

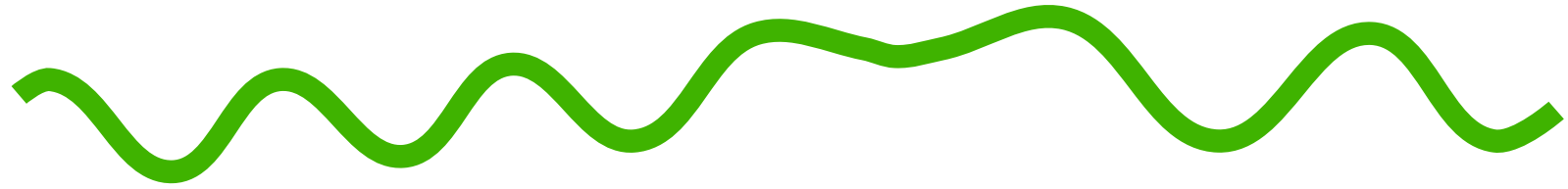
Performances et optimisations

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

Philippe MARQUET

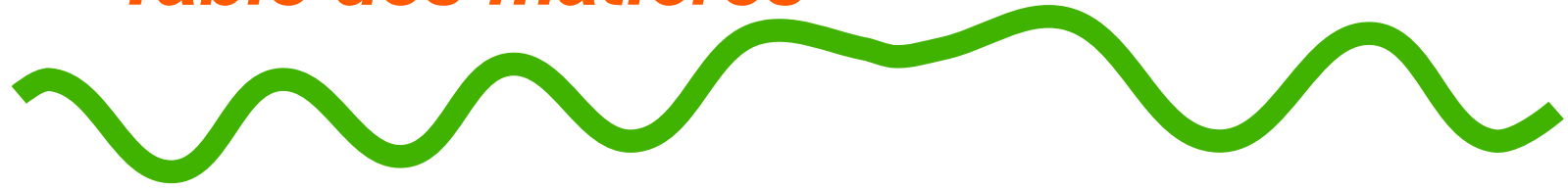
phm@lifl.fr

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: perf.tex,v 1.6 2001/11/02 13:08:26 marquet Exp $`

Table des matières



- ~ Motivations
- ~ Analyse de performance
- ~ Optimisations
- ~ Compilateurs IBM Fortran et C

