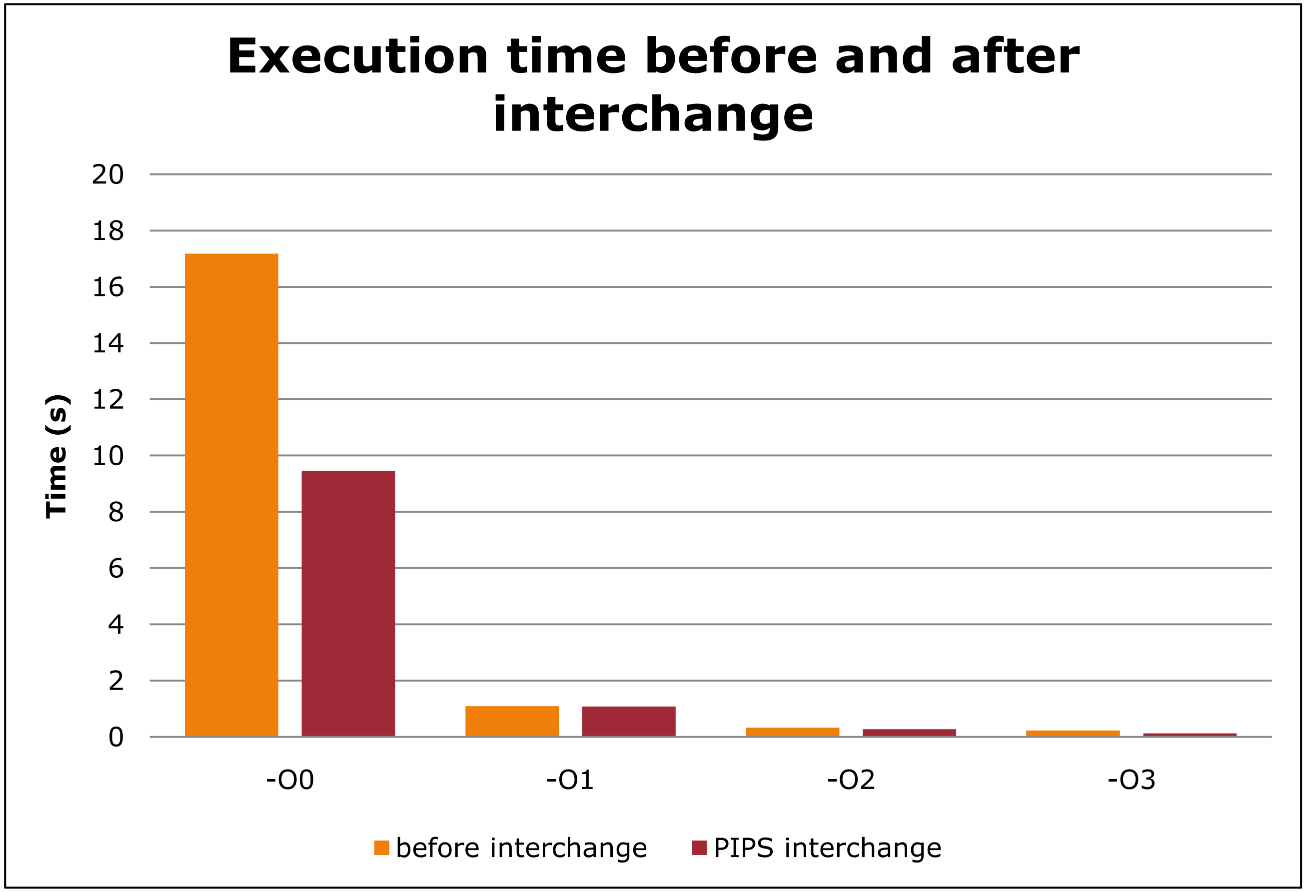
```
apply LOOP_INTERCHANGE
200
```

```
subroutine interchange (n)
   integer n,i,j,k
   real a(1:n,1:n), b(1:n,1:n), c(1:n,1:n)

   !loop after loop interchange
   do 300 j=1,n
     do 200 k=1,n
       do 100 i=1,n
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
       100 continue
     200 continue
   300 continue
end
```

