TRAINING'S PROGRAM

- Day 1: HPC basics / *Olympe* cluster insight
  - 09h00 - 10h30 : HPC systems architecture / cluster room visit
  - 10h45 - 12h15 : HPC programming techniques
    a) Code tuning

MATRIX MULTIPLICATION

$$C = A \times B \qquad A, B, C \in \mathcal{M}(n,n)$$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = a_{i:} \cdot b_{:j}$$

MATRIX MULTIPLICATION

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$C = A \times B$ $\qquad A, B, C \in \mathcal{M}(n,n)$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = a_{i:} \cdot b_{:j}$$

MATRIX MULTIPLICATION

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$C = A \times B \qquad A, B, C \in \mathcal{M}(n,n)$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$c_{ij} = a_{i:} \cdot b_{:j}$

- Floating point operations (Flop)
  - three nested loops of size $n = 1024$
  - two operations per instruction

MATRIX MULTIPLICATION

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$C = A \times B \qquad A, B, C \in \mathcal{M}(n,n)$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$c_{ij} = a_{i:} \cdot b_{:j}$

- Floating point operations (Flop)
  - three nested loops of size $n = 1024$
  - two operations per instruction

$\Rightarrow 2n^3$ Flop, runs in 28 s

$\Rightarrow 2 \times 1024^3$ Flop/28 s $= \sim 77$ MFlop/s

MATRIX MULTIPLICATION

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77 MFlop/s$

Olympe cores peak performance is $73{,}6\ GFlop/s$!

$C = A \times B \qquad A, B, C \in \mathcal{M}(n,n)$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$c_{ij} = a_{i:} \cdot b_{:j}$

- Floating point operations (Flop)
  - three nested loops of size $n = 1024$
  - two operations per instruction

$\Rightarrow 2n^3$ Flop, runs in $28\ \mathrm{s}$

$\Rightarrow 2 \times 1024^3\ \mathrm{Flop}/28\ \mathrm{s} = \sim 77\ \mathrm{MFlop/s}$

MATRIX MULTIPLICATION : UNROLLING

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77 MFlop/s$

```
do i = 1, n
  do j = 1, n
    do k = 1, n, 8
      s0 = A(i,k) * B(k,j)
      s1 = A(i,k+1) * B(k+1,j)
      s2 = A(i,k+2) * B(k+2,j)
      s3 = A(i,k+3) * B(k+3,j)
      s4 = A(i,k+4) * B(k+4,j)
      s5 = A(i,k+5) * B(k+5,j)
      s6 = A(i,k+6) * B(k+6,j)
      s7 = A(i,k+7) * B(k+7,j)
      C(i,j) = C(i,j) + (s0+s1+...+s7)
    end do
  end do
end do
```

MATRIX MULTIPLICATION : UNROLLING

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77 MFlop/s$

```
do i = 1, n
  do j = 1, n
    do k = 1, n, 8
      s0 = A(i,k) * B(k,j)
      s1 = A(i,k+1) * B(k+1,j)
      s2 = A(i,k+2) * B(k+2,j)
      s3 = A(i,k+3) * B(k+3,j)
      s4 = A(i,k+4) * B(k+4,j)
      s5 = A(i,k+5) * B(k+5,j)
      s6 = A(i,k+6) * B(k+6,j)
      s7 = A(i,k+7) * B(k+7,j)
      C(i,j) = C(i,j) + (s0+s1+...+s7)
    end do
  end do
end do
```

$\sim 109\ MFlop/s$

MATRIX MULTIPLICATION : UNROLLING

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77\ MFlop/s$

```
do i = 1, n
  do j = 1, n
    do k = 1, n, 8
      s0 = A(i,k) * B(k,j)
      s1 = A(i,k+1) * B(k+1,j)
      s2 = A(i,k+2) * B(k+2,j)
      s3 = A(i,k+3) * B(k+3,j)
      s4 = A(i,k+4) * B(k+4,j)
      s5 = A(i,k+5) * B(k+5,j)
      s6 = A(i,k+6) * B(k+6,j)
      s7 = A(i,k+7) * B(k+7,j)
      C(i,j) = C(i,j) + (s0+s1+...+s7)
    end do
  end do
end do
```

$\sim 109\ MFlop/s$

Iterations

ld a				
ld b	ld a			
add	ld b	ld a		
st a	add	ld b	ld a	
	st a	add	ld b	ld a
		st a	add	ld b
			st a	add
				st a