Remerciements



Cette présentation est essentiellement basée sur

✓ *Performance analysis tools*

Maui High Performance Computing Center, 2000

http://www.mhpcc.edu/training/workshop2/performance_tools/

✓ *Optimisations Fortran et C sur IBM Risc System 6000*

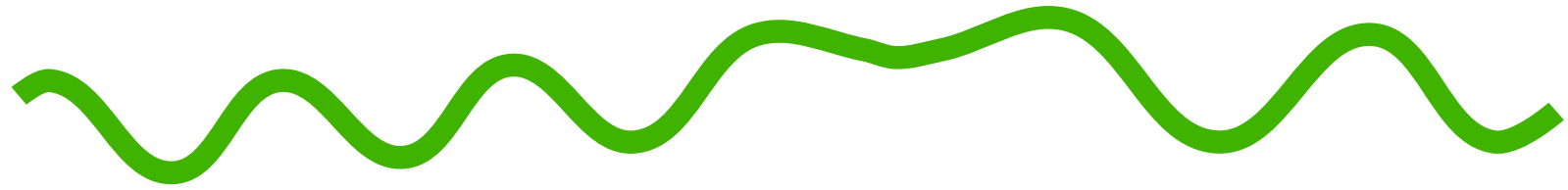
Olivier HESS

IBM France, 2000

✓ *Environnement Fortran, éléments d'optimisation*

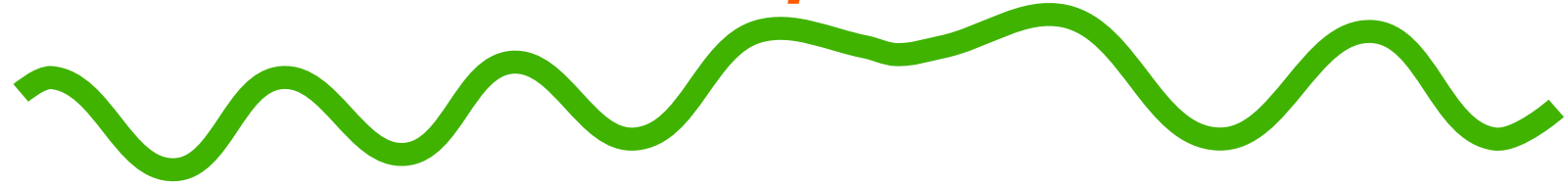
Yvon TINEL

CRI, Université de Lille 1, 2001

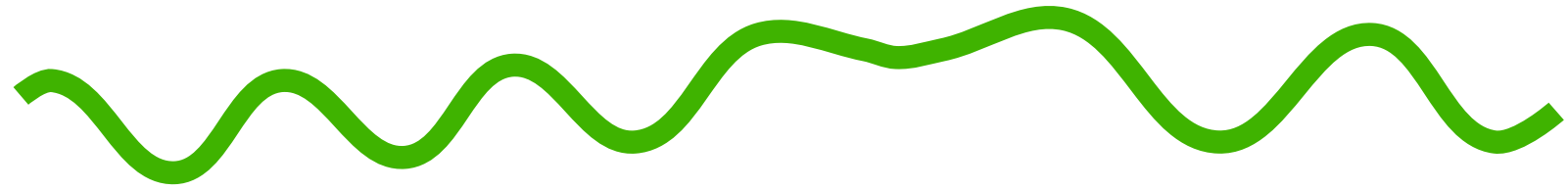


Motivations

Performances et optimisations

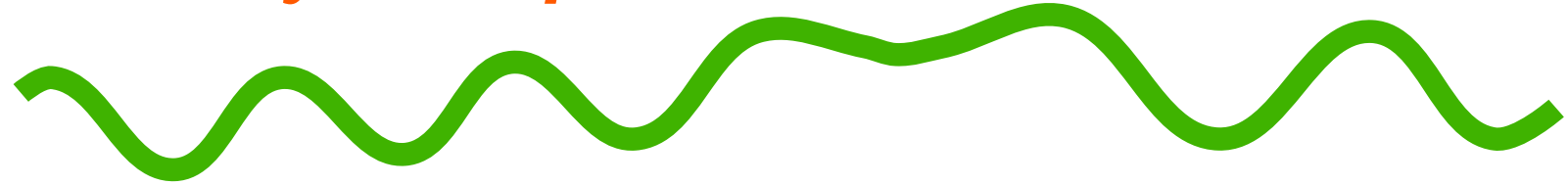


- ✓ Performance = raison d'être des programmes parallèles
- ✓ Complexité et multiplicité des facteurs influents les performances
 - ✓ application
 - ✓ matériel
 - ✓ logiciel
- ✓ Outils d'analyse de l'exécution, que fait mon programme ?
 - ✓ commandes de chronométrage
 - ✓ bibliothèques de prise de temps
 - ✓ profilers
 - ✓ générateurs de traces d'exécution
 - ✓ analyseur graphiques de traces
 - ✓ outils temps-réels et post-mortem
- ✓ Optimisations « bas niveau » du programme
 - ✓ réglage du compilateur
 - ✓ optimisations de l'allocation mémoire
 - ✓ optimisation de boucles



Analyse de performance

Analyse de performances



- ✓ Pourquoi une analyse des performances
 - ✓ réduire le temps d'exécution (wall-clock time)
 - ✓ optimiser l'utilisation des ressources (CPU time)
 - ✓ aussi réduire l'utilisation mémoire, la place disque
- ✓ Trouver les **points chauds** et éliminer les **goulots d'étranglement**
- ✓ Point chaud
 - ✓ zone de code utilisant beaucoup de temps CPU
- ✓ Goulot d'étranglement
 - ✓ zone de code utilisant inefficacement les ressources
- ✓ Processus itératif et interactif
 - ✓ boucle
 - ✓ homme dans la boucle

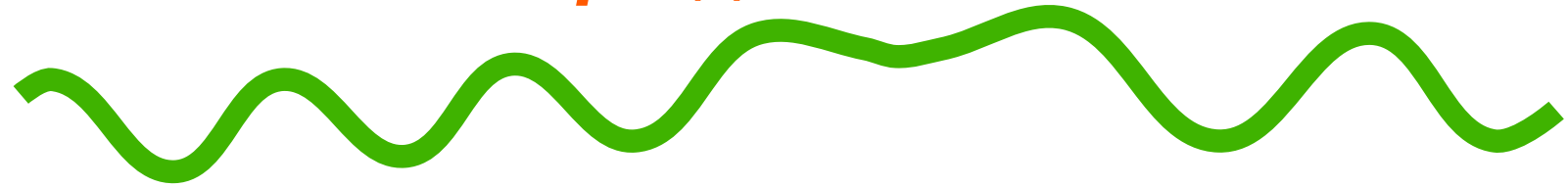
Prise de temps (1)



- ✓ Temps total d'exécution d'un programme
 - ✓ temps écoulé (elapsed time, real time, wall-clock time)
 - ✓ temps utilisateur : temps CPU utilisé par les instructions du programme
 - ✓ temps système : temps CPU utilisé par le système d'exploitation pour le compte du programme
- ✓ Commande Unix `time`
 - ✓ exemple :

```
% time monprogramme
real    2m2.65s
user    1m1.48s
sys     0m4.61s
```
 - ✓ commandes internes `time` des shells
 - ✓ affichage différent suivant le shell (`tcsh`, `ksh`, variable `time` du shell...)
- ✓ Commande `timex`
 - ✓ informations complémentaires : I/O, attente...