# Loop transformation

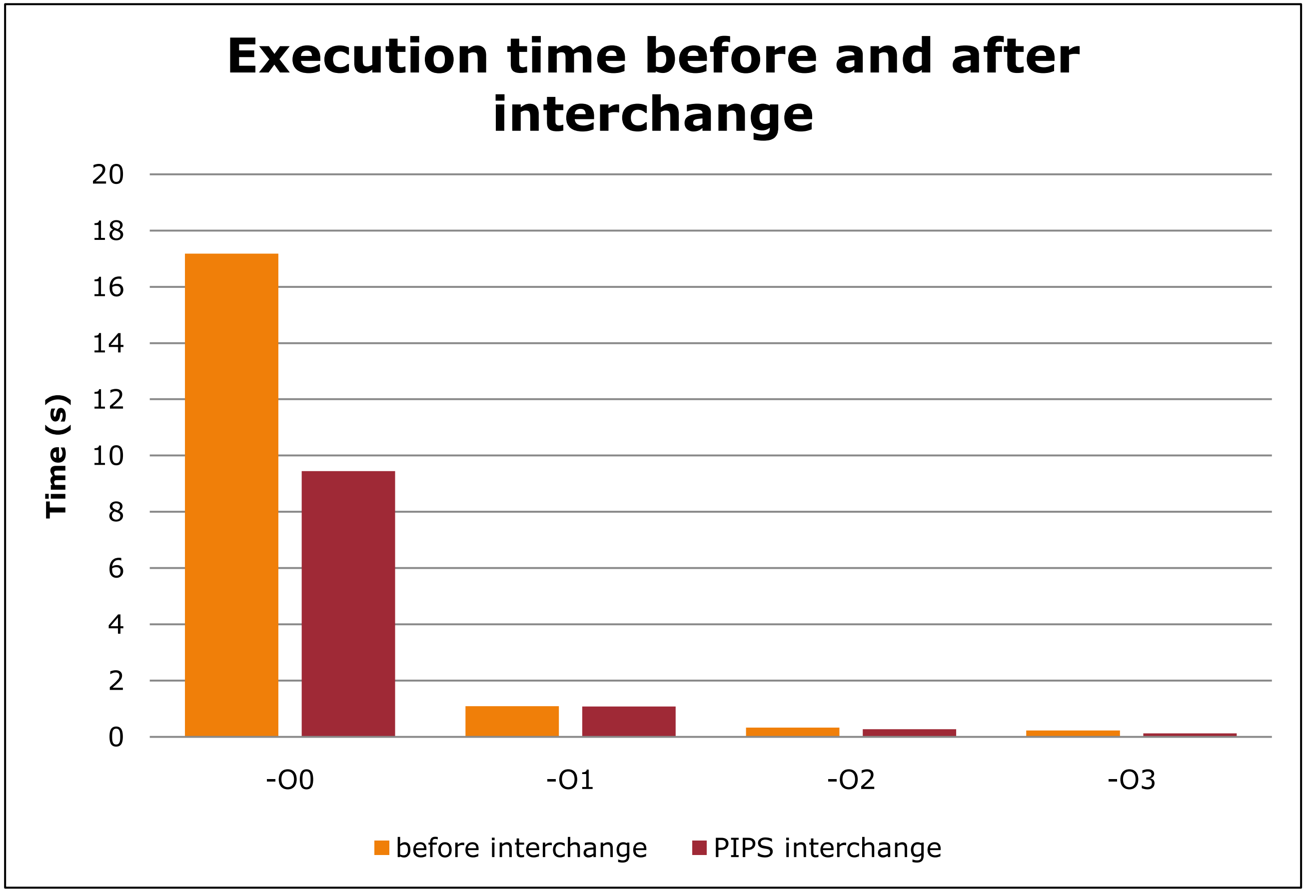
## Interchange




The lower, the better

# Loop transformation

Interchange




The lower, the better

Opportunity not detected by compiler

# Loop transformation

Distribution

```
subroutine distribution (n1,n2)
  integer :: n1, n2, i, j
  real, dimension (n1 ,n2) :: a, b
  real, dimension (n1) :: c, d
  ! Loop before distribution
  do i = 1, n1
    c(i) = i
    d(i) = i
    do j=1, n2
      a(i,j) = a(i,j) + b(i,j)*c(i)
    end do
  end do
end subroutine
```

distributio
n

interchang
e

```
subroutine distribution (n1 ,n2)
  integer :: n1 , n2 , i, j
  real , dimension (n1 ,n2) :: a, b
  real , dimension (n1) :: c, d
  ! Loop after manual distribution
  do i = 1, n1
    c(i) = i
    d(i) = i
  end do
  do j=1, n2
    do i=1, n1
      a(i,j) = a(i,j) + b(i,j)*c(i)
    end do
  end do
end subroutine
```

# Loop transformation

## Distribution

```
subroutine distribution (n1,n2)
  integer :: n1, n2, i, j
  real, dimension (n1 ,n2) :: a, b
  real, dimension (n1) :: c, d
  ! Loop before distribution
  do i = 1, n1
    c(i) = i
    d(i) = i
    do j=1, n2
      a(i,j) = a(i,j) + b(i,j)*c(i)
    end do
  end do
end subroutine
```

