Porting a 2D Acoustic Wave Modeling Application to HPF: issues and problems

Julien Zory

—

Centre de Recherche en Informatique

École des Mines de Paris

June 1998

`http://cri.ensmp.fr/~zory`

`julien.zory@cri.ensmp.fr`

PIPS : `http://cri.ensmp.fr/~pips`

Porting an application to HPF

➡ **Theory**

- declare a processor topology
- define and distribute some templates onto processors
- align data with templates
- compile and it works ...

➡ **Practice**

- get a good knowlegde of the data movements
- find out the HPF subset supported by your compiler
- debug and use profiling tools
- change the compiler and play again ...

On the menu

① Introduction

② ONDE24 overview

③ Encountered problems

- I/O related problems
- Porting problems
- Timing considerations

④ Performance results

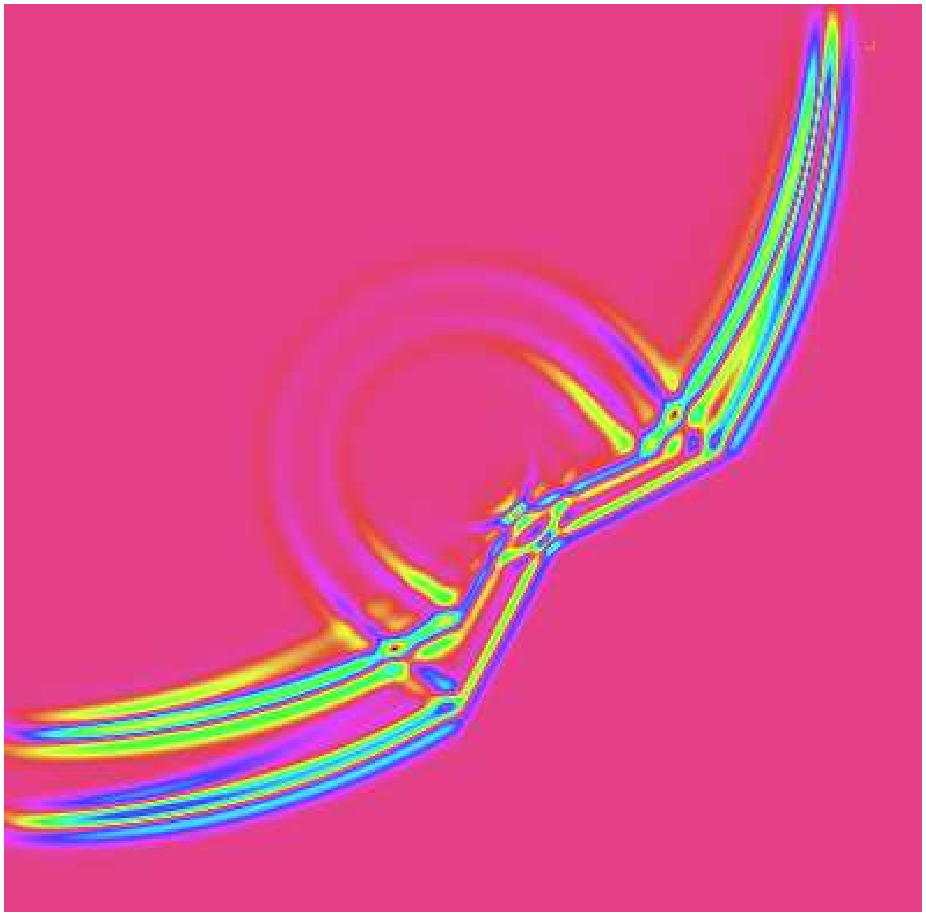
⑤ Conclusion

2-dimensionnal acoustic wave modeling application

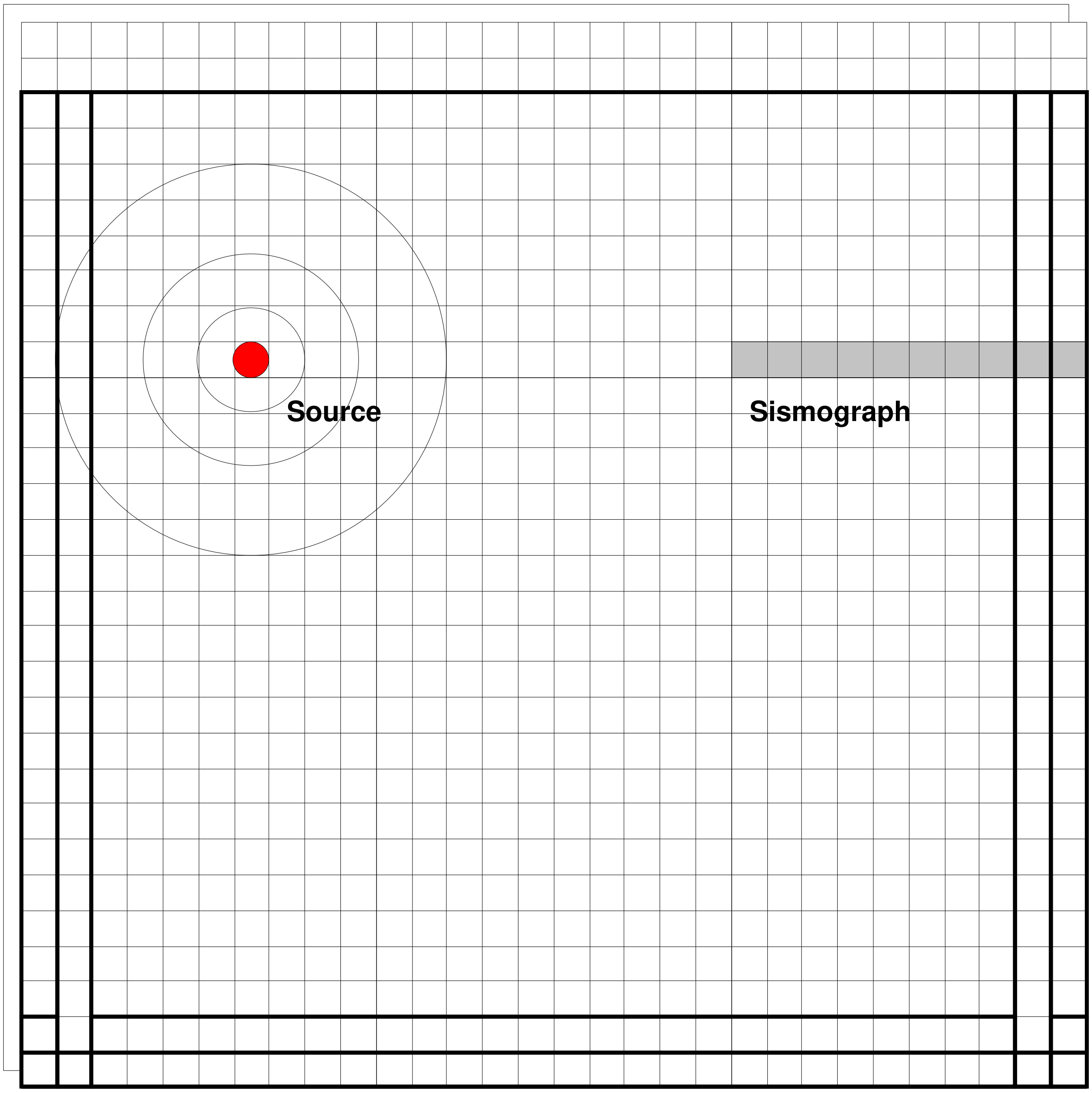
- French Oil Institute

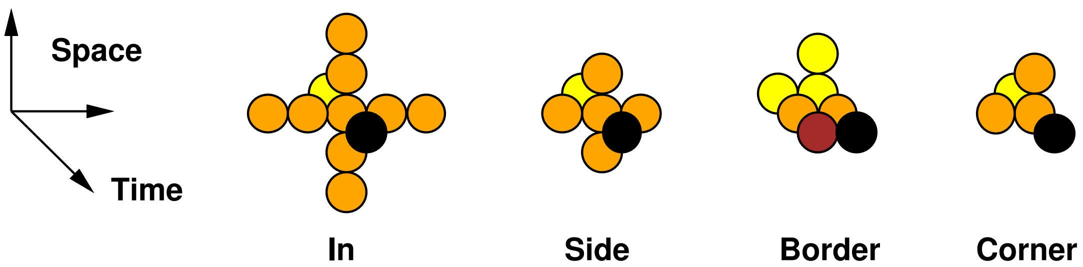
  (Institut Français du Pétrole)

