

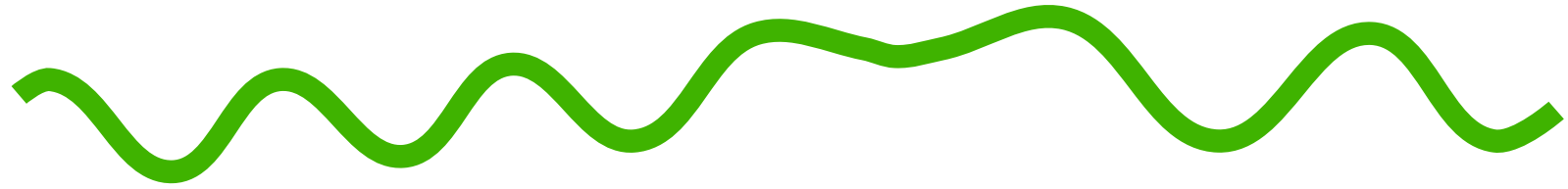
# ***Introduction à HPF – High Performance Fortran***

***Outils pour le calcul scientifique à haute performance  
École doctorale sciences pour l'ingénieur  
mai 2001***

Pierre BOULET

Pierre.Boulet@lifl.fr

Laboratoire d'informatique fondamentale de Lille  
Université des sciences et technologies de Lille



- ~ Ce cours est diffusé sous la licence GNU Free Documentation License,  
<http://www.gnu.org/copyleft/fdl.html>
- ~ La dernière version de ce cours est accessible à partir de  
<http://www.lifl.fr/west/courses/cshp/>
- ~ \$Id: hpf.tex,v 1.5 2001/11/02 13:08:26 marquet Exp \$

# *Table des matières*



- ~ Tableaux en Fortran 95
- ~ Distribution des données
- ~ Expression du parallélisme
- ~ Exemples à faire en TP
- ~ Compléments

# Remerciements



Cette présentation de HPF est essentiellement basée sur

~ *The Liverpool HPF Courses*

University of Liverpool

<http://www.liv.ac.uk/HPC/HPFpage.html>

~ *Designing & Building Parallel Programs*

Chapitre *HPF*

Ian FORSTER

Argonne National Laboratory

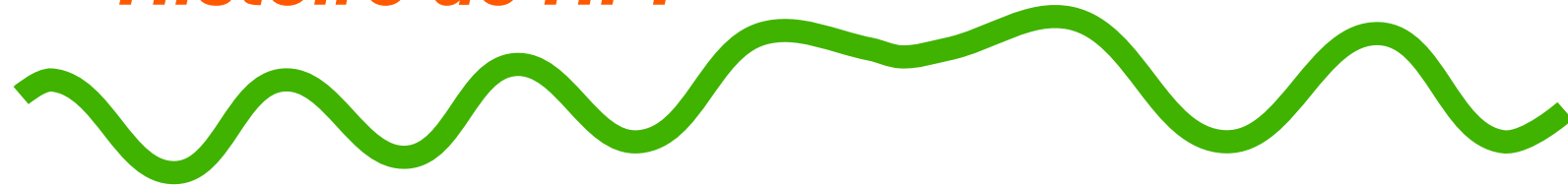
<http://www.mcs.anl.gov/dbpp>

~ *EPCC HPF Training*

Edinburgh Parallel Computing Centre

<http://www.epcc.ed.ac.uk/epcc-tec/hpf/>

# Histoire de HPF



- ✓ au début était Fortran 77
  - ✓ langage archaïque mais extrêmement performant et très répandu
- ✓ puis vint Fortran 90
  - ✓ langage moderne (type utilisateur, modules, allocation dynamique de la mémoire, syntaxe libre, ...)
  - ✓ apporte la notation de tableau
  - ✓ évolue légèrement avec Fortran 95
- ✓ naissance d'High Performance Fortran
  - ✓ standardisation par le High Performance Forum (HPFF, <http://www.crpc.rice.edu/HPFF/>)
  - ✓ versions successives :
    - ✓ v1.0 en mai 1993 (après 2 ans de développement)
    - ✓ v1.1 en novembre 1994
    - ✓ v2.0 en janvier 1997
  - ✓ le compilateur IBM `xlhpfc` : sous-ensemble de HPF 1.1 avec quelques extensions

