Analyse et optimisation des programmes de calculs hautes performances

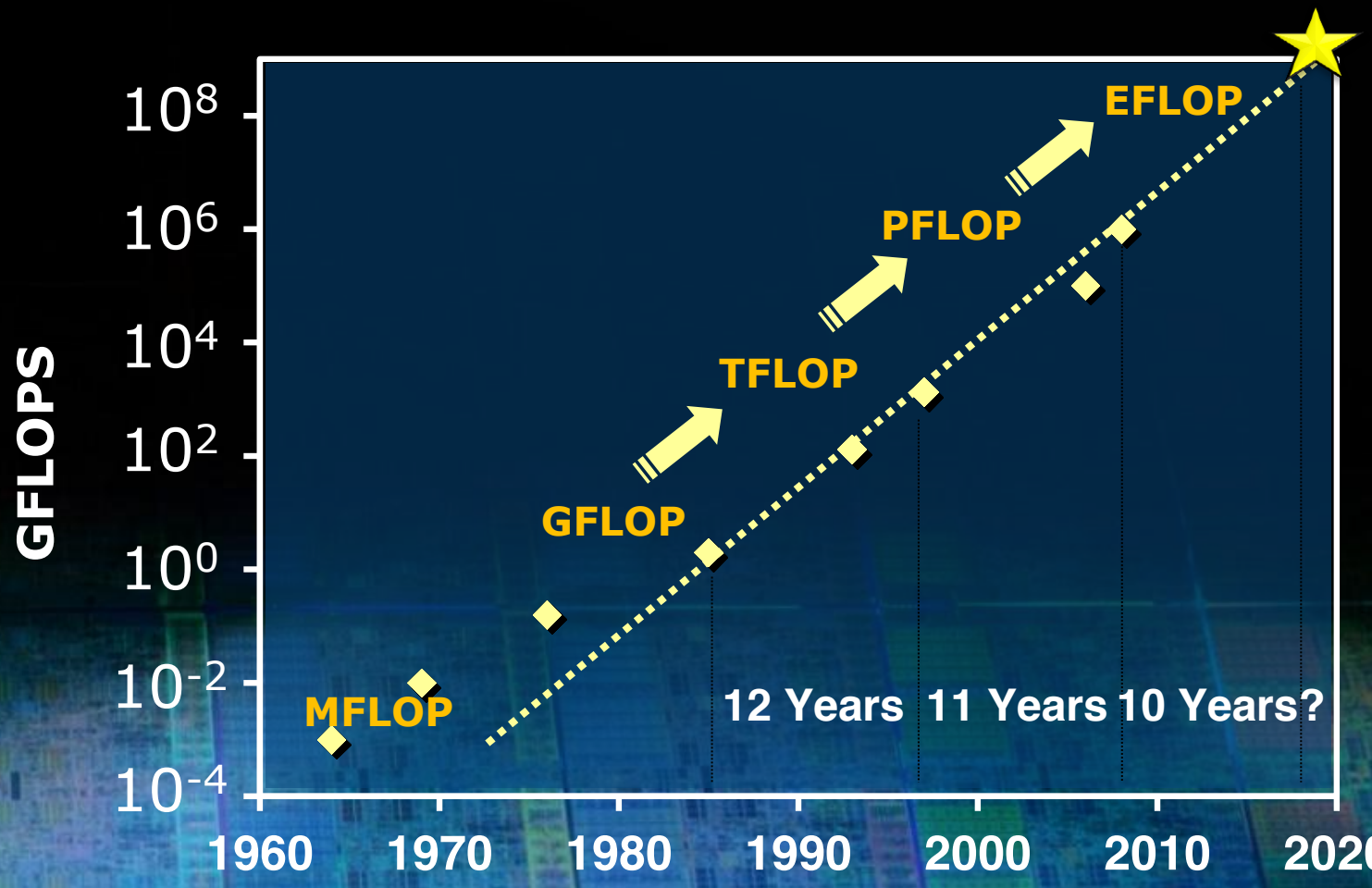
Pr Sid TOUATI, UFR des sciences, département d'informatique