Prise de temps (2)



✓ Chronomètres, fonctions départ et arrêt

✓ Fonction Unix `gettimeofday()`

✓ résolution dépendante du système (microseconde sur IBM SP)

✓ utilisation dans un code C

```
#include <sys/time.h>
```

```
#include <unistd.h>
```

```
struct timeval start, end ;
```

```
long int useconds;
```

```
gettimeofday (&start, (struct timeval*)0);
```

```
/* ... code a mesurer ... */
```

```
gettimeofday (&end, (struct timeval*)0);
```

```
useconds = (end.tv_sec - start.tv_sec) * 1000000
```

```
          + end.tv_usec - start.tv_usec ;
```

✓ appel depuis Fortran via une fonction C

Prise de temps (3)



~ Fonctions `read_real_time()` et `time_base_to_time()` sur AIX

~ haute résolution (nanoseconde sur l'IBM SP)

~ utilisation dans un code C

```
#include <sys/time.h>
```

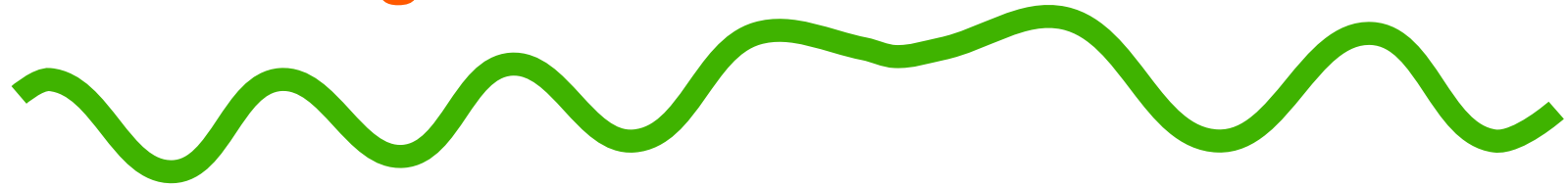
```
timebasestruct_t start, end ;  
long int seconds, nseconds;
```

```
read_real_time (&start, TIMEBASE_SZ);  
/* ... code a mesurer ... */  
read_real_time (&end, TIMEBASE_SZ);
```

```
/* normalisation */  
time_base_to_time (&start, TIMEBASE_SZ);  
time_base_to_time (&end, TIMEBASE_SZ);
```

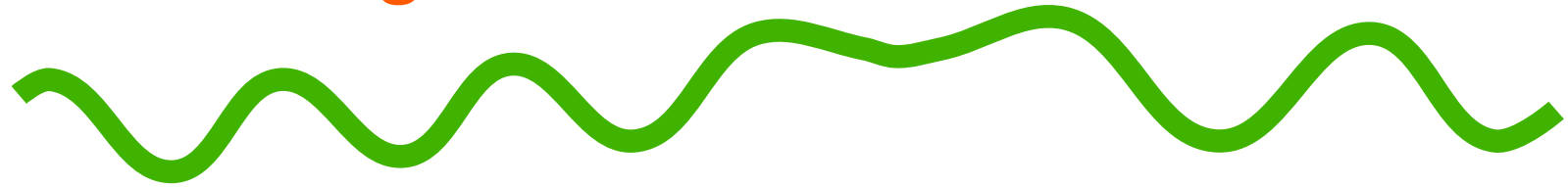
```
seconds = end.tb_high - start.tb_high;  
nseconds = end.tb_low - start.tb_low;  
if (nseconds < 0) {  
    seconds--; nseconds += 1000000000;  
}
```

Profiling (1)



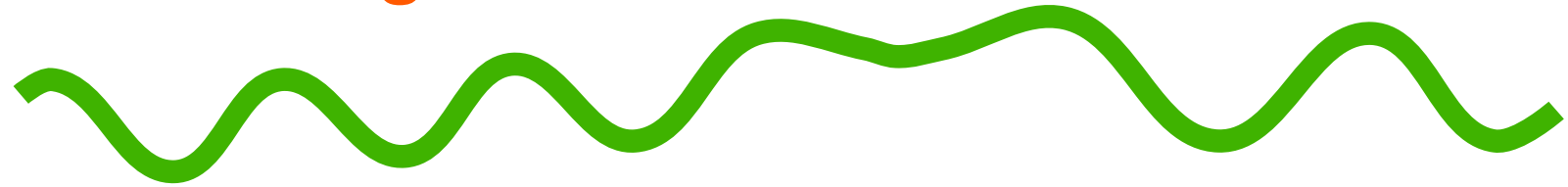
- ~ Instrumentaliser le code
- ~ Exécuter le programme
 - ~ génération de fichiers de statistiques
- ~ Visualiser ces statistiques
 - ~ arbre des appels de fonctions
 - ~ pourcentage de temps CPU utilisé
 - ~ pour chaque sous arbre (fonction et sa descendance)
 - ~ pour chaque instance de fonction
 - ~ pour chaque fonction
 - ~ nombre d'appel à chaque fonction
- ~ Résultat pour l'exécution donnée
 - ~ état de la machine
 - ~ jeu de données du programme
 - ~ perturbation par observation

Profiling (2)