# Objectifs et concepts

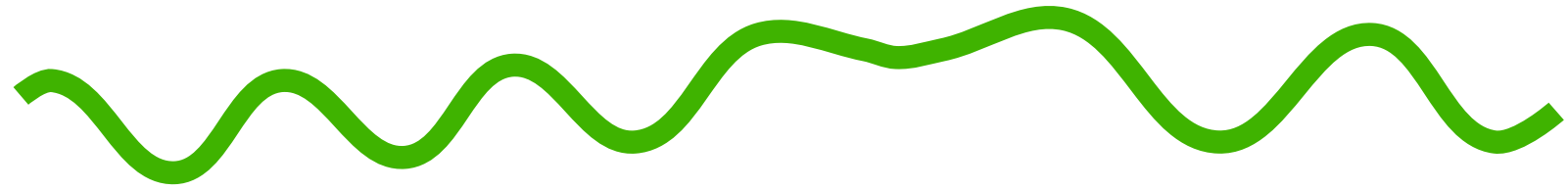


## ~ objectifs

- ~ programmation à parallélisme de données
- ~ performances excellentes sur machines MIMD et SIMD
- ~ possibilités d'optimisation de code sur des architectures variées

## ~ concepts essentiels

- ~ un seul programme : programmation SPMD
- ~ chaque processeur travaille sur une partie des données
- ~ des directives HPF indiquent quel processeur possède quelles données
- ~ des instructions Fortran 95 et HPF spécifient le parallélisme



## ***Tableaux en Fortran 95***

# Affectation



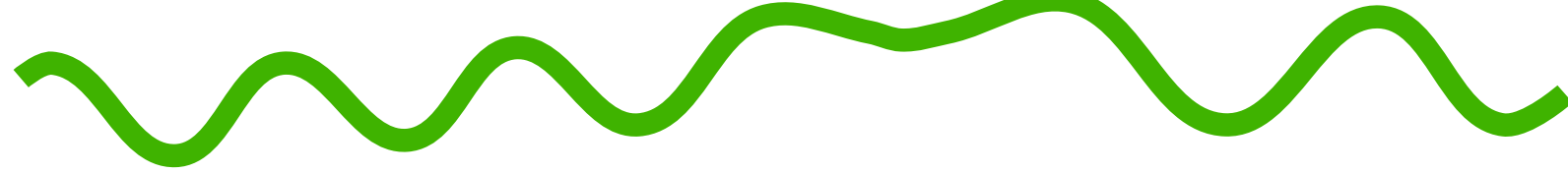
- ~ on manipule les tableaux sans indices
  - ~ les tableaux doivent être conformes
    - ~ même dimension
    - ~ même taille dans chaque dimension
    - ~ pas forcément mêmes bornes
  - ~ opérations scalaires élément par élément
  - ~ les scalaires sont répliqués
- ~ exemples
  - ~ `a = a_new`
  - ~ `diff = ABS(a - a_new)`

## Sections de tableaux



- ✓ accès à une partie régulière d'un tableau
  - ✓ peuvent-être substituées à un tableau n'importe où
- ✓ notation : `tableau(dbut:fin:pas, ...)`
  - ✓ on peut omettre le pas (1 par défaut)
  - ✓ : dénote la dimension complète
- ✓ exemple : un pas des différences finies sans rebouclage
  - ✓  $a(2:n-1) = (a(1:n-2) + 2*a(2:n-1) + a(3:n)) / 4$

## Accès conditionnel : *WHERE*



~ permet les affectations sur des sections non régulières de tableaux

~ exemples :

```
WHERE ( I /= 0 ) A=B/I
```

```
WHERE ( A > 0 )
```

```
    B = LOG(A)
```

```
ELSE WHERE
```

```
    B = 0.0
```