- Written in Fortran 77



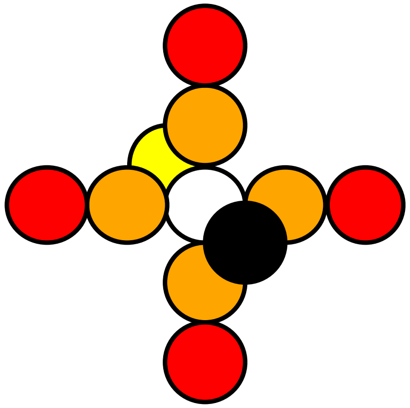
NP x NP x 2



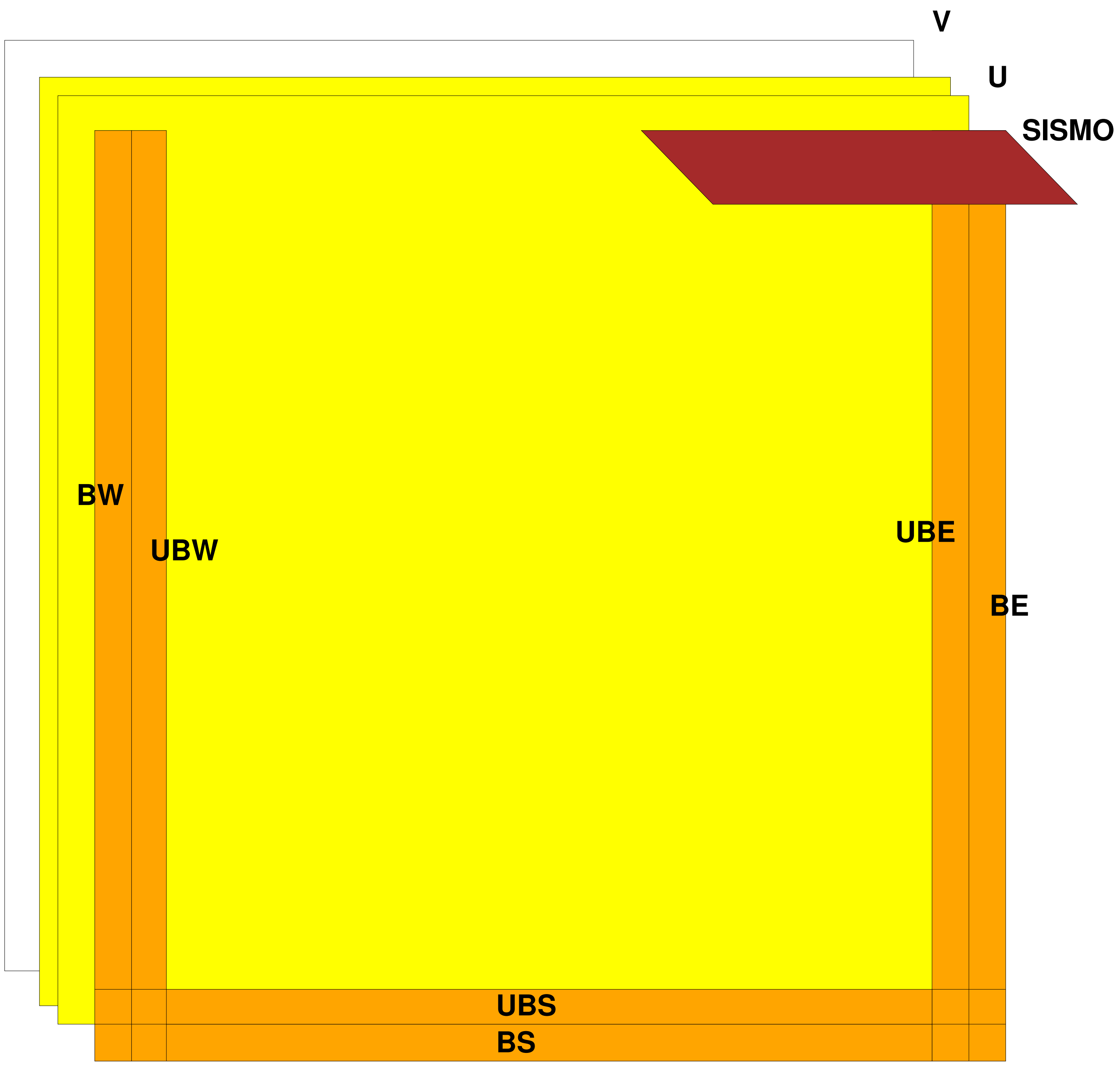


```
PROGRAM   ONDE24

    READ  V ( : , : )
    INITIALIZE
        data      U ( : , : , x )  SISMO ( : , : )
        corners    V  BE  BS  BW
        borders  BE ( : ) BS ( : ) BW ( : )  V ( : , 1 )  V ( : , n )  V ( n , : )
        sides  V ( : , 2 )  V ( : , n–1 )  V ( n–1 , : )
        inside   V ( 3:n–2 , 3:n–2 )

    DO  time
        SOURCE  U ( is , js )
        SAVE sides  UBE ( : )   UBS ( : )    UBW ( : )
        COMPUTE
            inside   U  ( 3:n–2 , 3:n–2 , x )
            sides    U ( : , 2 , x )   U ( : , n–1, x )   U( n–1 , : , x )
            borders  U ( : , 1 , x )  U ( : , n , x )   U ( n , : , x )
            corners  U
        RECORD  SISMO  ( : , time )
        flip–flop x
    ENDDO

    WRITE   SISMO ( : , : )
    STATS on   U ( : , : , x )    SISMO ( : , : )
END
```

```
[...]
 DO J = 3,NP-2
   DO I = 3, NP-2
     U(I,J,KP) =
$      2 * U(I,J,KM) - U(I,J,KP)
$      - V(I,J) * ( 60*U(I,J,KM)
$      -16*( U(I+1,J,KM) + U(I-1,J,KM)
$      + U(I,J-1,KM) + U(I,J+1,KM))
$      + U(I+2,J,KM) + U(I-2,J,KM)
$      + U(I,J-2,KM) + U(I,J+2,KM))
   ENDDO
 ENDDO
[...]
```