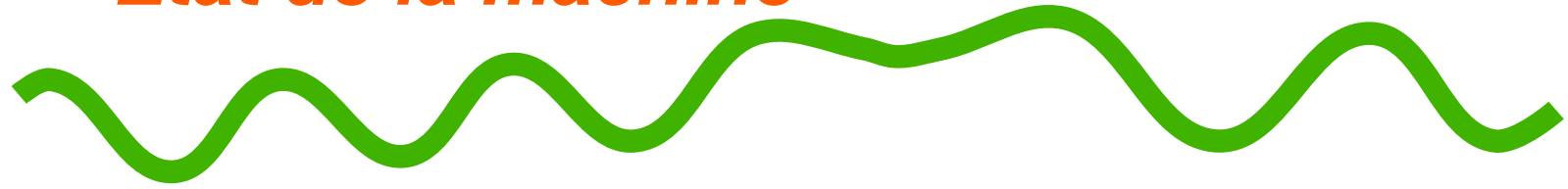
- ~ Commande Unix `prof`
- ~ compilation avec l'option `-p`
- ~ fichiers de statistiques
 - ~ fichier `mon.out`
 - ~ fichiers `mon.out.0`, `mon.out.1` si plusieurs processus
- ~ commande `prof` d'affichage textuel des statistiques
 - ~ `prof`
 - ~ `prof -m mon.out`
 - ~ `prof -m mon.out.0 mon.out.1`
 - ~ `prof -m mon.out.*`
 - ~ plusieurs fichiers de statistiques : moyenne
 - ~ utilise le fichier binaire pour exploiter les statistiques

Profiling (3)



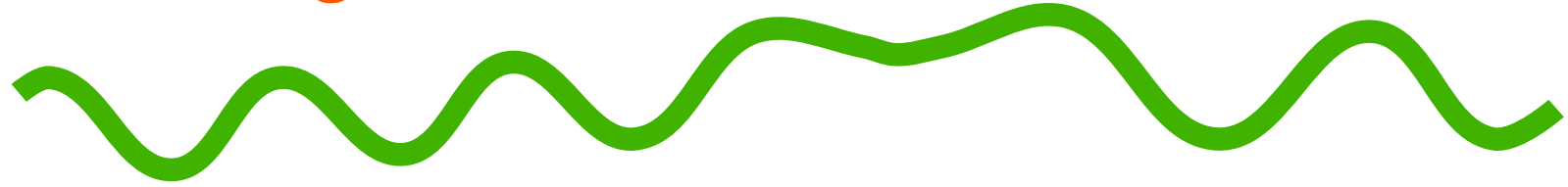
- ~ Commande Unix `gprof` — graphical `prof`
 - ~ compilation avec l'option `-pg`
 - ~ fichiers de statistiques
 - ~ fichier `gmon.out`
 - ~ fichiers `gmon.out.0`, `gmon.out.1` si plusieurs processus
 - ~ commande `gprof` d'affichage graphique des statistiques
 - ~ `gprof monprogramme`
 - ~ `gprof monprogramme -m mon.out.*`
- ~ Commande AIX `tprof`
 - ~ analyse au niveau de la ligne de source
- ~ Outil visuel `xprofiler`
 - ~ couche X Windows au dessus de `gprof`

État de la machine



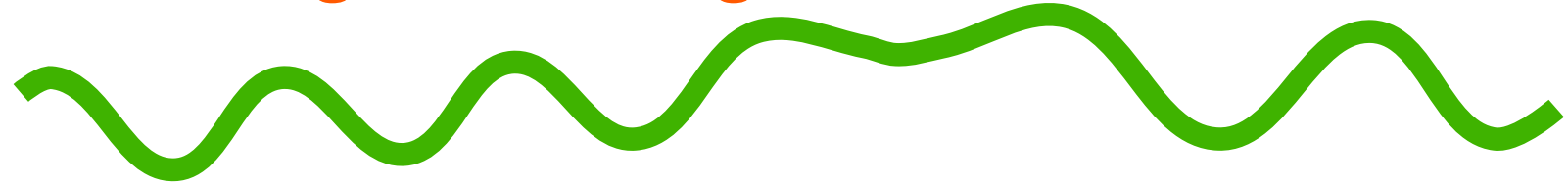
- ~ Configuration matérielle
- ~ Configuration logicielle
- ~ Utilisation des ressources

Configuration matérielle de l'IBM SP



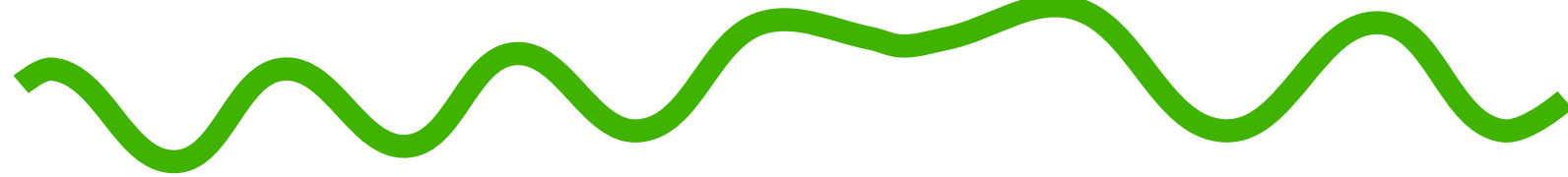
- ✓ Demander aux administrateurs système !
- ✓ Liste du matériel installé
 - ✓ commande `lscfg`
 - ✓ processeurs, carte mémoire, périphériques (disques...)
- ✓ Informations à propos des processeurs
 - ✓ commande `qcpu`
 - ✓ type de processeur
 - ✓ taille des mémoires caches