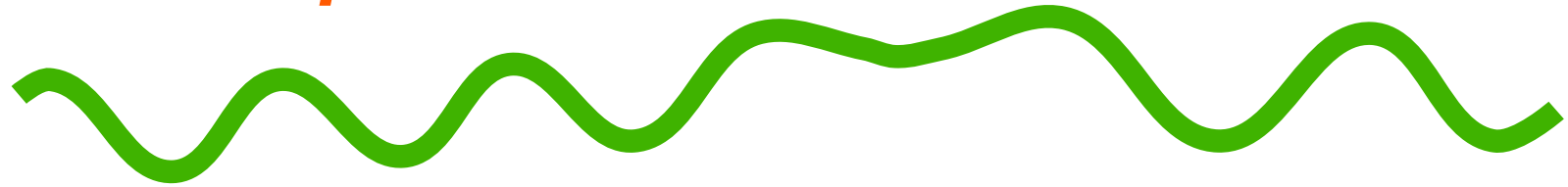
```
END WHERE
```

# Fonctions intrinsèques



- ~ toutes les fonctions intrinsèques travaillant sur les scalaires
- ~ des fonctions structurelles : `LBOUND(A, DIM)`, `UBOUND(A, DIM)`, `SIZE(A, DIM)`, `SHAPE(A)`, `ALLOCATED(A)`
- ~ des fonctions de calcul sur les tableaux : `MAXVAL(A)`, `MINVAL(A)`, `SUM(A)`, `PRODUCT(A)`, `MAXLOC(A)`, `MINLOC(A)`, `MATMUL(A, B)`, `DOT_PRODUCT(A, B)`, `TRANSPOSE(A)`, `CSHIFT(A, SHIFT, DIM)`
- ~ exemple
  - ~ le calcul d'erreur  
$$\text{error} = \text{MAXVAL}(\text{ABS}(a - a_{\text{new}}))$$
  - ~ un pas des différences finies avec rebouclage  
$$a_{\text{new}} = (\text{CSHIFT}(a, -1) + 2*a + \text{CSHIFT}(a, 1)) / 4$$

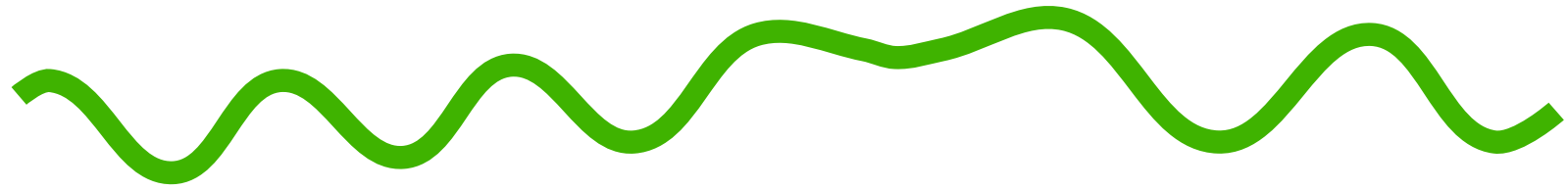
## Exemple : les différences finies



...

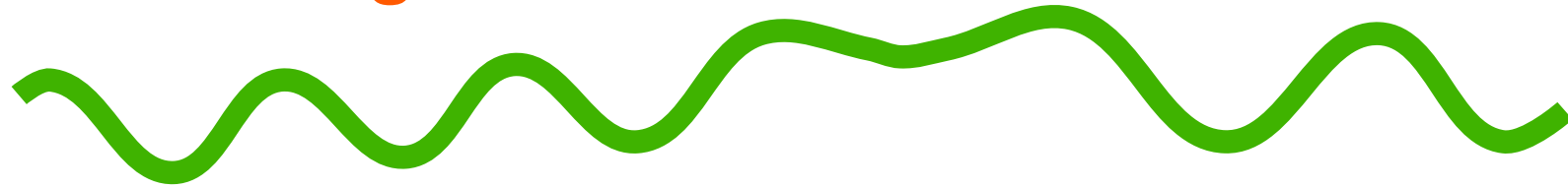
```
DO WHILE (error > epsilon)
  a_new = (CSHIFT(a,-1) + 2*a + CSHIFT(a,1))/4
  error = MAXVAL(ABS(a - a_new))
  a = a_new
END DO

WRITE *, 'voici le tableau resultat :'
WRITE *, a
```