I3S/INRIA

Plan

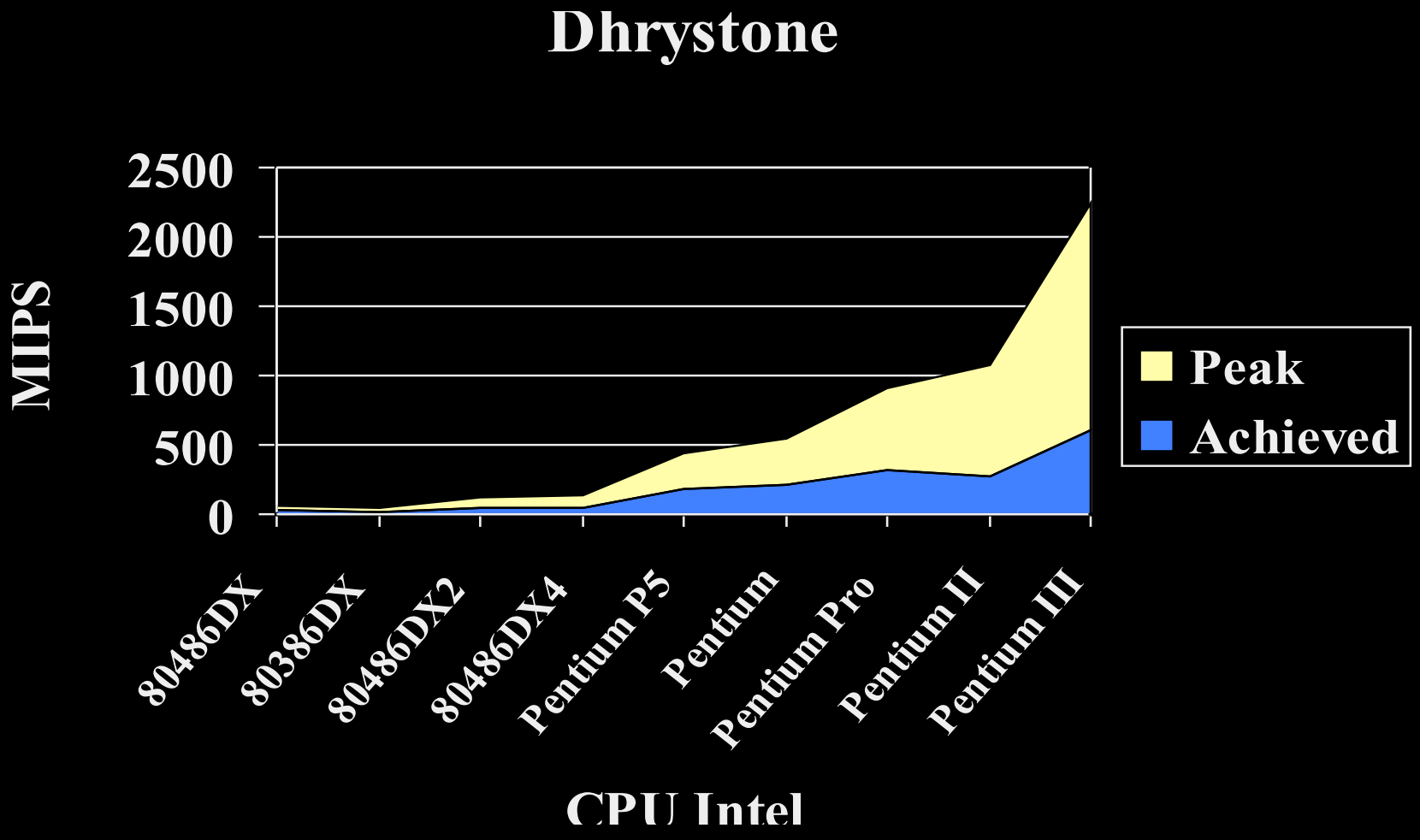
- HPC vs. Calculs intensifs
- Algorithmique et programmation
- Amélioration des performances logicielles
- Parallélisme
- Compilation et optimisation de code
- Analyse statistique des performances
- Conclusion