distribution →

interchange →

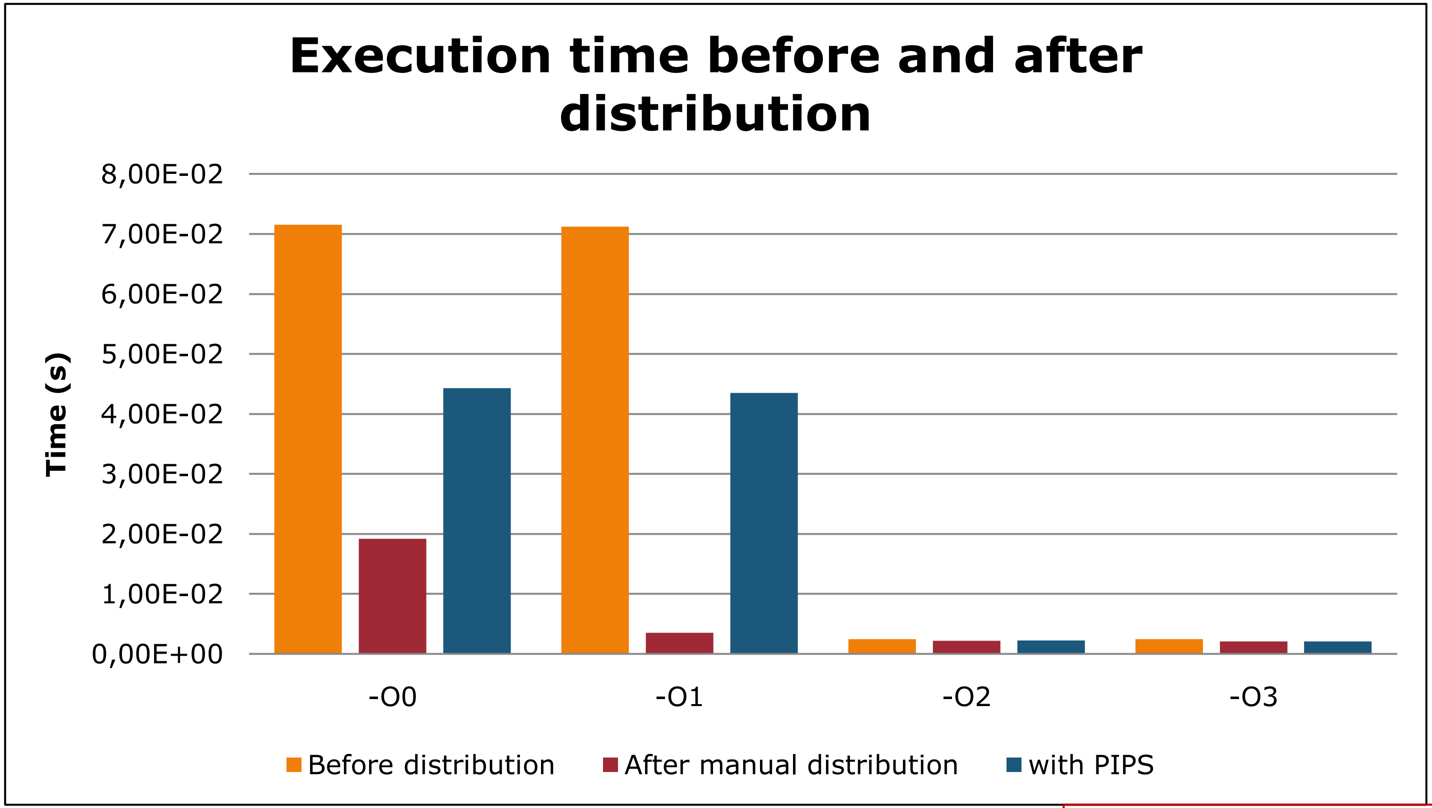
```
subroutine distribution (n1 ,n2)
  integer :: n1 , n2 , i, j
  real , dimension (n1 ,n2) :: a, b
  real , dimension (n1) :: c, d
  ! Loop after manual distribution
  do i = 1, n1
    c(i) = i
    d(i) = i
  end do
  do j=1, n2
    do i=1, n1
      a(i,j) = a(i,j) + b(i,j)*c(i)
    end do
  end do
end subroutine
```

PIPS

```
subroutine distribution (n1 ,n2)
  integer :: n1 , n2 , i, j
  real , dimension (n1 ,n2) :: a, b
  real , dimension (n1) :: c, d
  ! Loop after distribution with PIPS
  do i = 1, n1
    d(i) = i
  end do
  do i = 1, n1
    c(i) = i
    do j=1, n2
      a(i,j) = a(i,j) + b(i,j)*c(i)
    end do
  end do
end subroutine
```

# Loop transformation

Distribution




**The lower, the better**

More thorough distribution,
More optimization

# Loop transformation

Unrolling

```
subroutine unroll (n1 ,n2)
  integer :: n1,n2,i,j
  real, dimension (0:n1+1,0:n2+1) :: a,b,c

  ! Loop before unrolling
  do j=1, n2
     do i=1, n1
        a(i,j)=a(i+1,j)*b(i,j)+a(i,j +1)*c(i,j)
     end do
  end do
end
```

# Loop transformation

Unrolling

```
subroutine unroll (n1 ,n2)

  integer :: n1 , n2 , i, j, k
  real , dimension (0: n1 +1 ,0: n2 +1) :: a, b, c
  ! Loop after manual unrolling
  K = mod (n2 ,4)
  do j=1,n2 -K ,4
    do i=1, n1
      a(i,j ) = a(i+1,j )*b(i,j ) + a(i,j +1)* c(i,j )
      a(i,j +1) = a(i+1,j +1)* b(i,j +1) + a(i,j +2)* c(i,j +1)
      a(i,j +2) = a(i+1,j +2)* b(i,j +2) + a(i,j +3)* c(i,j +2)
      a(i,j +3) = a(i+1,j +3)* b(i,j +3) + a(i,j +4)* c(i,j +3)
    end do
  end do

  ! post conditioning part of loop
  do j= n2 -K+2, n2 , 4
    do i=1, n1
      a(i,j) = a(i+1,j)*b(i,j) + a(i,j +1)* c(i,j)
    end do
  end do
end
```