ONDE24 - array alignement



V: specific speed of the field

U: computed pressure

B/UB{W,S,E}: temporaries

SISMO: recorded data

Data Mapping

```
!HPF$ TEMPLATE DOMAIN(NP,NP)
!HPF$ PROCESSORS PE(2,2)
!HPF$ DISTRIBUTE DOMAIN(BLOCK,BLOCK) ONTO PE

!HPF$ ALIGN V(I,J) WITH DOMAIN(I,J)
!HPF$ ALIGN U(I,J,*) WITH DOMAIN(I,J)

!HPF$ ALIGN UBE(I) WITH DOMAIN(I,NP-1)
!HPF$ ALIGN BE(I,*) WITH DOMAIN(I,NP)
```

Independent and related directives

```
!HPF$ INDEPENDENT
!HPF$ INDEPENDENT, REDUCTION(V)
!HPF$ INDEPENDENT, NEW(V)
```

- IBM `SP2` - P2SC 595 processors (120MHz)
- 3 compilers

  `XLHPF 1.2` : native *IBM* compiler

  `ADAPTOR 6.0 (beta)` : public domain compiler from *GMD* (T. Brandes)

  `HPFC` : compiler prototype from *ENSMP-CRI* (F. Coelho)

- Execution

  IBM Parallel Environnement

  IBM native `MPI` implementation

I/O related problems

Initial I/O code

```
Do I = 1,NP
   Do J = 1,NP
      READ (3,*) V(I,J)
   Enddo
Enddo
```

- communications inside the loop nest

  XLHPF and ADAPTOR

- communications outside the loop nest

  HPFC

  thanks to array region analysis (PIPS)

What kind of solutions ?