Cycle (time)

Unrolling ⇒ pipeline

(instruction level parallelism)

MATRIX MULTIPLICATION : BLOCKING

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77\ MFlop/s$

(b = 16)

```
do i = 1, n, b
  do j = 1, n, b
    do k = 1, n, b
      do ii = i, i+b-1
        do jj = j, j+b-1
          do kk = k, k+b-1
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          end do
        end do
      end do
    end do
  end do
end do
```

MATRIX MULTIPLICATION : BLOCKING

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77\ MFlop/s$

(b = 16)

```
do i = 1, n, b
  do j = 1, n, b
    do k = 1, n, b
      do ii = i, i+b-1
        do jj = j, j+b-1
          do kk = k, k+b-1
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          end do
        end do
      end do
    end do
  end do
end do
```

$\sim 119\ MFlop/s$

MATRIX MULTIPLICATION : BLOCKING

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

$\sim 77\ MFlop/s$

(b = 16)

```
do i = 1, n, b
  do j = 1, n, b
    do k = 1, n, b
      do ii = i, i+b-1
        do jj = j, j+b-1
          do kk = k, k+b-1
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          end do
        end do
      end do
    end do
  end do
end do
```

$\sim 119\ MFlop/s$

Blocking ⇒ better cache use

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} += \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

MATRIX MULTIPLICATION : COMPILATION

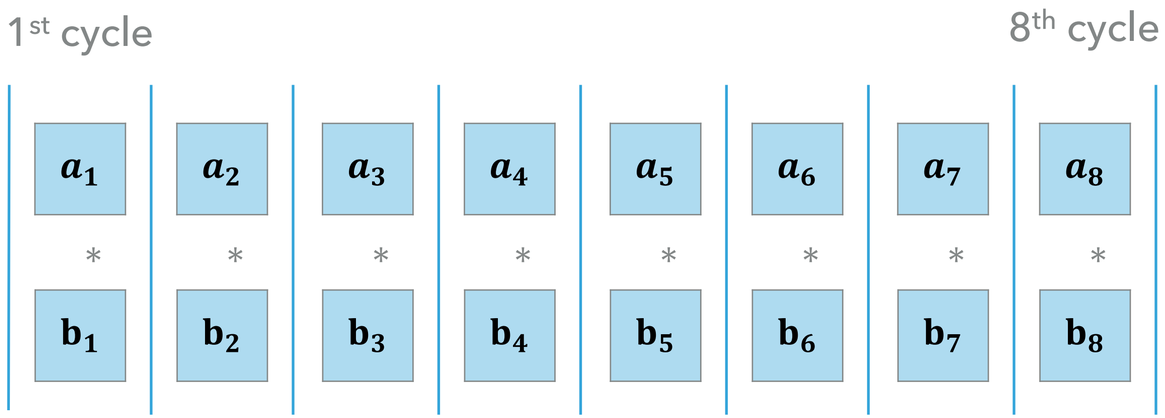
- Unrolling, pipelining … can be bone automatically by the compiler !

ifort -O0 prog.f90	ifort -O1 prog.f90	ifort -O2 prog.f90	ifort -O3 prog.f90
-O0	-O1	-O2	-O3
$\sim 77\ MFlop/s$	$\sim 510\ MFlop/s$	$\sim 2{,}65\ GFlop/s$	$\sim 4{,}37\ GFlop/s$
	• Few optimisations	• Unrolling • Pipelining • Instruction Level Parallelism	• Unrolling • Pipelining • Instruction Level Parallelism • Prefetching

- Same source code, speed-up goes up to *x56* !