## ***Distribution des données***

# Vision globale



- ✓ placement des données spécifié par des directives
  - ✓ **commentaires** commençant par !HPF\$
  - ✓ indications au compilateur, il en fait ce qu'il veut !
  - ✓ traités comme des commentaires par un compilateur Fortran 95
- ✓ placement sur une grille virtuelle de processeurs
  - ✓ implicite
  - ✓ ou explicite (directive PROCESSORS)
- ✓ alignement des tableaux
  - ✓ pour la localité des données
  - ✓ **directive ALIGN**
  - ✓ peut se faire sur des tableaux virtuels (directive TEMPLATE)
- ✓ distribution des tableaux
  - ✓ **directive DISTRIBUTE**
- ✓ *remarque : on ne présentera qu'une des syntaxes possibles*

## Directive *ALIGN*



~ alignement relatif de deux tableaux

~ !HPF\$ ALIGN <tableau> WITH <cible>

~ exemples :

```
!HPF$ ALIGN A1(:) WITH B1(:)
```

```
!HPF$ ALIGN A2(I) WITH B2(I)
```

```
!HPF$ ALIGN A3(I) WITH B3(2*I+1)
```

```
!HPF$ ALIGN A4(I,J) WITH B4(J,I)
```

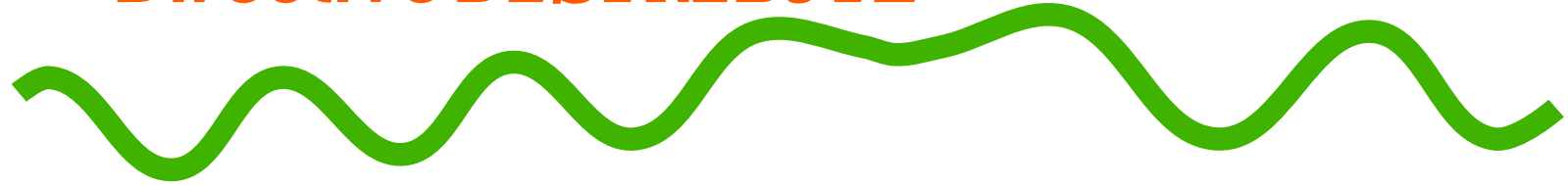
~ « écrasement » de dimension

```
!HPF$ ALIGN A5(I,*) WITH B5(I)
```

~ réplique de dimension

```
!HPF$ ALIGN A6(I) WITH B6(I,*)
```

# Directive *DISTRIBUTE*



- répartition des tableaux sur les processeurs virtuels
  - lien entre processeurs virtuels et processeurs physiques laissé à l'interprétation du compilateur
- distributions possibles pour chaque dimension :
  - \* : pas de distribution (placement en mémoire)
  - BLOCK( $n$ ) : distribution par blocs (par défaut  $n = N/P$ )
  - CYCLIC( $n$ ) : distribution cyclique (par défaut  $n = 1$ )

## exemples :

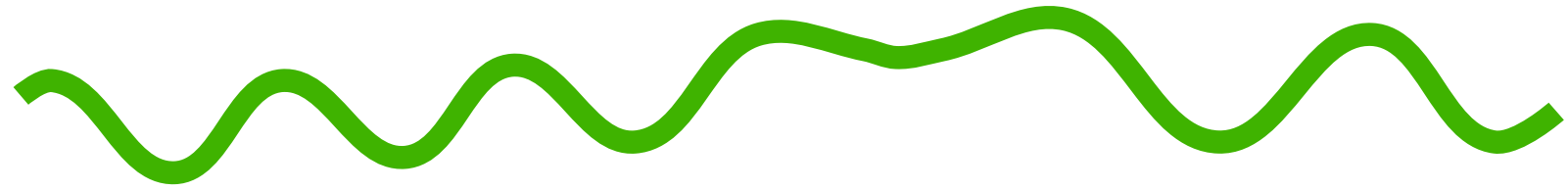
```
!HPF$ DISTRIBUTE A1(BLOCK)           ! A1(8), 4 procs
!HPF$ DISTRIBUTE A2(BLOCK,*)        ! A2(8,8), 4 procs
!HPF$ DISTRIBUTE A3(*,CYCLIC)       ! A3(8,8), 4 procs
!HPF$ DISTRIBUTE A4(BLOCK,*,BLOCK)  ! A4(8,8,8), 2x2 procs
!HPF$ DISTRIBUTE A5(CYCLIC,BLOCK)   ! A5(8,8), 4x4 procs
!HPF$ DISTRIBUTE A6(CYCLIC(2),BLOCK(5))
                                     ! A6(8,8), 4x2 procs
```