Implied Do statement

```
READ (3,*) ((V(I,J), J=1, NP), I=1, NP)
```

Array language statement (Fortran 90)

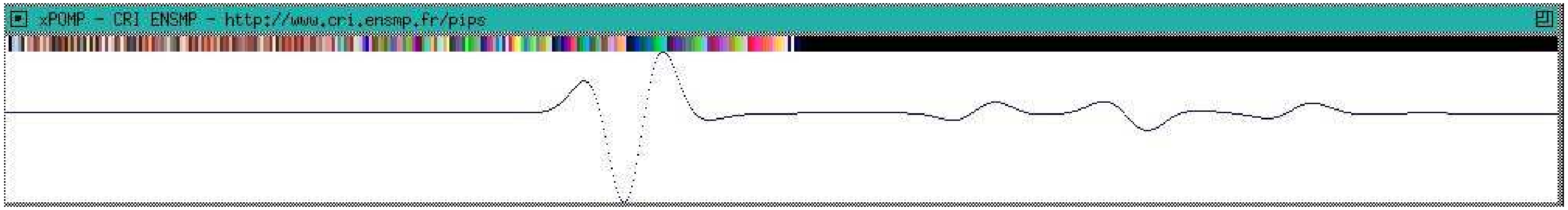
column major order - need to be transposed

```
READ (3,*) V(1:NP,1:NP)
V =  TRANSPOSE(V)
```

I/O subroutine call

- Input-Output operations using a library subroutine
- Only one processor must execute it
- For instance

```
open display(X_size,Y_size,window)
build display(SISMO(1,1),record,NT,window)
close display(window)
```



HPF solution

```
INTERFACE
   EXTRINSIC(F77_SERIAL) SUBROUTINE
&      open display(X, Y, window)
       INTEGER X, Y, window
       END SUBROUTINE open display
END INTERFACE
```

- Supported by ADAPTOR
- Not yet supported by XLHPF
- Not supported by HPFC - (use a specific directive !fcd$ io)

Reductions are not available in xlhpf V1.2

```
!HPF$ INDEPENDENT, REDUCTION(VMIN)
      Do J=1,NP
!HPF$ INDEPENDENT
         Do I=1,NP
            VMIN = MIN(VMIN,V(I,J))
         Enddo
      Enddo
```

➡ Fortran 90 solution

```
VMIN = minval(V)
```

Embedded dimensions

A problem with ADAPTOR

```
!HPF$ ALIGN UBE(I) WITH DOMAIN(I, NP-1)
!HPF$ ALIGN UBS(I) WITH DOMAIN( NP-1,I)
!HPF$ ALIGN UBW(I) WITH DOMAIN(I, 2)
```

➡ what kind of solutions ?

- declare UB(NP,NP)

  readability loss

  memory loss

- replicate data onto the second dimension

  increase the number of communications

  memory loss

No need to communicate

```
! south extern corner initialization
          C2       = 4/((1/V(NP,1))+(1/V(NP,2)) +
&                     (1/V(NP-1,1))+(1/V(NP-1,2)) )
          W        = C2*DELTAT
          BS(1,1)  = 2*W/(4*W+3*H)
          V(NP,1)  = C2
```