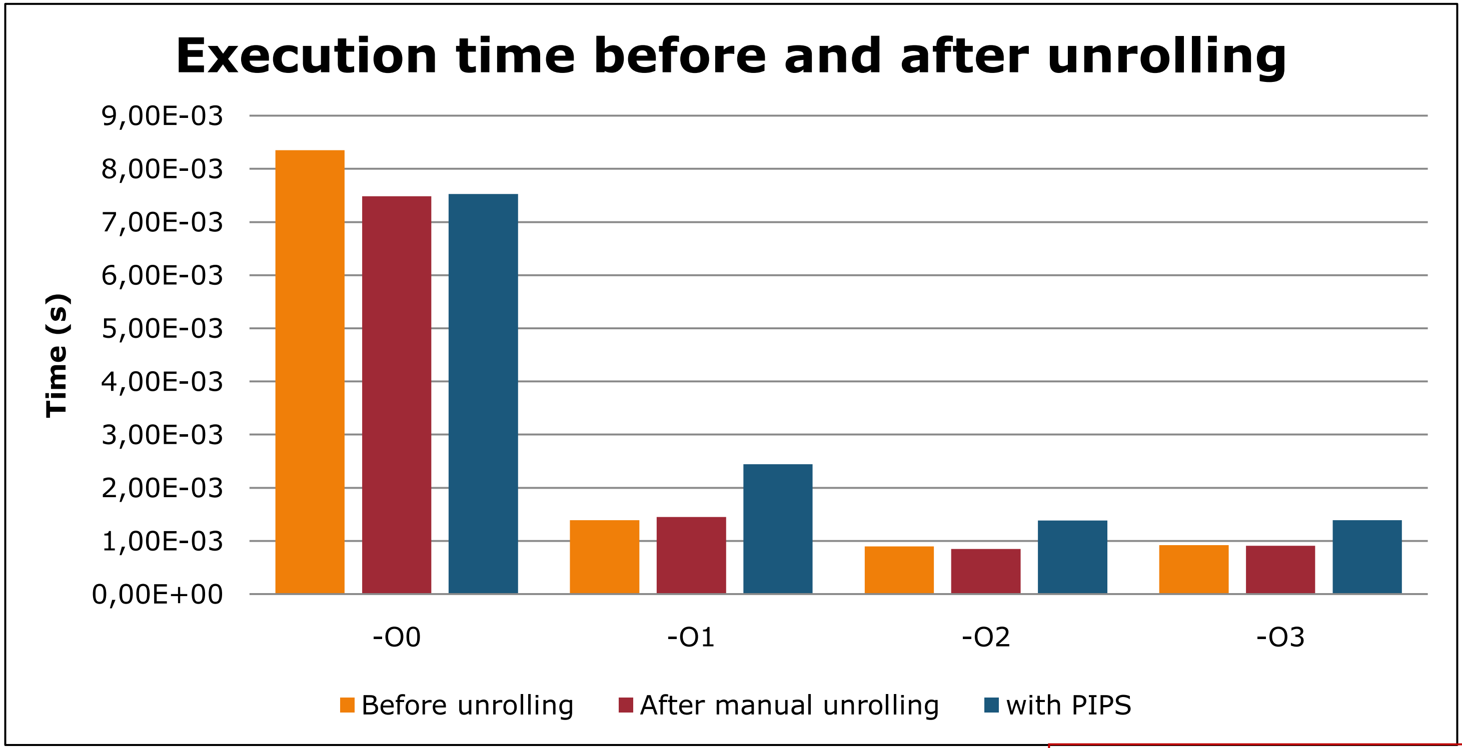
# Loop transformation

Unrolling

```
subroutine unroll (n1 ,n2)
  integer :: n1 , n2 , n3 , i, j
  integer :: LU_NUB0 , LU_IB0 , LU_IND0
  real , dimension (0: n1 +1 ,0: n2 +1) :: a, b, c
  ! Loop after unrolling with PIPS
  DO 200 J = 1, N2
    LU_NUB0 = (N1 -1+1)/1
    LU_IB0 = MOD ( LU_NUB0 , 4)
    DO 99999 LU_IND0 = 0, LU_IB0 -1
    A(LU_IND0 *1+1 ,J) = A( LU_IND0 *1+1+1 , J)*B( LU_IND0 *1+1 ,J)+A(&
    LU_IND0 *1+1 ,J +1)* C( LU_IND0 *1+1 ,J)
    99999 CONTINUE
    DO 99998 LU_IND0 = LU_IB0 , LU_NUB0 -1, 4
    A(( LU_IND0 +0)*1+1 , J) = A(( LU_IND0 +0)*1+1+1 , J)*B(( LU_IND0 + &
    0)*1+1 , J)+A(( LU_IND0 +0)*1+1 , J +1)* C(( LU_IND0 +0)*1+1 , J)
    A(( LU_IND0 +1)*1+1 , J) = A(( LU_IND0 +1)*1+1+1 , J)*B(( LU_IND0 + &
    1)*1+1 , J)+A(( LU_IND0 +1)*1+1 , J +1)* C(( LU_IND0 +1)*1+1 , J)
    A(( LU_IND0 +2)*1+1 , J) = A(( LU_IND0 +2)*1+1+1 , J)*B(( LU_IND0 + &
    2)*1+1 , J)+A(( LU_IND0 +2)*1+1 , J +1)* C(( LU_IND0 +2)*1+1 , J)
    A(( LU_IND0 +3)*1+1 , J) = A(( LU_IND0 +3)*1+1+1 , J)*B(( LU_IND0 + &
    3)*1+1 , J)+A(( LU_IND0 +3)*1+1 , J +1)* C(( LU_IND0 +3)*1+1 , J)
    99998 CONTINUE
   I = 1+ MAX0 ( LU_NUB0 , 0)*1
  200 CONTINUE
end
```

# Loop transformation

Unrolling




The lower, the better

Unrolling following the 2nd dimension rather than the 1st.