## Choix de la distribution



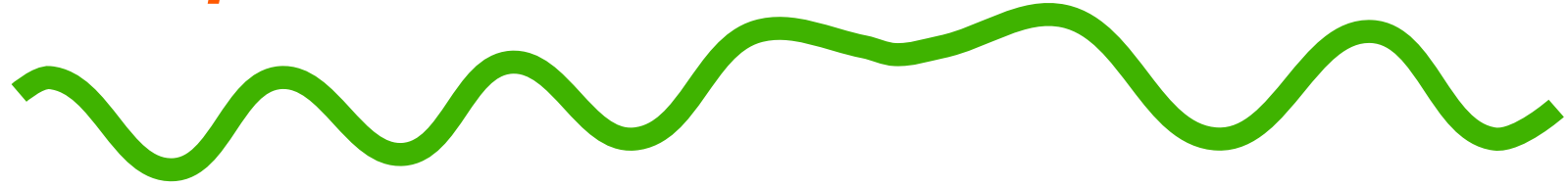
- ~ dépend du motif de calcul, objectif = minimiser les communications
  - ~ BLOCK : utile pour des calculs avec les voisins
  - ~ CYCLIC : bon pour l'équilibrage de charge
  - ~ \* : utile si le calcul se fait sur une ligne ou une colonne complète
- ~ exemple des différences finies :

```
!HPF$ ALIGN a_new(:) WITH a(:)
!HPF$ DISTRIBUTE a(BLOCK)
```



## *Expression du parallélisme*

# *Expressions de tableaux*



- ✓ pas de placement implicite en Fortran 95
  - ✓ optimisable par le compilateur
  - ✓ distribution de données possible
- ✓ intrinsèquement parallèles
  - ✓ elles expriment uniquement les dépendances
  - ✓ pas d'ordre de parcours explicite
  - ✓ les utiliser le plus possible
  - ✓ y compris les fonctions intrinsèques

## Instruction FORALL

~ permet d'exprimer des calculs irréguliers

~ deux formes :

```
FORALL (<triplets>[, <masque>]) <affectation>
```

OU

```
FORALL (<triplets>[, <masque>])
```

```
  <affectation>
```

...

```
END FORALL
```

~ triplet :

```
<indice> = <borne inf> : <borne sup> : <pas>
```

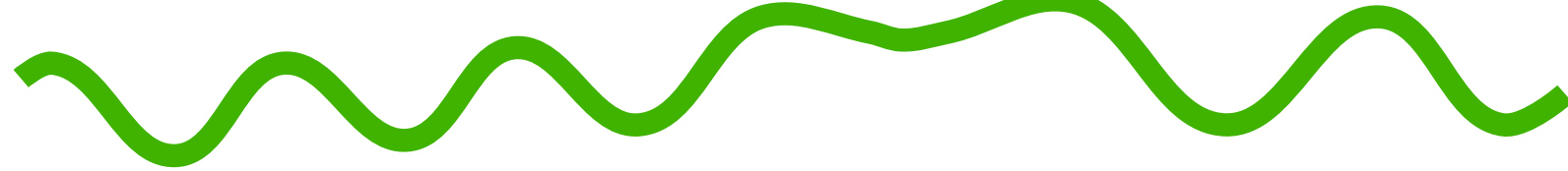
~ exemples simples :

```
FORALL (i=1:m, j=1:n)      X(i,j)=i+j
```

```
FORALL (i=1:n)            Y(i,i)=0.0
```

```
FORALL (i=1:m, j=1:n, i<j) Z(i,j)=0.0
```

## FORALL : utilisation avancée



~ ordre d'exécution (synchronisation entre chaque étape) :