Today's Exascale Expectation

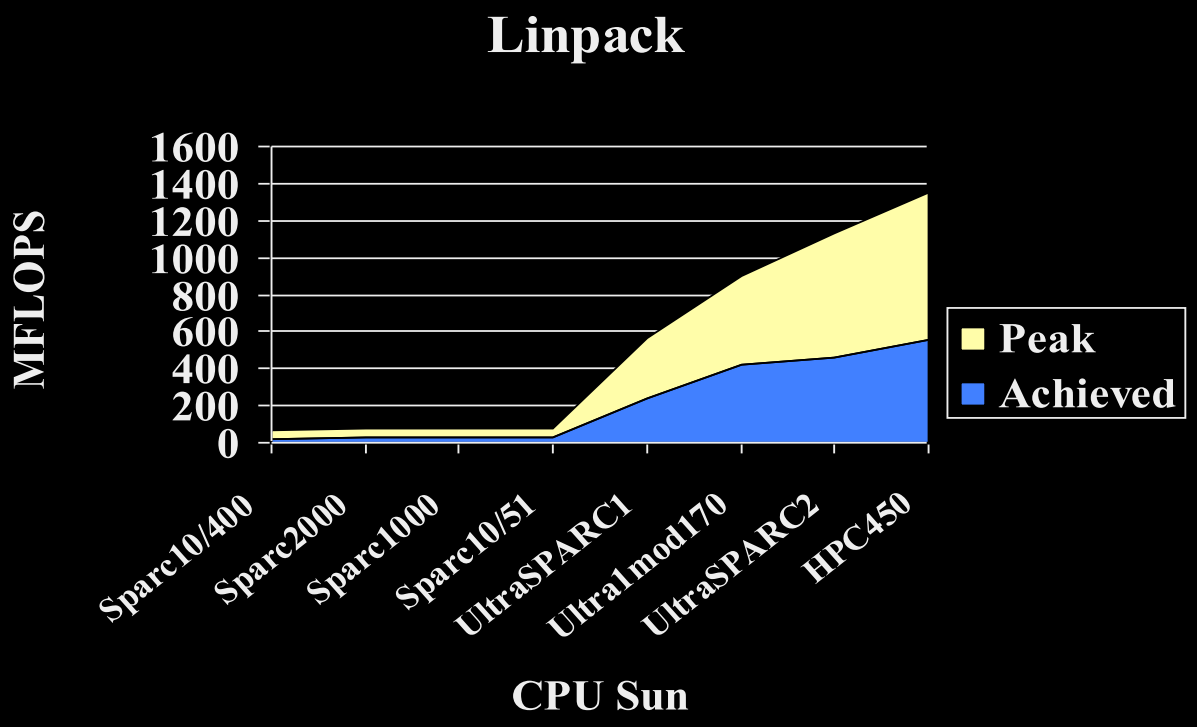


Slide de Dr. Jean-Laurent Philippe, Intel

Performance crête vs. Performance observée par programme



Performance crête vs. Performance observée par programme



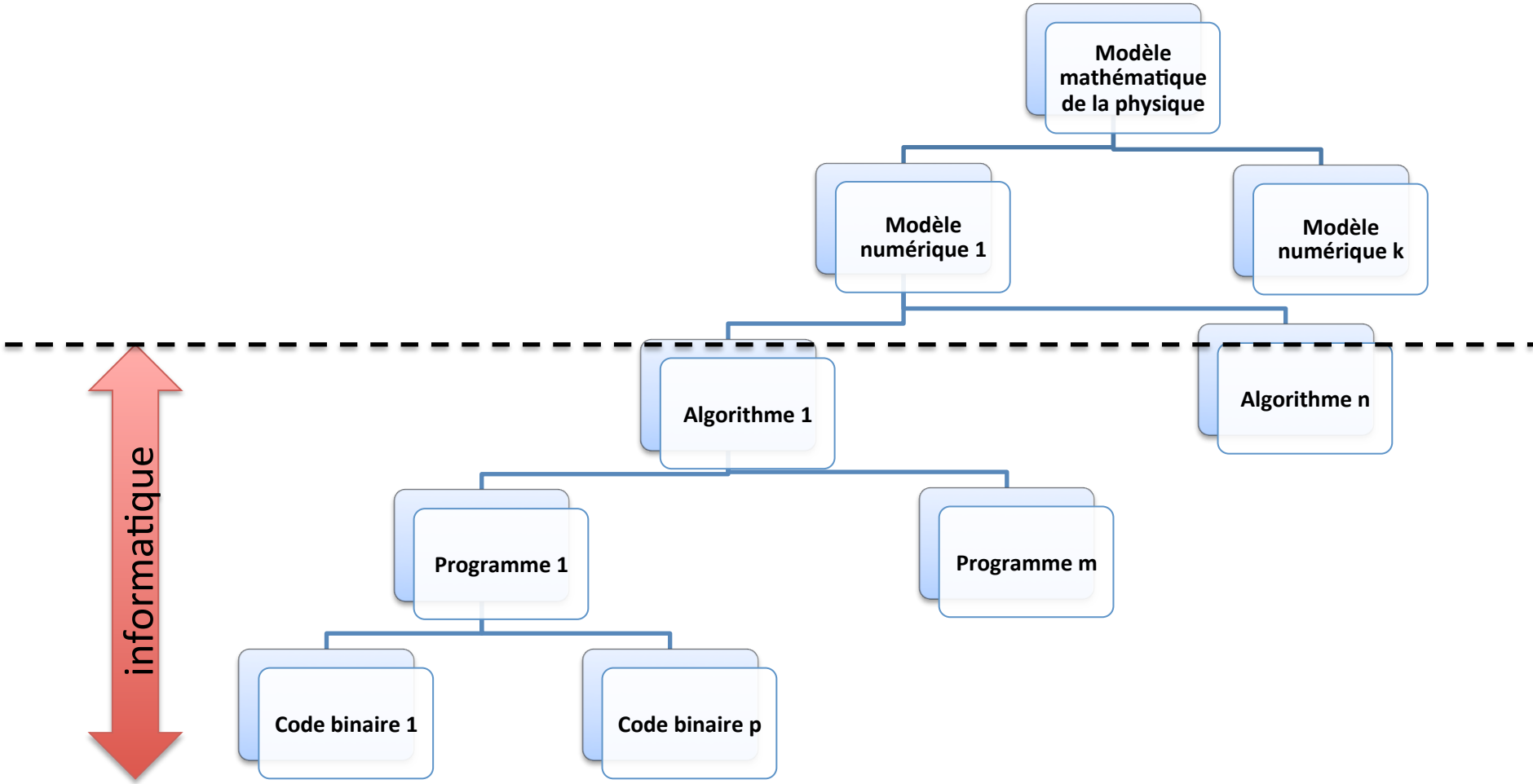
Différences entre le HPC et les calculs intensifs