# Loop transformation

Tiling

```
subroutine tiling (n1,n2,n3)
  integer :: n1,n2,n3,i,j,k
  real, dimension (n1+1,n2+1,n3+1) :: a,b,c

  ! Loop before tiling



  do k=1, n3
    do j=1, n2
      do i=1, n1
        a(i,j,k) = &
          a(i,j,k+1)*b(i,j,k)+&
          a(i+1,j,k)*c(i,j,k)+&
          a(i,j+1,k)*a(i,j,k)
      end do
    end do
  end do



end
```

# Loop transformation

Tiling

```
subroutine tiling (n1,n2,n3)
  integer :: n1,n2,n3,i,j,k
  real, dimension (n1+1,n2+1,n3+1) :: a,b,c

  ! Loop after manual tiling

  do v=1, n3 ,20
    do u=1, n2 ,20
  do k=v, v+19
    do j=u, u+19
      do i=1, n1
        a(i,j,k) = &
          a(i,j,k+1)*b(i,j,k)+&
          a(i+1,j,k)*c(i,j,k)+&
          a(i,j+1,k)*a(i,j,k)
      end do
    end do
  end do
    end do
  end do

end
```

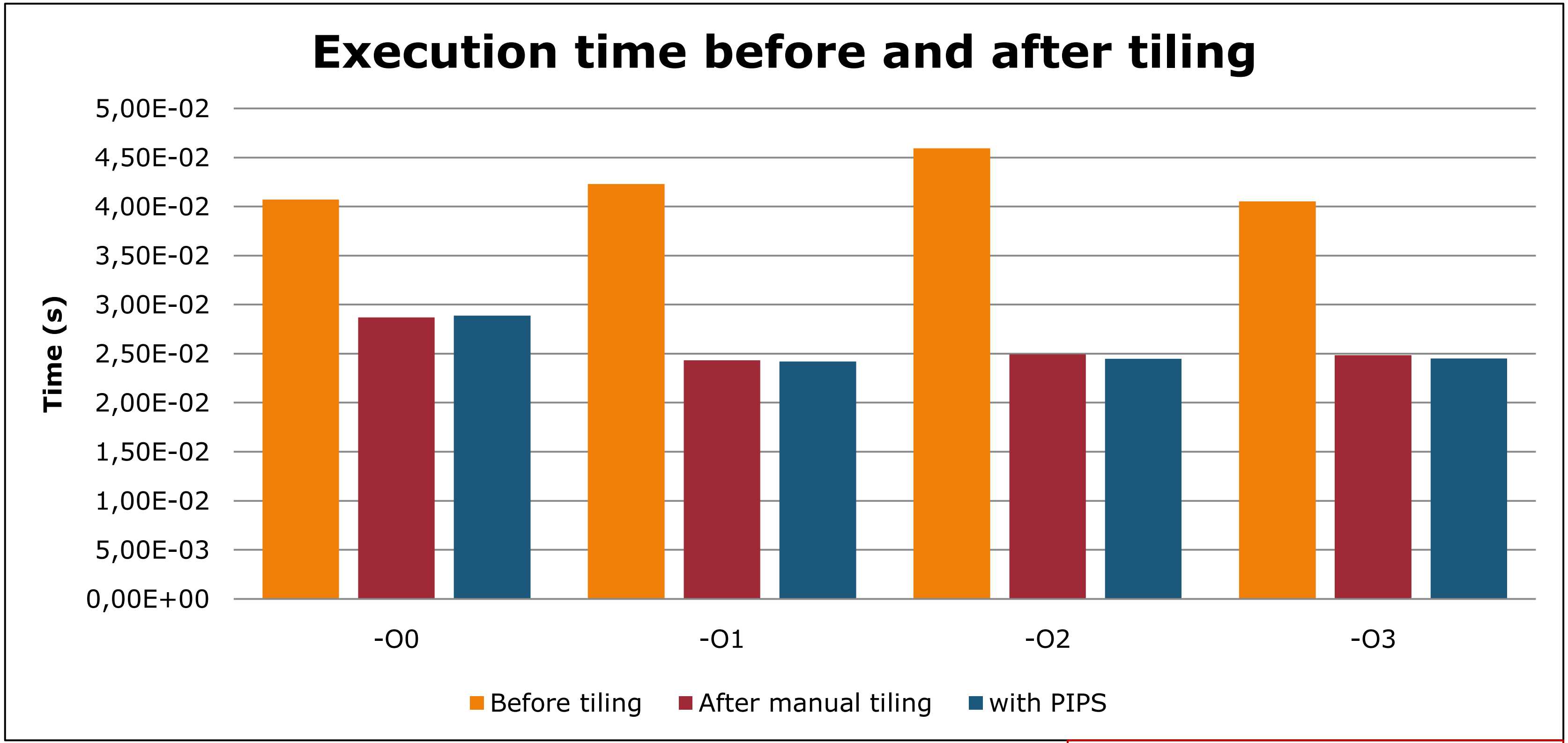
```
subroutine tiling (n1,n2,n3)
  integer :: n1,n2,n3,i,j,k
  real, dimension (n1+1,n2+1,n3+1) :: a,b,c

  ! Loop after tiling with PIPS
  DO K_t = 0, (N3 -1)/20
    DO J_t = 0, (N2 -1)/20
      DO I_t = 0, N1 -1
  DO K = 20* K_t+1, MIN (20*K_t+20,N3)
    DO J = 20* J_t +1, MIN (N2,20*J_t+20)
      DO I = I_t +1, I_t +1
        a(i,j,k) = &
          a(i,j,k+1)*b(i,j,k)+&
          a(i+1,j,k)*c(i,j,k)+&
          a(i,j+1,k)*a(i,j,k)
      END DO
    END DO
  END DO
      END DO
    END DO
  END DO
end
```

```
apply LOOP_TILING
100 # Loop label
# Tiling matrix
20 0 0 # tile size for the outer loop
0 20 0 # tile size for the middle loop
0  0 1 # tile size for the inner loop
```