1. évalue les expressions d'indices
2. évalue le masque pour tous les indices
3. pour tous les éléments où le masque vaut `.TRUE.`, évalue la partie droite de l'affectation
4. affecte les résultats

~ validité si affectation une fois au plus de chaque élément

~ traitement séquentiel des affectations

~ exemples :

```
FORALL (i=1:m, j=1:n) A(V(i), j) = i + V(j)
```

```
FORALL (j=1:n)
```

```
  FORALL (i=1:j-1) A(i, j) = B(i)
```

```
  A(j, j) = 1.0
```

```
END FORALL
```

## *Directive INDEPENDENT*



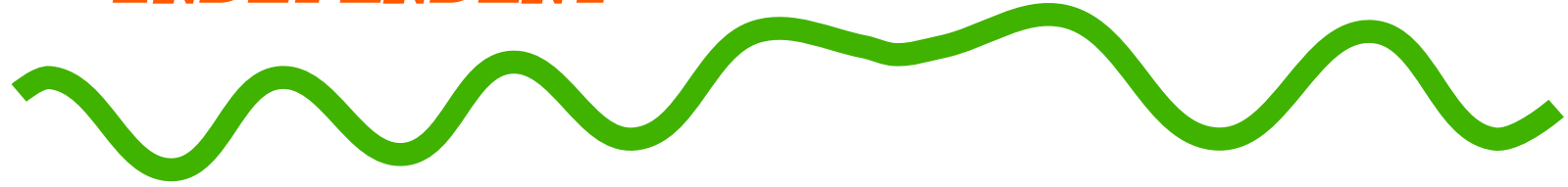
- indique au compilateur que toutes les itérations de la boucle DO ou FORALL suivante sont indépendantes
- ordre d'exécution quelconque possible (en particulier parallèle)
- vérification à la charge du programmeur
- aucune itération ne lit une valeur produite par une autre

exemples :

```
!HPF$ INDEPENDENT
FORALL (i=1:n) A(i)=i**2
!HPF$ INDEPENDENT
DO i=1,m
  DO j=1,n-1
    A(i,j)=A(i,j+1)
  END DO
END DO
```

# Variables locales à une boucle

## INDEPENDENT

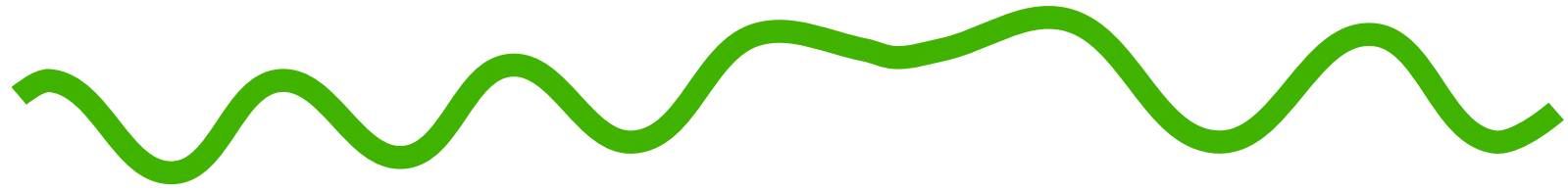


~ possibilité de spécifier qu'une boucle est indépendante à condition qu'un stockage distinct soit utilisé pour un certain nombre de variables

~ !HPF\$ INDEPENDENT, NEW(<variables>)

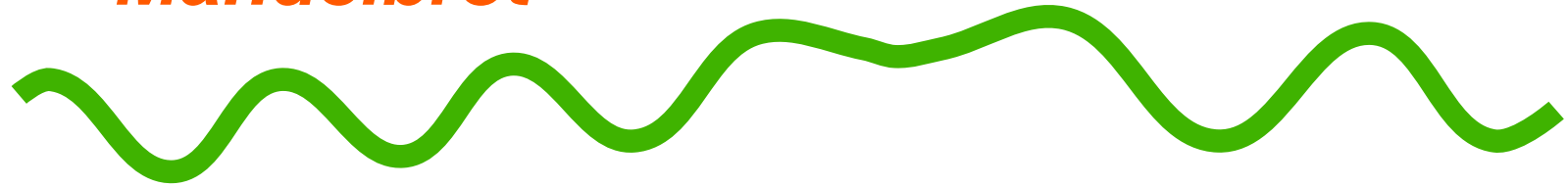
~ exemple :

```
!HPF$ INDEPENDENT, NEW(tmp)
DO i=1,n
  tmp = SUM(B(i,:))
  A = tmp*tmp
END DO
```