HPC	Calculs intensifs
Historiquement dirigé par les constructeurs de supercalculateurs	Historiquement dirigés par les applicatifs et les personnes qui font des OS
Communautés impliquées : architectures des processeurs, micro-architecture, compilateurs, OS bas niveau, parallélisation automatique	Communautés impliquées : simulation numérique, applicatifs, OS, calculs sur grilles, parallélisme, systèmes distribués
Métrique de performance : cycles processeur, temps d'exécution (secondes)	Métriques de performance indirectement liées au temps d'exécution : indicateurs algorithmiques, parallélisme haut niveau, taux d'occupation des CPU, quantité de communications, etc.
Approche d'optimisation des performances : bottom-top (très proche du matériel, optimisations bas niveau du compilateur)	Approche d'optimisation des performances : top-bottom (peu proche du matériel, plus proche de l'algorithme haut niveau)
Exécution du code binaire directement sur les processeurs physiques	Middleware, virtualisation, interprétation de code
Les performances d'abord, la portabilité du code n'est pas importante	La portabilité du code est un enjeu.

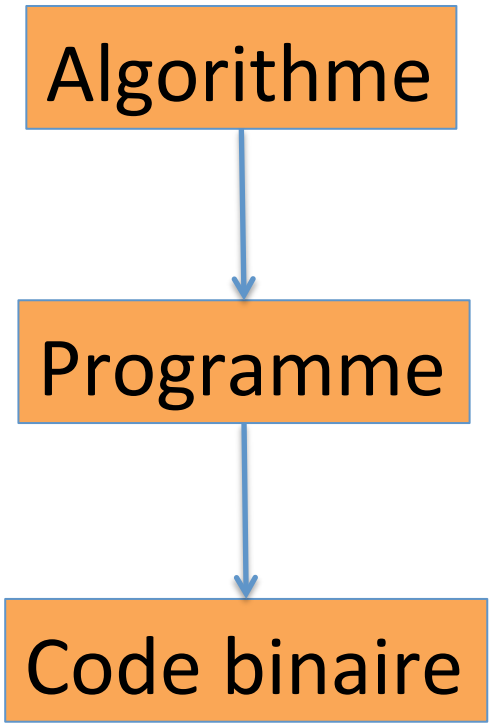
Plan

- HPC vs. Calculs intensifs
- Algorithmique et programmation
- Amélioration des performances logicielles
- Parallélisme
- Compilation et optimisation de code
- Analyse statistique des performances
- Conclusion