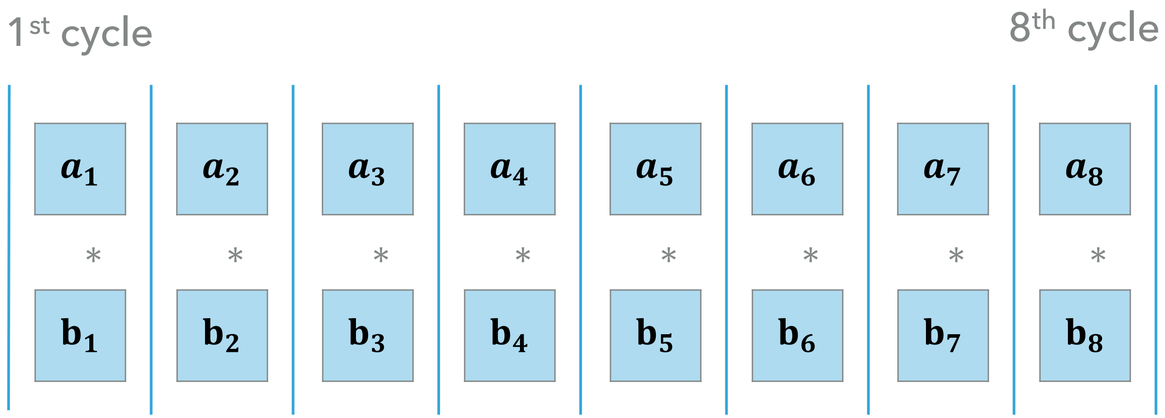
MATRIX MULTIPLICATION : VECTORISATION

- Scalar processing

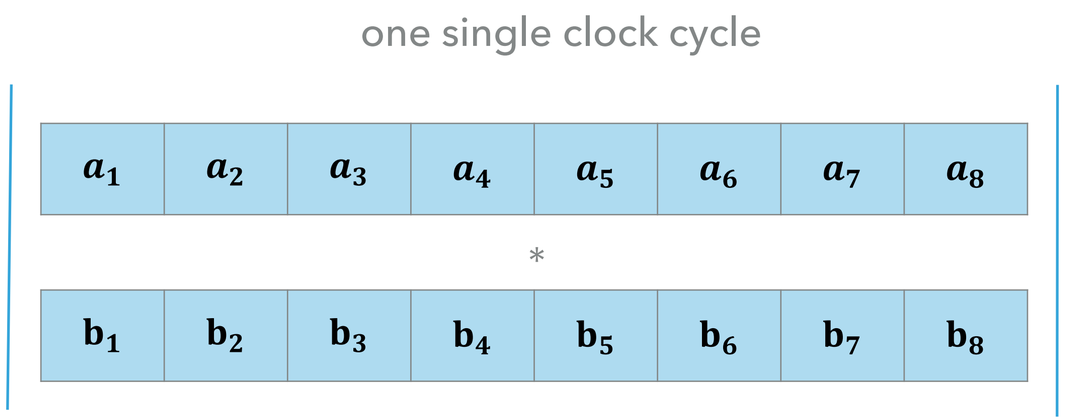


MATRIX MULTIPLICATION : VECTORISATION

- Scalar processing



- Vector processing



$\Longrightarrow$ x8 operations per cycle

MATRIX MULTIPLICATION : VECTORISATION

- Scalar processing

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

- Vector processing

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ b_{21} & \ddots & \vdots \\ b_{31} & \ddots & \vdots \\ b_{41} & \ddots & \vdots \\ b_{51} & \ddots & \vdots \\ b_{61} & \ddots & \vdots \\ b_{71} & \ddots & \vdots \\ b_{81} & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

@ : see CALMIP website for further details about vectorisation

MATRIX MULTIPLICATION : VECTORISATION

- Scalar processing

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

-O3 -no-vec	-O3 -xCOMMON-AVX512
$\sim 4{,}37\ GFlop/s$	$\sim 26{,}7\ GFlop/s$

- Vector processing

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ b_{21} & \ddots & \vdots \\ b_{31} & \ddots & \vdots \\ b_{41} & \ddots & \vdots \\ b_{51} & \ddots & \vdots \\ b_{61} & \ddots & \vdots \\ b_{71} & \ddots & \vdots \\ b_{81} & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

@ : see CALMIP website for further details about vectorisation

MATRIX MULTIPLICATION : VECTORISATION

- Scalar processing

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

- Vector processing

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ b_{21} & \ddots & \vdots \\ b_{31} & \ddots & \vdots \\ b_{41} & \ddots & \vdots \\ b_{51} & \ddots & \vdots \\ b_{61} & \ddots & \vdots \\ b_{71} & \ddots & \vdots \\ b_{81} & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

-O3 -no-vec	-O3 -xCOMMON-AVX512	-mkl=sequential
$\sim 4{,}37\ GFlop/s$	$\sim 26{,}7\ GFlop/s$	$\sim 65{,}1\ GFlop/s$ *

* $n = 4096$

@ : see CALMIP website for further details about vectorisation

calmip