## *Exemples à faire en TP*

# Mandelbrot



✓ dessiner l'ensemble de Mandelbrot

✓ pour les points  $c \in [-1, 1]^2$  calculer la suite itérée  $z_{n+1} = z_n^2 + c$

✓ la valeur retenue est le premier indice pour lequel  $|z_n| > 2$  et si  $n > n_{\max}$ , retourner 0

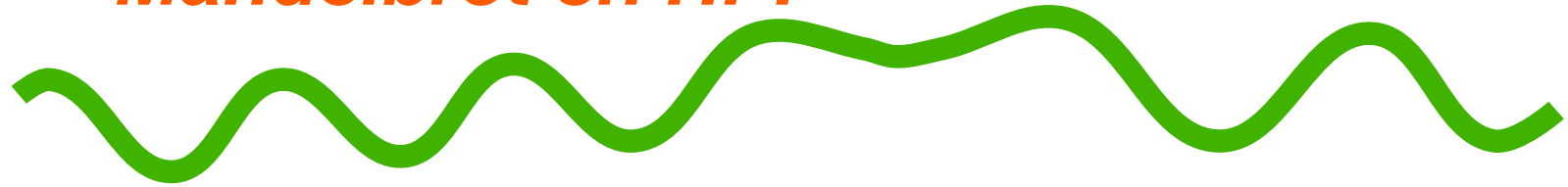
✓ modélisation en séparant partie réelle et imaginaire

✓ image codée par le tableur couleur

✓ on calcule pour chaque itération n

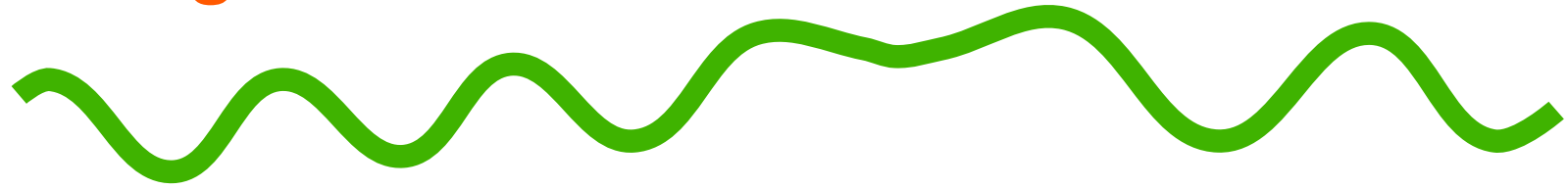
```
DO i = 1,taille
  DO j = 1,taille
    IF (zrs(i,j)+zis(i,j) <= 4) THEN
      zrs(i,j) = zr(i,j)*zr(i,j)
      zis(i,j) = zi(i,j)*zi(i,j)
      zi(i,j) = 2.0*zr(i,j)*zi(i,j) + ci(i,j)
      zr(i,j) = zrs(i,j) - zis(i,j) + cr(i,j)
      couleur(i,j) = n
    END IF
  END DO
END DO
```