Du modèle mathématique de la physique au code informatique exécutable



Algorithme, programme, code binaire



Science du calcul en informatique,
Turing-complet,
complexité algorithmique

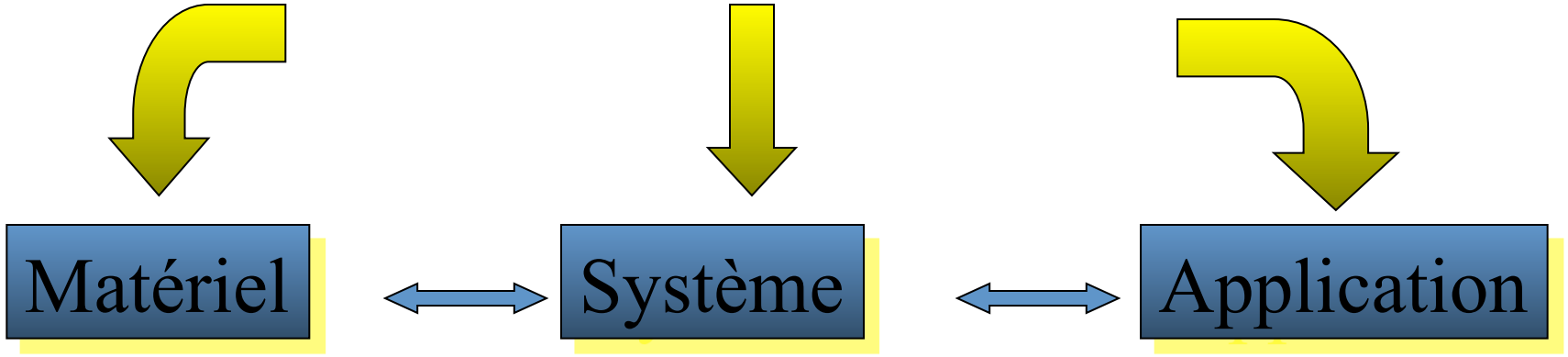
Traduction de l'algorithme dans
un langage de programmation. Code textuel

Compilation et optimisation de code. Code binaire

Plan

- HPC vs. Calculs intensifs
- Algorithmique et programmation
- Amélioration des performances logicielles
- Parallélisme
- Compilation et optimisation de code
- Analyse statistique des performances
- Conclusion