Configuration logicielle de l'IBM SP



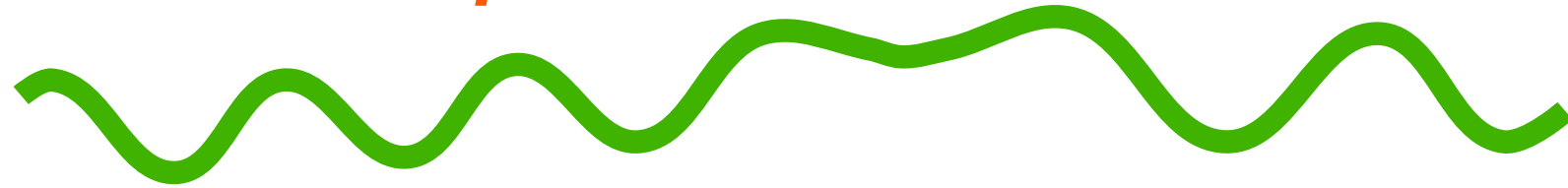
- ~ Demander aux administrateurs système !
- ~ Commande `lslpp, lslpp -l`

Utilisation des ressources



- ✔ Utilisation des ressources préalablement à l'exécution d'un programme
 - ✔ équilibre de la charge entre différents nœuds
 - ✔ charger raisonnablement un nœud
 - ✔ système d'équilibre automatique : LSF – Load Sharing Facilities
non (encore ?) installé à Lille
- ✔ Utilisation exclusive des ressources
 - ✔ utilisation non-interactive de la machine
 - ✔ file d'attente des travaux
 - ✔ risque de gaspillage des ressources
 - ✔ non-mis en place à Lille
- ✔ Quelle utilisation des ressources par mon programme

Liste des processus



- ✓ Pour un nœud de la machine
- ✓ Liste des processus – programmes s'exécutant

- ✓ commande `ps`
- ✓ nombreuses options
- ✓ exemple :

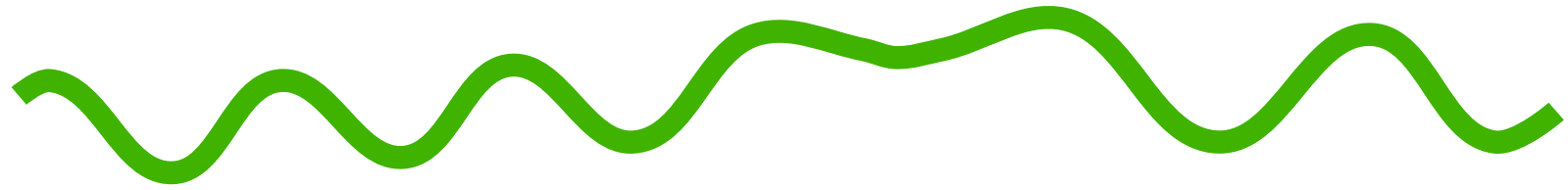
```
% ps ux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
marquet       681  0.0  1.0   2084  1288 tty1     S    08:33   0:00 -tcsh
marquet      1067  0.1  5.3   8488  6868 tty1     S    08:34   0:20 emacs
marquet      1727  0.0  0.6   2464   884 pts/0    R   12:22   0:00 ps ux
```

- ✓ Identification des processus les plus actifs
 - ✓ commandes `top` et `monitor`
 - ✓ tri par temps CPU ou utilisation mémoire
 - ✓ pourcentage d'utilisation des CPU
 - ✓ monitoring
- ✓ Utilisation de la mémoire virtuelle
 - ✓ commande `vmstat`

Visualisation avec VT

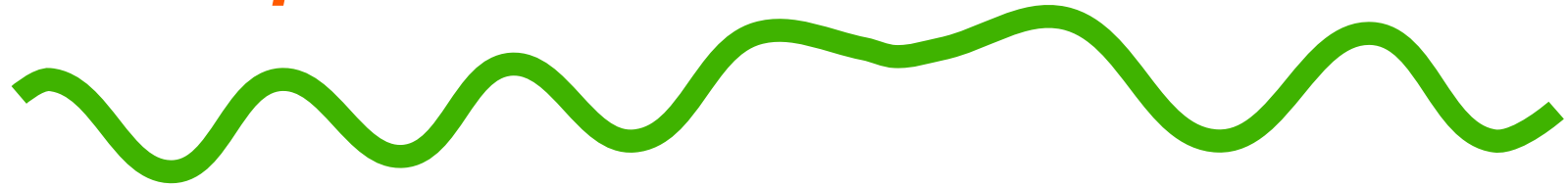


- ✔ VT – visualization tool
 - ✔ visualisation graphique (outils X Windows)
 - ✔ prise en compte (voire vue globale) de plusieurs nœuds de la machine
- ✔ Deux utilisations
 - ✔ visualisation de l'exécution d'un programme
 - ✔ visualisation de l'état de la machine
- ✔ Deux modes
 - ✔ visualisation de traces : analyse post-mortem
 - ✔ monitoring : analyse temps-réels
- ✔ Nombreux types de vues
 - ✔ état des processeurs : utilisateur/système/libre/attente
 - ✔ communication entre les processus d'un programmes
 - ✔ activité système : changements de contexte, défaut de page...
 - ✔ activité réseau, activité des disques
- ✔ Choix des vues préférées...



Optimisations

Optimiser : est-ce le rôle du compilateur ?