## Mandelbrot en HPF

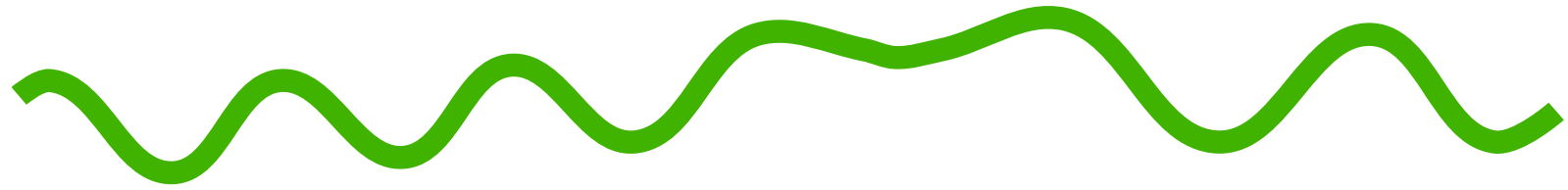


- ↻ récupérer le squelette du code (`~boulet/cshp/TP/mandel.f`)
- ↻ initialiser les tableaux avec des `FORALL`
- ↻ coder la boucle sur  $n$
- ↻ utiliser `WHERE` et des expressions de tableaux pour coder le corps de boucle (traduction du code donné précédemment)
- ↻ distribuer les tableaux avec des directives HPF
- ↻ afficher l'image générée : - )

# Algèbre linéaire

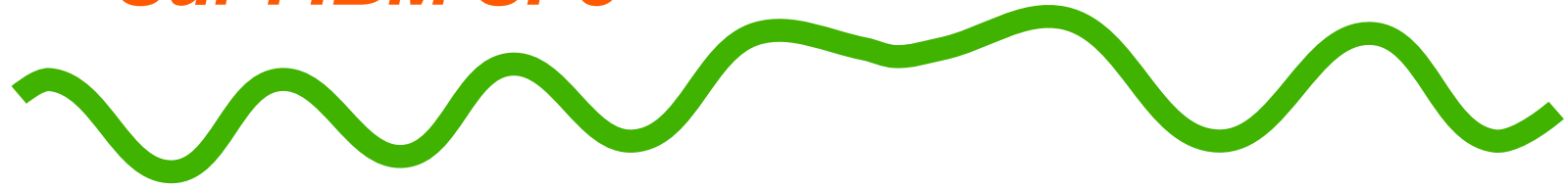


- ✓ produit scalaire
- ✓ produit matrice vecteur
- ✓ produit de matrice
- ✓ remontée triangulaire
  - ✓  $T$  matrice triangulaire supérieure
  - ✓ résoudre  $Tx = b$
- ✓ algorithme de Gauss



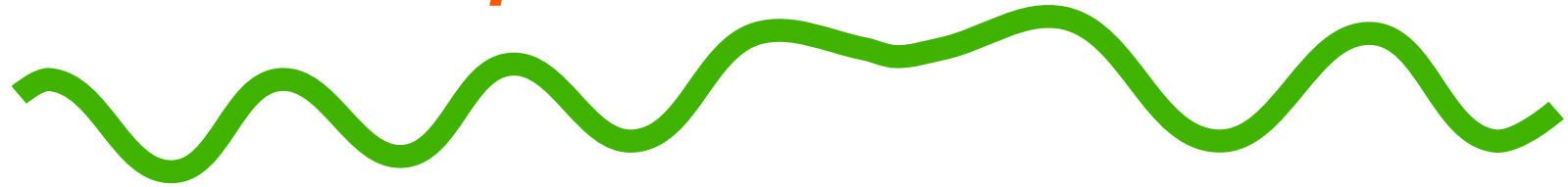
## ***Compléments***

## Sur l'IBM SP3



- ✓ compilateur x1hpf v1.4
  - ✓ plusieurs variantes : x1hpf, x1hpf90, x1hpf95
  - ✓ de nombreuses options de compilation (voir xx1hpf)
- ✓ environnement d'exécution
  - ✓ comme pour MPI
  - ✓ lancement du programme avec `poe`
- ✓ limitations
  - ✓ sous-ensemble de HPF 1.1
  - ✓ `CYCLIC(n)` traité comme `CYCLIC(1)`
  - ✓ pas de pointeurs
  - ✓ pas de `WHERE` imbriqués

## *Pour aller plus loin*



- ✓ appel de procédures
  - ✓ attribut `PURE` pour indiquer que la procédure ne fait pas d'effets de bords
  - ✓ procédures venant d'un autre module (`INTERFACE`)
  - ✓ procédures écrites dans un autre langage
  - ✓ placement des arguments ?
- ✓ placement dynamique
  - ✓ `REALIGN`, `REDISTRIBUTE`
  - ✓ très coûteux en temps de communication
  - ✓ non supporté par `xlhpF`