Les leviers d'optimisation des performances d'un programme informatique

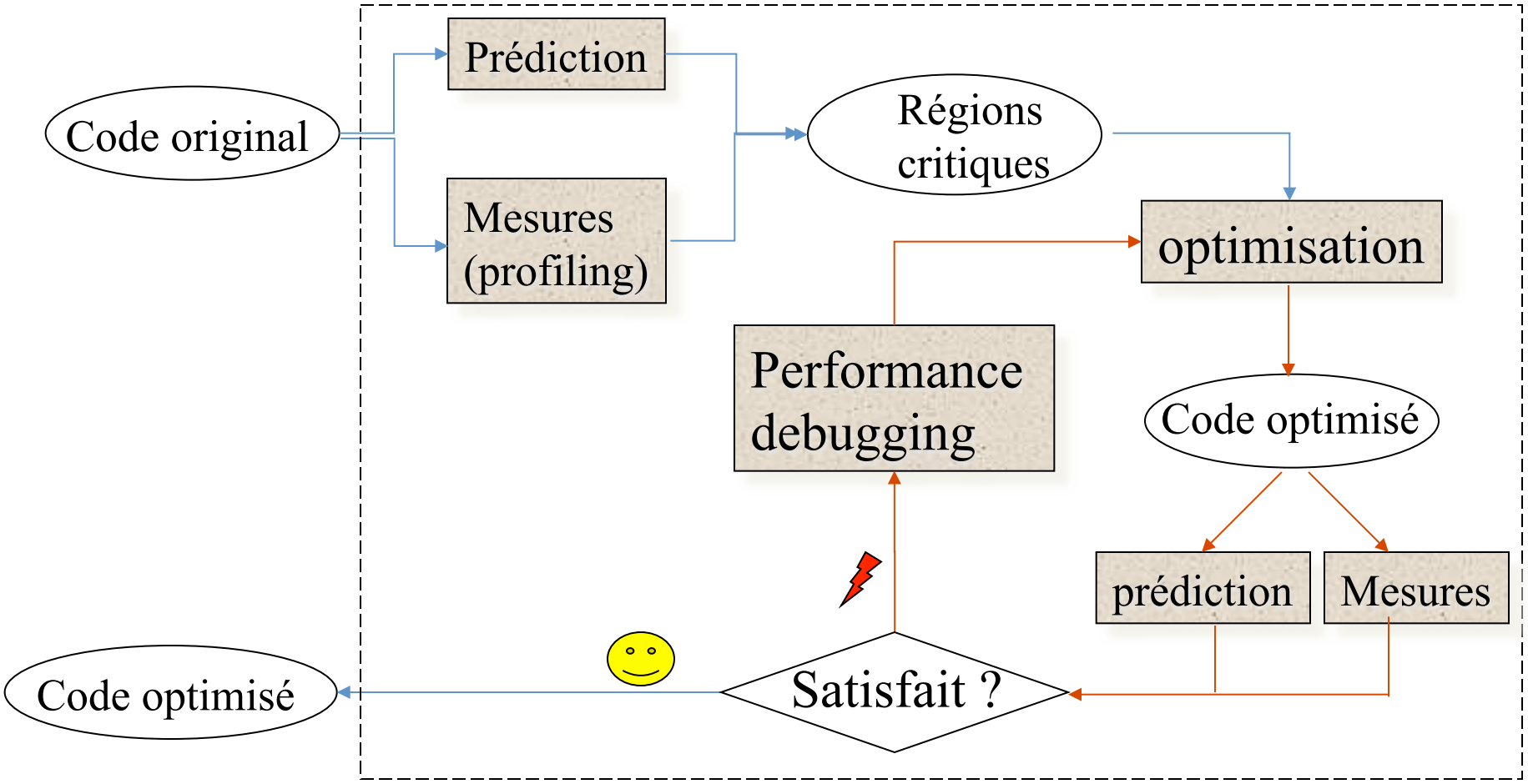


- Electronique
- Architecture proc
- Circuits reconfigurables
- Co-processeur (GPGPU)

- Compilation à la volée
- Ordonnancement de processus et threads
- E/S

- Expert-Programmeur
- Compilation
- Méthode semi-automatique

Une méthodologie simple et itérative d'optimisation des performances d'un logiciel



Comment mesurer les performances

- Appel système: commande time
  - Granularité grossière: temps CPU, temps système, temps utilisateur
- Profilage (utiliser des outils)
  - Granularité moyenne (fonctions)
- Instrumentation
  - Insertion de sondes logiciels au sein du programme
  - Granularité fine
- Compteur matériel de performances
  - Exploiter des registres spéciaux au sein du processeur pour étudier finement l'interaction entre le logiciel et le matériel

Plan

- HPC vs. Calculs intensifs
- Algorithmique et programmation
- Amélioration des performances logicielles
- Parallélisme
- Compilation et optimisation de code
- Analyse statistique des performances
- Conclusion

Choisissez votre langage de programmation parallèle !

ABCPL
ACE
ACT++
Active messages
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BSP
BlockComm
C*.
"C* in C
C**
CarlOS
Cashmere
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL

CORRELATE
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE.
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease
ECO
Eiffel
Eilean
Emerald
EPL
ERLANG
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda

GLU
GUARD
HAsL
Haskell
HPC++
JAVAR.
HORUS
HPF
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold

Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm

Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
PCN
PCP:
PH
PEACE
PCU
PET
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
pthreads
PVM
PSI
PSDM
Quake
Quark
Quick Threads
QPC++
Sage++
SCANDAL
SAM

pC++
SCHEDULE
SciTL
SDDA.
SHMEM
SIMPLE
Sina
SISAL.
distributed
smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
XENOOPS
XPC
Zounds
ZPL