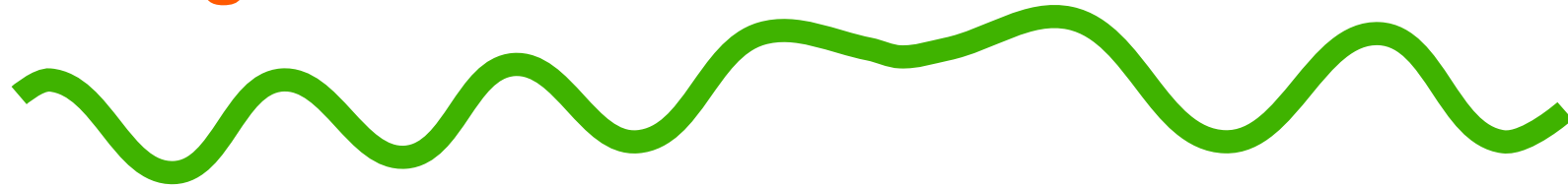
- ~ Optimisation du code généré
 - ~ travail du compilateur
 - ~ respect obligatoire des dépendances de données
 - analyse du code pour trouver ces dépendances
 - ~ respect de l'ordre des calculs (vs performances, mul-add...)
 - optionnel, problèmes numériques (arrondis)
- ~ Pourquoi le compilateur ne peut pas tout faire
 - ~ fausses dépendances
 - ~ connaissance du programmeur (vecteurs d'indices)
 - ~ analyse interprocédurale
 - ~ ordre des calculs à respecter ?
- ~ Optimisation manuelle
 - ~ optimisation du code généré
 - ~ compréhension du schéma d'exécution
 - ~ en modifiant le code source !
 - ~ compréhension du schéma de génération de code

Hiérarchies mémoires



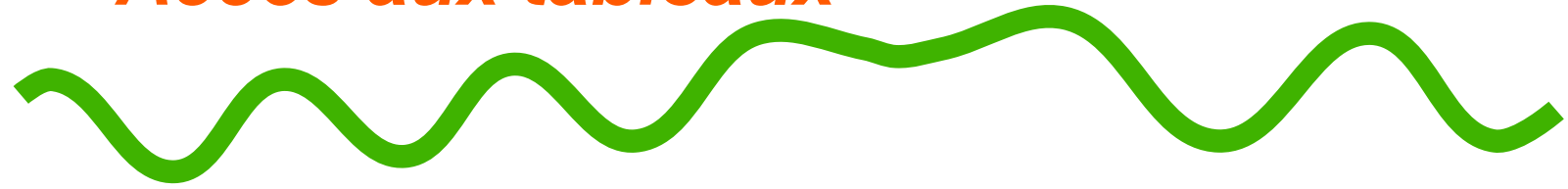
- ✔ Comment faire un grosse mémoire bon marché mais rapide ?
 - ✔ intercaler une petite mémoire rapide (donc chère)
 - ✔ entre le processeur et une grande mémoire bon marché
 - ✔ fonctionne remarquablement bien
- ✔ Hiérarchie à plusieurs niveaux
 - ✔ registres > cache L1 > cache L2 > (cache L3) > mémoire centrale > mémoire d'échange sur disque
 - ✔ image partielle des niveaux supérieurs dans les niveaux inférieurs
- ✔ Défaut de cache
 - ✔ cache organisé en ligne de cache
 - ✔ donnée non présente dans le cache \Rightarrow transfert depuis la mémoire
 - ✔ pas de place dans le cache \Rightarrow transfert du cache vers la mémoire
 - ✔ transfert de lignes entre la mémoire et le cache
- ✔ Exploiter la localité
 - ✔ spatiale : accès à une adresse voisine
 - ✔ temporelle : accès successifs à une même adresse

Organisation des caches



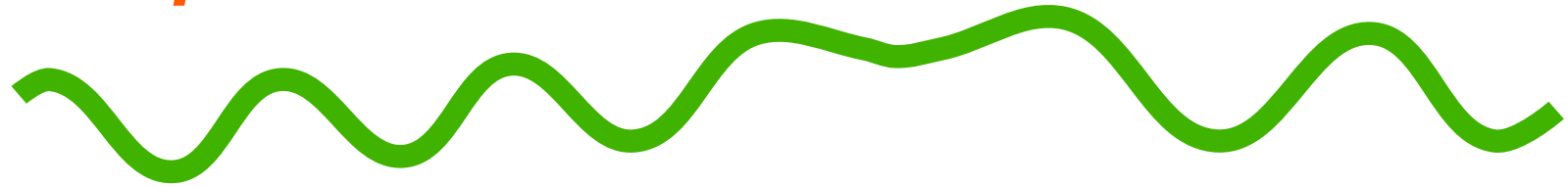
- ✓ Organisation du cache en « ensembles associatifs »
 - ✓ *cache set associativity*
 - ✓ un ensemble associatif dans le cache correspond à toutes les adresses mémoire égales à un modulo près
 - ✓ un ensemble associatif est de taille limité
on parle du nombre de voies d'un cache
cache du Power 2 : *4-way cache set associativity*
cache du Power 3 : *128-way cache set associativity*
 - ✓ lors de la descente dans le cache d'une adresse, il faut trouver une place libre dans *son* ensemble associatif
- ✓ Inconvénient de l'organisation en ensembles associatifs
 - ✓ accès à des lignes de matrices (Fortran)
 - ✓ une ligne de matrice = une ligne de cache
 - ✓ plusieurs lignes de même adresse modulo une puissance de 2
 - ✓ toutes les adresses des lignes tombent dans le même ensemble associatif !
 - ✓ solution : matrice de lignes de taille $2^n + 1$