# Loop transformation

## Tiling




The lower, the better

# Loop transformation

Stripmining

```
subroutine stripmining (n1 ,n2 ,n3)
  integer :: n1 , n2 , n3 , i, j, k
  real, dimension (0:n1+1,0:n2+1,0:n3+1)::a,b

  ! Loop before strip - mining




  do 100 k=1, n3
    do 200 i=1, n2
      do 300 j=1, n1
        a(j,i,k) = a(j-1,i,k)+&
          b(i,j,k)*b(j,i,k)
      300 continue
    200 continue
  100 continue




end
```

# Loop transformation

Stripmining

```
subroutine stripmining (n1 ,n2 ,n3)
  integer :: n1 , n2 , n3 , i, j, k
  real, dimension (0:n1+1,0:n2+1,0:n3+1)::a,b

  ! Loop after manual strip - mining
  do k_1 = 1, n3 , 10
    do i_1 = 1, n2 , 10
      do j_1 = 1, n1 , 10
  do k = k_1 , min (k_1 +9, n3)
    do i = i_1 , min (i_1 +9, n2)
      do j = j_1 , min (j_1 +9, n1)
        a(j,i,k) = a(j-1,i,k)+&
          b(i,j,k)*b(j,i,k)
      enddo
    enddo
  enddo
      enddo
    enddo
  enddo
end
```

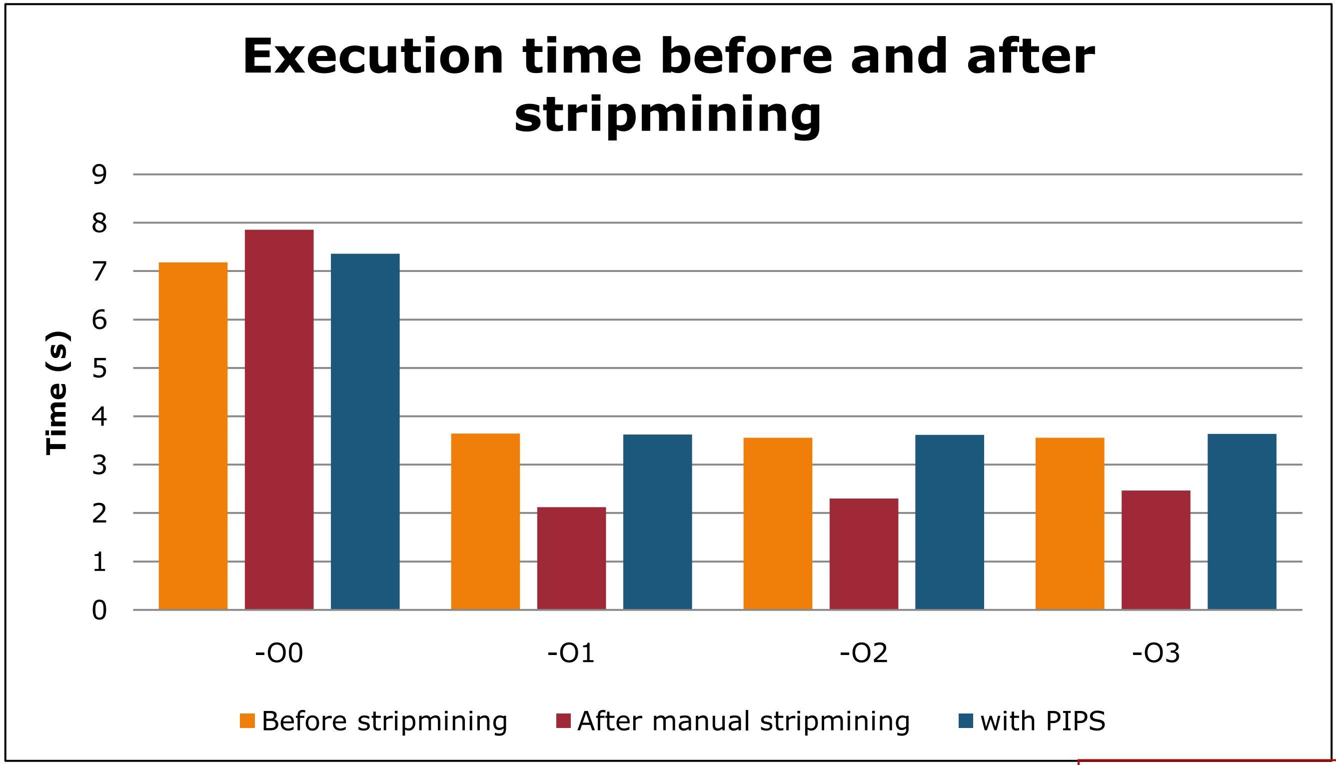
```
subroutine stripmining (n1 ,n2 ,n3)
  integer :: n1 , n2 , n3 , i, j, k
  real, dimension (0:n1+1,0:n2+1,0:n3+1)::a,b

  ! Loop after strip - mining with PIPS
  DO K_1 = 1, N3 , 10
  DO K = K_1 , MIN (K_1 +9, N3)
    DO I_1 = 1, N2 , 10
    DO I = I_1 , MIN (I_1 +9, N2)
      DO J_1 = 1, N1 , 10
      DO J = J_1 , MIN (J_1 +9, N1)
        A(J,I,K) = A(J-1,I,K)+&
          B(I,J,K)*B(J,I,K)
      ENDDO
      ENDDO
    ENDDO
    ENDDO
  ENDDO
  ENDDO
end
```

```
apply STRIP_MINE
100 # Loop label
0
10 # slice size
apply STRIP_MINE
200 # Loop label
0
10 # slice size
apply STRIP_MINE
300 # Loop label
0
10 # slice size
```

# Loop transformation

Stripmining




**The lower, the better**

Loop partitioning more effective for all the loops.