➡ We must express that C2 and W are private values

ON HOME directive

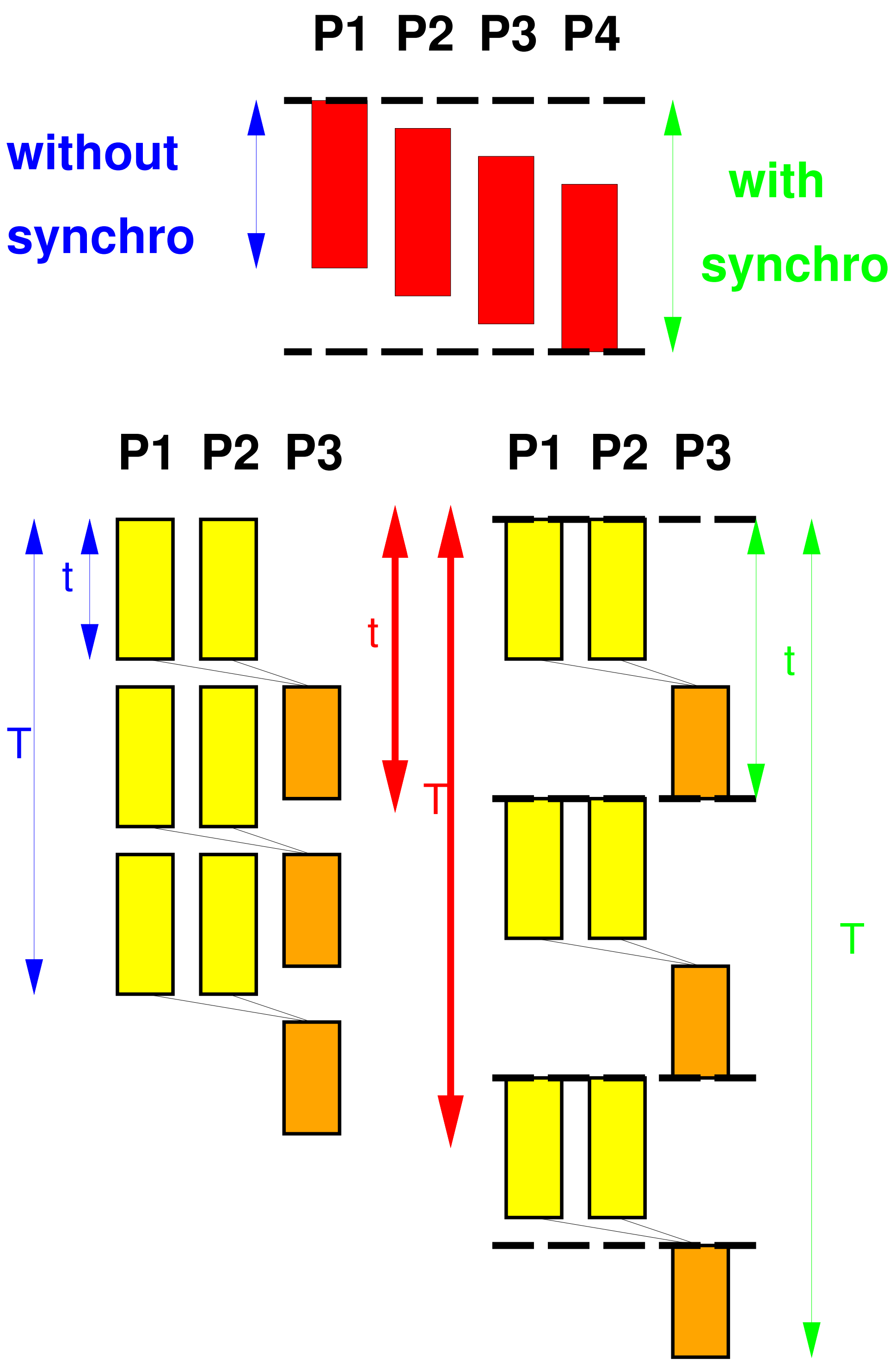
```
!HPF$ ON HOME (V(NP,1)), NEW(C2,W) BEGIN
      C2       = 4/((1/V(NP,1))+(1/V(NP,2)) +
&                 (1/V(NP-1,1))+(1/V(NP-1,2)) )
      W        = C2*DELTAT
      BS(1,1)  = 2*W/(4*W+3*H)
      V(NP,1)  = C2
!HPF$ END ON
```

ONLY an **approved extension** in HPF2!!!

- Supported by ADAPTOR
- Not yet supported by XLHPF
- Not supported by HPFC

(a specific directive is used !fcd$ local, new())

What do compilers really measure ?



Timing considerations

What do compilers really measure ?

- 2 kinds of measurements

  without synchronisation - *under* estimation (false ?)

  with synchronisation - *over* estimation (useless ?)

- 3 compilers

  ADAPTOR: intrinsic `system_clock` - no synchronization

  XLHPF: intrinsic `system_clock` - no synchronization

  HPFC: 2 directives (timeon - timeoff) - with synchronization

➡ To be specified in the HPF specification ?