Accès aux tableaux



- ✓ Placement des tableaux en mémoire
 - ✓ C : suite de lignes
 - ✓ F77 : suite de colonnes
 - ✓ F90 : pas de placement spécifié
- ✓ Localité spatiale : utiliser successivement des données contiguës en mémoire
 - ✓ C : boucle interne sur les colonnes
 - ✓ F77 : boucle interne sur les lignes
 - ✓ pas d'accès le plus petit possible
- ✓ Localité temporelle : réutiliser plusieurs fois les mêmes données
 - ✓ Calculs par blocs de taille adaptée aux niveaux rapides de la hiérarchie mémoire

Optimisations des boucles



- ✓ Concentrent en peu de lignes un maximum de temps de calcul
- ✓ Optimisations « élémentaires »
 - ✓ permutation, fusion, partitionnement, factorisation, réunion, déroulage, ...
 - ✓ valides seulement si elles respectent les dépendances entre les données
- ✓ Parallélisation automatique
 - ✓ analyse de dépendances, ordonnancement, placement, génération de code
 - ✓ trouve du parallélisme bien caché
 - ✓ garantie d'optimalité sur le domaine d'application
 - ✓ pas présent dans les compilateurs car
 - ✓ code généré « trop compliqué »
 - ✓ redistribution éventuelle des données
 - ✓ les cas réels sont ou trop simples ou trop compliqués !

Permutation de boucles



```
DO i = 1, N
  DO j = 1, N
    A(i,j) = 1/B(i,j)
  END DO
END DO
```

```
DO j = 1, N
  DO i = 1, N
    A(i,j) = 1/B(i,j)
  END DO
END DO
```

- ✔ utile pour aligner les accès aux tableaux avec le placement en mémoire
- ✔ pas toujours légal

Partitionnement de boucles

```
DO j = 1, N
  DO i = 1, N
    A(i,j) = 1/B(i,j)
  END DO
END DO
```

```
DO jj = 1, N, sj
  DO ii = 1, N, si
    DO j = jj, jj+sj-1
      DO i = ii, ii+si-1
        A(i,j) = 1/B(i,j)
      END DO
    END DO
  END DO
END DO
```

- ↪ adaptation de la granularité à la hiérarchie mémoire
- ↪ calcul par blocs (pas forcément alignés sur les axes)
- ↪ légal quand toutes les boucles sont permutable

Fusion/distribution de boucles

```
DO i = 1, N
  A(i) = ...
  B(i) = ... A(i) ...
END DO
```

```
DO i = 1, N
  A(i) = ...
END DO
DO i = 1, N
  B(i) = ... A(i) ...
END DO
```

- fusion augmente la localité temporelle, élimine des tableaux temporaires
- distribution permet de construire des boucles parallèles
- l'une ou l'autre des formes est préférable selon les cas

Partage de l'espace d'itération

```
DO i = 1, N
  A(a*i+b) = A(c)+...
END DO
```

```
Ic = (c-b)/a
DO// i = 1, Ic-1
  A(a*i+b) = A(c)+...
END DO
A(c) = A(c)...
DO// i = Ic+1, N
  A(a*i+b) = A(c)+...
END DO
```

- ✓ utilisation typique : se débarrasser d'itérations problématiques
- ✓ exemple ci-dessus
- ✓ test à l'intérieur de la boucle pour une seule itération
- ✓ ...

Expansion des scalaires



```
DO i = 1, N
  T=f(i)
  A(i) = A(i)+T*T
END DO
```

```
DO// i = 1, N
  T[i]=f(i)
  A(i) = A(i)+T[i]*T[i]
END DO
```

- peut aller jusqu'à la mise en assignation unique
- élimine de « fausses » dépendances dues à la réutilisation de la mémoire

Déroutage de boucles

```
DO i = 1, N  
  A(i) = ...  
END DO
```

```
DO i = 1, N, 4  
  A(i) = ...  
  A(i+1) = ...  
  A(i+2) = ...  
  A(i+3) = ...  
END DO
```

- réduit le surcoût de contrôle des boucles
- donne des possibilités de pipeline et autres optimisations

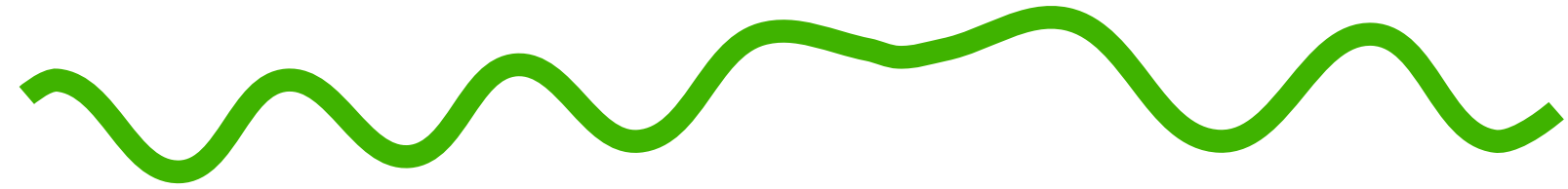
Écrasement de boucles



```
int a[100][300];  
  
for (i=0; i<300; i++)  
    for (j=0; j<100; j++)  
        a[j][i] = 0;
```