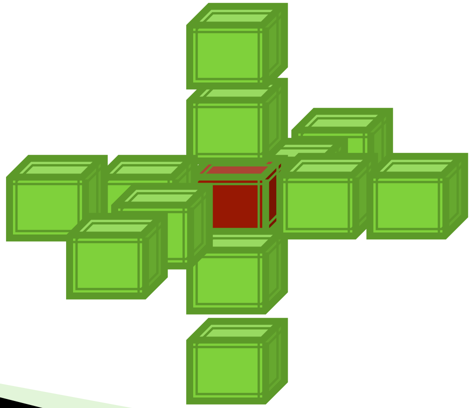
# Agenda

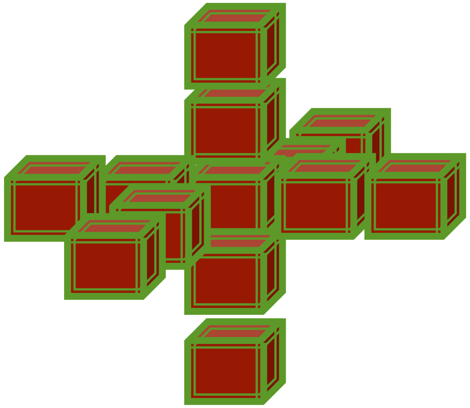
- Analyses de code C produit par Simulink
- Transformations de boucles et mesures de performances
- Optimisation du calcul de Stencil
- Conclusion

# Stencil Optimization

Stencil ?

```
do k = ks ,ke
 do j = js ,je
  do i = is ,ie
    b(i,j,k)=c_2*(a(i-2,j,k)+a(i,j-2,k)+a(i,j,k-2))&
            +c_1*(a(i-1,j,k)+a(i,j-1,k)+a(i,j,k-1))&
            +c0 * a(i,j,k)*3&
            +c1 *(a(i+1,j,k)+a(i,j+1,k)+a(i,j,k+1))&
            +c2 *(a(i+2,j,k)+a(i,j+2,k)+a(i,j,k+2))
  end do
 end do
end do
```





# Stencil Optimization

Tiling

```
do i3_t = 5/ bs3 , (ie3 -4)/ bs3
  do i2_t = 5/ bs2 , (ie2 -4)/ bs2
    do i1_t = 5/ bs1 , (ie1 -4)/ bs1

do i3 = max (bs3*i3_t , 5), min ( bs3 * i3_t +bs3 -1, ie3 -4)
  do i2 = max (bs2*i2_t , 5), min ( bs2 * i2_t +bs2 -1, ie2 -4)
    do i1 = max (bs1*i1_t , 5), min ( bs1 * i1_t +bs1 -1, ie1 -4)

      u(i1 ,i2 ,i3) =&
            c_2 *(v(i1-2,i2,i3) + v(i1,i2-2,i3) + v(i1,i2,i3-2))&
          + c_1 *(v(i1-1,i2,i3) + v(i1,i2-1,i3) + v(i1,i2,i3-1))&
          + c0  * v(i1 , i2,i3)*3 &
          + c1  *(v(i1+1,i2,i3) + v(i1,i2+1,i3) + v(i1,i2,i3+1))&
          + c2  *(v(i1+2,i2,i3) + v(i1,i2+2,i3) + v(i1,i2,i3+2))
    enddo
  enddo
enddo

    enddo
  enddo
enddo
```

# Stencil Optimization

Tiling

```
do i3_t = 5/ bs3 , (ie3 -4)/ bs3
  do i2_t = 5/ bs2 , (ie2 -4)/ bs2
    do i1_t = 5/ bs1 , (ie1 -4)/ bs1

do i3 = max (bs3*i3_t , 5), min ( bs3 * i3_t +bs3 -1, ie3 -4)
  do i2 = max (bs2*i2_t , 5), min ( bs2 * i2_t +bs2 -1, ie2 -4)
    do i1 = max (bs1*i1_t , 5), min ( bs1 * i1_t +bs1 -1, ie1 -4)

      u(i1 ,i2 ,i3) =&
            c_2 *(v(i1-2,i2,i3) + v(i1,i2-2,i3) + v(i1,i2,i3-2))&
          + c_1 *(v(i1-1,i2,i3) + v(i1,i2-1,i3) + v(i1,i2,i3-1))&
          + c0  * v(i1 , i2,i3)*3 &
          + c1  *(v(i1+1,i2,i3) + v(i1,i2+1,i3) + v(i1,i2,i3+1))&
          + c2  *(v(i1+2,i2,i3) + v(i1,i2+2,i3) + v(i1,i2,i3+2))
    enddo
  enddo
enddo

    enddo
  enddo
enddo
```

```
apply LOOP_TILING
100 # Loop label
# Tiling matrix
bs3 0   0
0   bs2 0
0   0   bs1
```