How do compilers handle optimized loop nests

- A wide range of optimizations

  loop unrolling

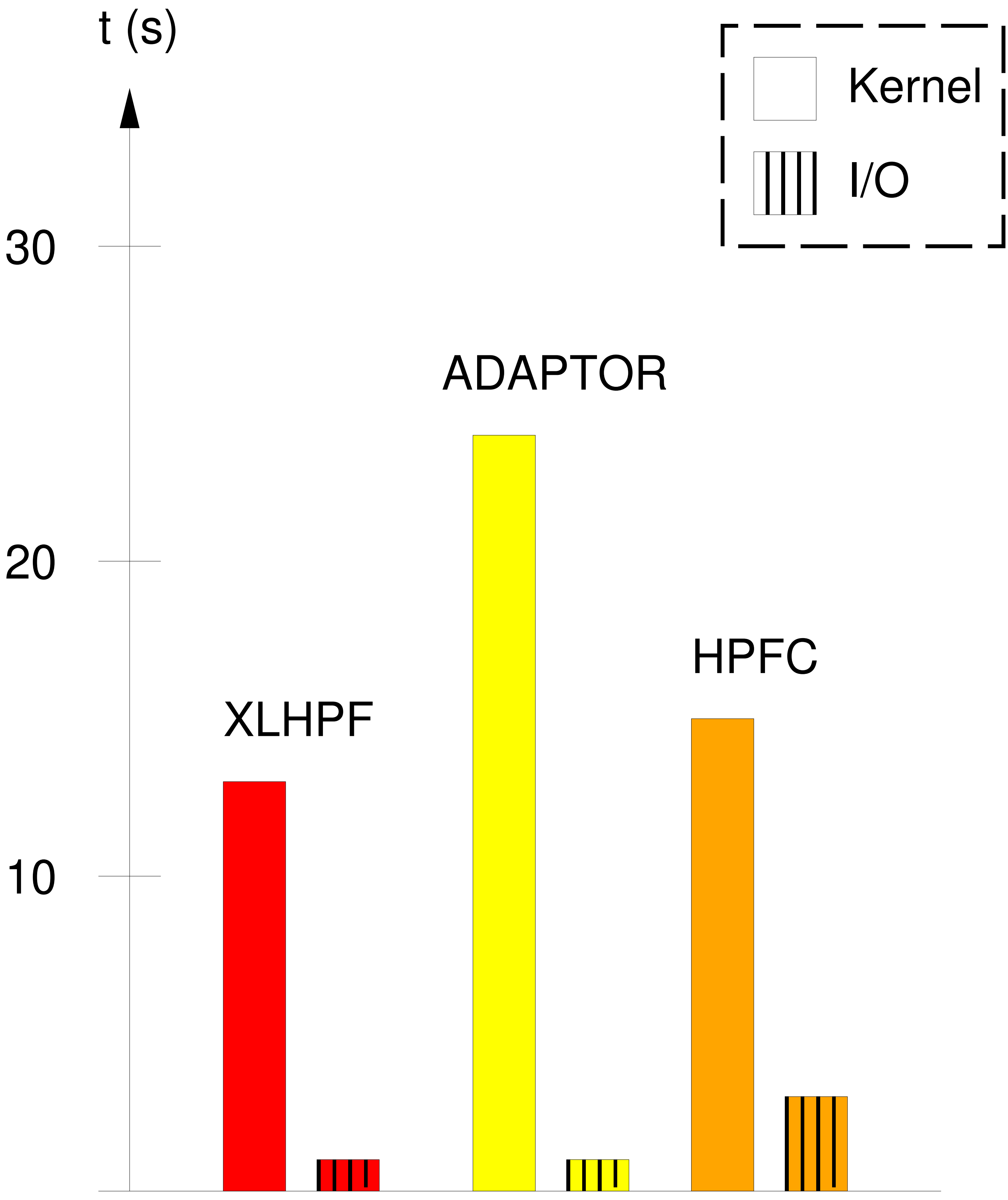
  software pipelining

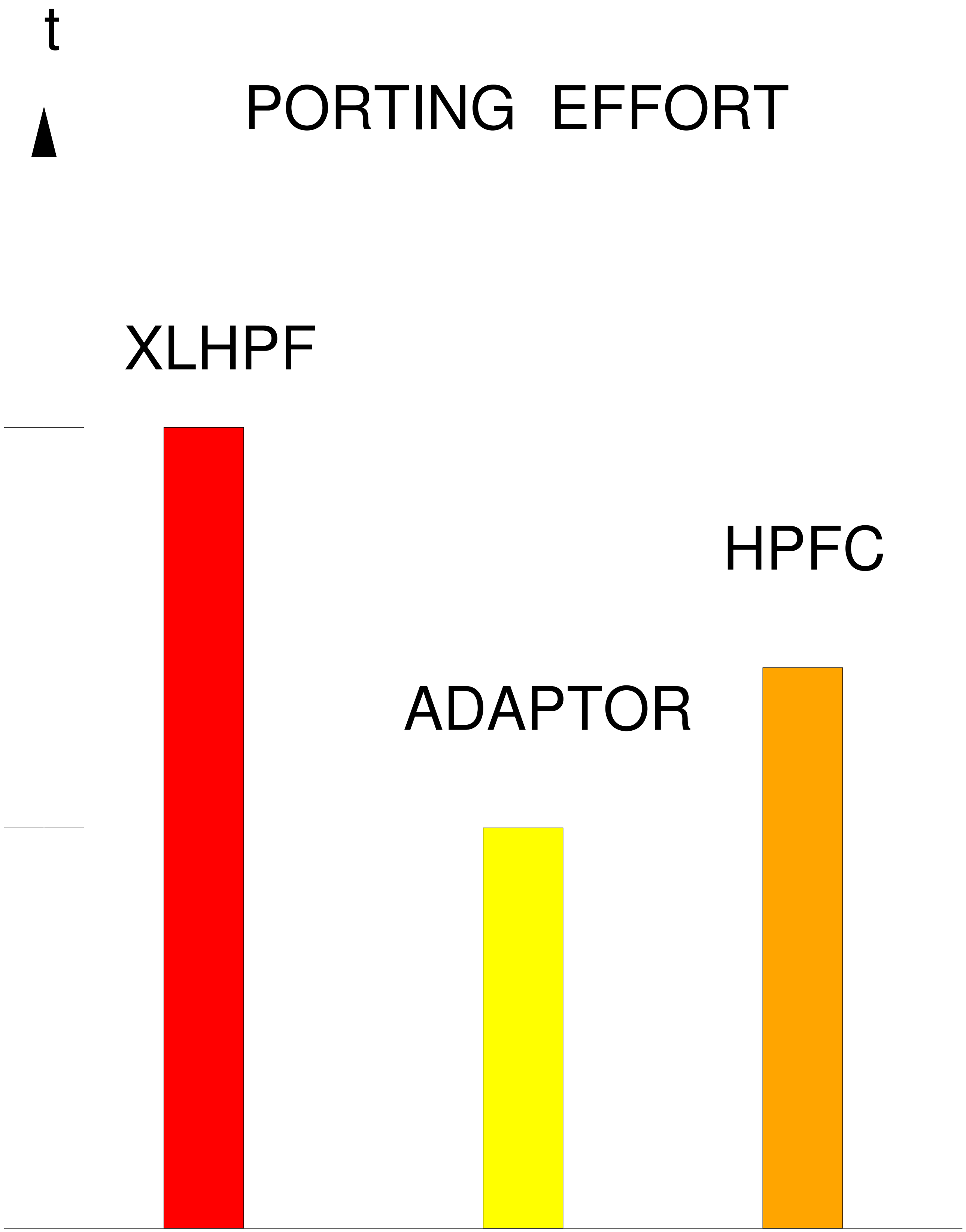
  atomization (a lot of new variables)

- A very long but efficient *sequential* code

BUT THE LOOP IS **SERIALIZED**

➡ Very poor performance


t (s)
30
20
10
Kernel
I/O
XLHPF
ADAPTOR
HPFC


t
PORTING EFFORT
XLHPF
ADAPTOR
HPFC

Conclusion

- A really *trivial* application

- A *successful* porting to HPF

  HPF is easier than message passing programming

  Quite good performance

- BUT porting an application to HPF is *not so easy*

  F77 initial code has to be transformed

  Debugging and profiling techniques are necessary

  Porting efforts are compiler dependent

These experiments showed that ...

① Some HPF2 approved extensions are really **NEEDED in pratice**

② 2 possible compiler strategies

work on optimization features and get higher *efficiency*

support most of the HPF langage specification