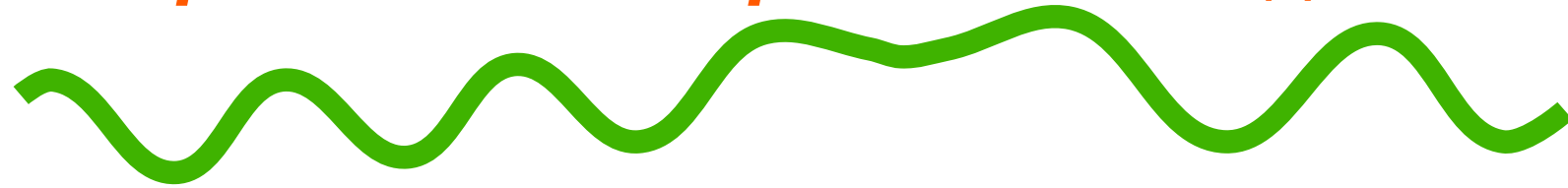
```
int a[100][300];  
int *p = &a[0][0];  
  
for (i=0; i<30000; i++)  
    *p++ = 0;
```

- réduit le surcoût de contrôle des boucles
- peut donner des opportunités de déroulage



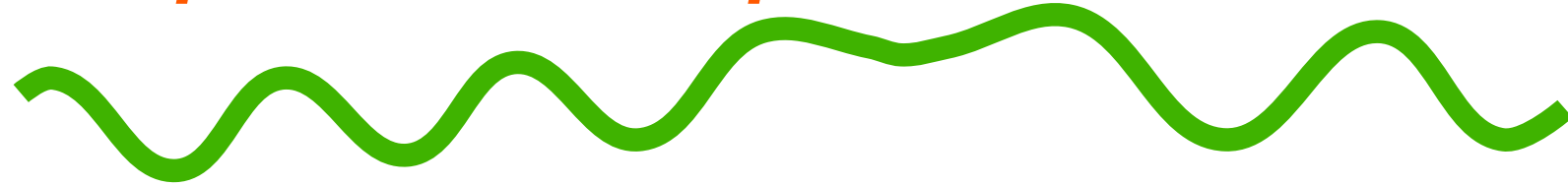
Compilateurs IBM Fortran et C

Options des compilateurs IBM (1)



- ✔ Compilateurs Fortran `xlf` et C `xlc`
- ✔ Niveau d'optimisation `-O` ou `-O2`
 - ✔ par défaut pas d'optimisation
 - ✔ allonge le temps de compilation et la possiblement taille du code généré...
- ✔ Niveau d'optimisation `-O3`
 - ✔ éventuel non respect de la sémantique/norme
 - ✔ ré-arrangement des expressions arithmétiques
 - ✔ suppression des expressions constantes des boucles `DO`
 - ✔ remplacement des divisions par des multiplication par l'inverse
- ✔ Respect de la sémantique et de la norme
 - ✔ option `-qstrict`
 - ✔ limite l'agressivité de `-O3`

Options des compilateurs IBM (2)



~ Niveaux -O4 et -O5 : alias

- ~ -O3 -qhot -qipa -qarch=auto -qtune=auto -qcache=auto
- ~ et -O4 -qipa=level=2

~ Choix d'architecture

- ~ options -qarch=*arch* et -qtune=*arch*
- ~ optimisations spécifiques à l'architecture *arch*
- ~ com : commun, portabilité
- ~ auto : détection de l'architecture et optimisation pour celle-ci
exécution sur la machine de compilation
- ~ pwr, pwr2, pwr3... optimisation pour l'architecture donnée

~ Analyse interprocédurale

- ~ option -qipa
- ~ à passer aussi à l'édition de liens

Options des compilateurs IBM (3)



✓ Opérations flottantes

- ✓ option `-qfloat`
- ✓ non respect de la norme IEEE

✓ Parallélisation automatique

- ✓ option `-qsmp` de `xlf_r` ou `xlf90_r`
- ✓ en vue d'une exécution sur un SMP
- ✓ parallélisation de certaines boucles DO

✓ Optimisation agressive sur les boucles

- ✓ option `-qhot`
- ✓ pas toujours sûre

✓ Rapport de compilation

- ✓ option `-qreport=report`
- ✓ listing des parallélisations effectuées et des optimisations de boucles
- ✓ `report` prend les valeurs `smp` ou `hot`

Compilateurs IBM, quelles options ?



- ✓ Code de production
 - ✓ -O2 ou -O3 -qstrict
- ✓ Optimisations avancées
 - ✓ -O4, -qhot...
 - ✓ compilation avec et sans l'option
 - ✓ comparer les résultats de l'exécution
 - ✓ comparer les temps d'exécution
- ✓ xxlf : interface graphique