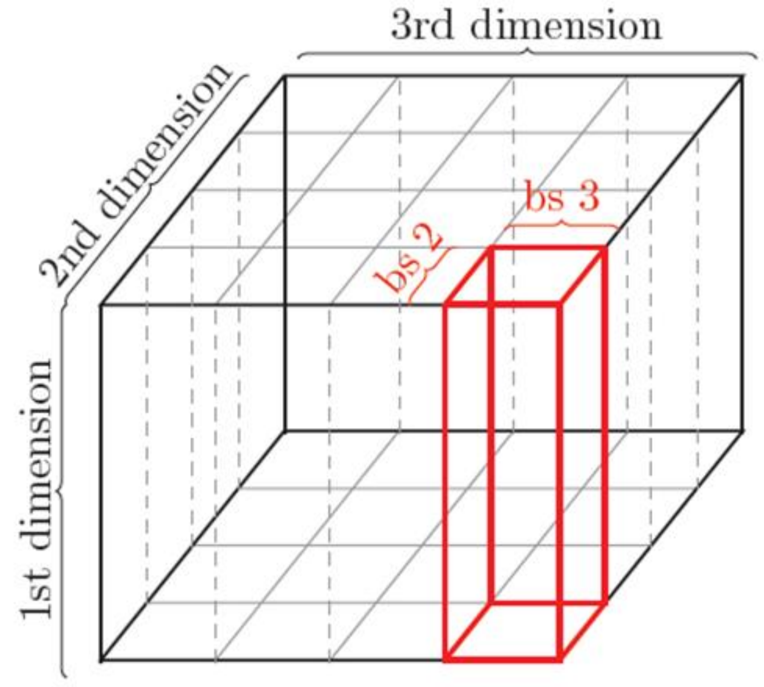
# Stencil Optimization

## Tiling

-the 1st dimension size is ~$10^3$

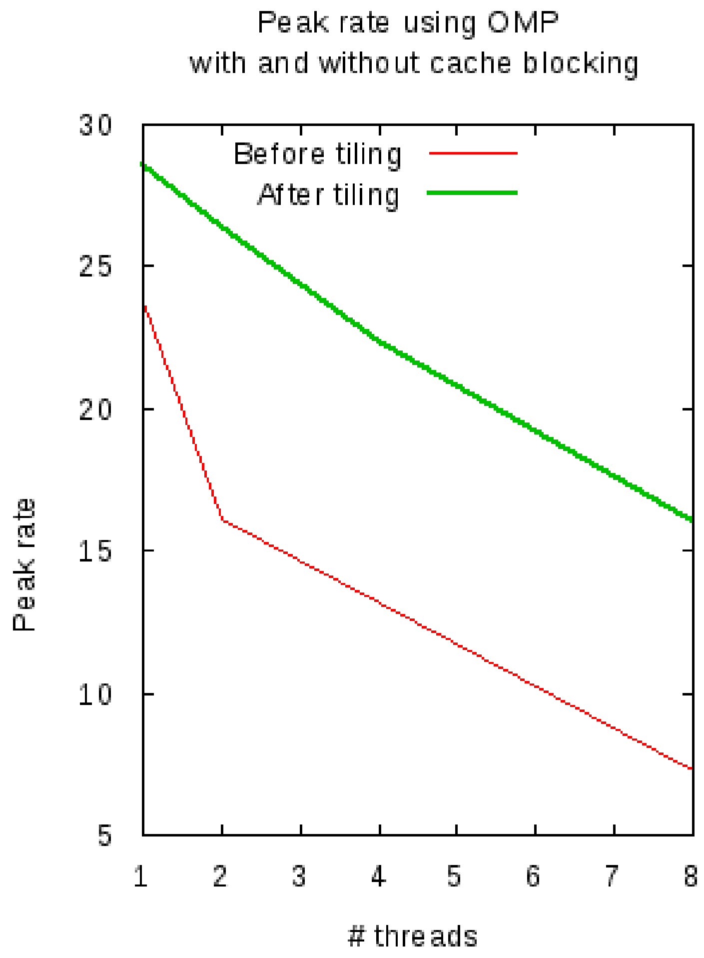
-the $2^{nd}$ dimension size is ~$10^2$

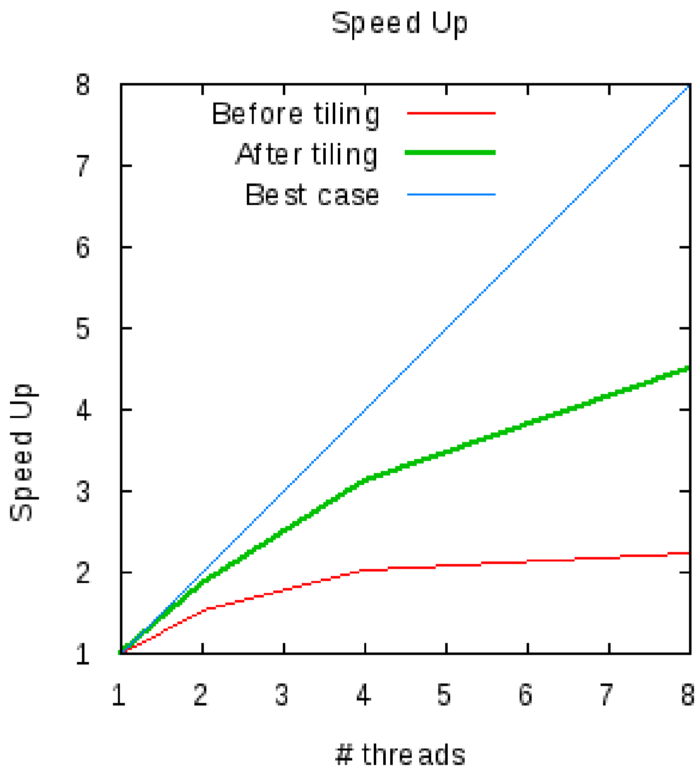
-the 3rd dimension size does not matter
(a special case for stencil's size blocks)



# Stencil Optimization

Tiling & OMP




The higher, the better

# Agenda

- Analyses de code C produit par Simulink
- Transformations de boucles et mesures de performances
- Optimisation du calcul de Stencil
- Conclusion