Slide de Dr. Herbert Cornelius , Intel

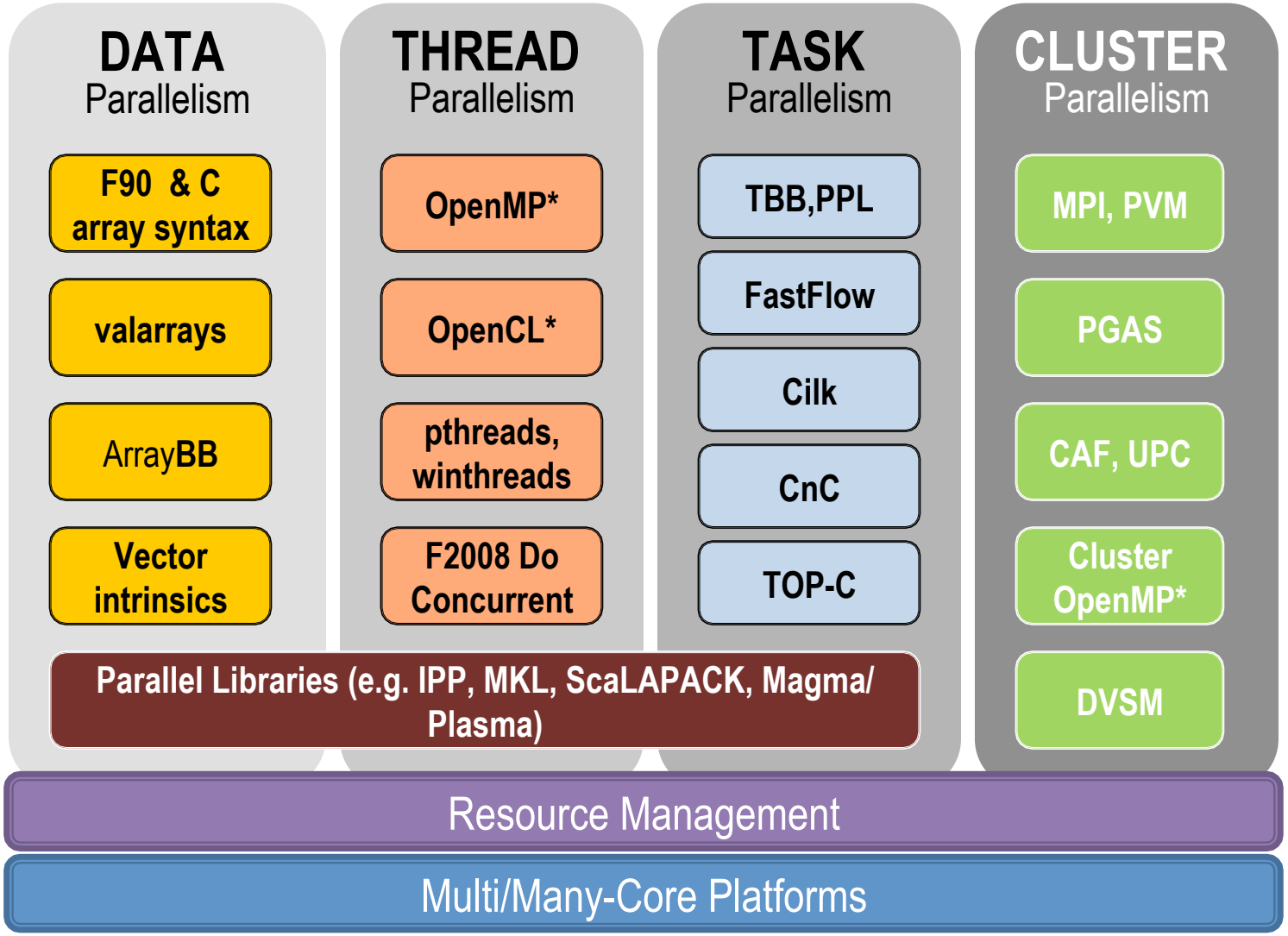
Paradigmes de programmation parallèle

- Parallélisme de données / Parallélisme de tâches
- Parallélisme explicite / parallélisme implicite
- Mémoire partagée / mémoire distribuée
- Autre paradigmes de programmation parallèle
  - Orienté objet
  - Programmation fonctionnelle
  - Programmation logique

Outils logiciels de programmation parallèle

- Parallel Virtual Machine (PVM)
  - Parallélisme explicite, mémoire distribuée
- Message-Passing Interface (MPI)
  - Parallélisme explicite, mémoire distribuée
- PThreads
  - Parallélisme explicite, mémoire partagée
- OpenMP
  - Parallélisme explicite, mémoire partagée
- High-Performance Fortran (HPF)
  - Parallélisme explicite
- Parallélisation automatique par compilation
  - Parallélisme implicite

Quelques langages de programmation parallèle



Slide de Dr. Herbert Cornelius , Intel

Tableau de bord des CPU actifs (vue de l'utilisateur qui a réservé 64 cœurs)

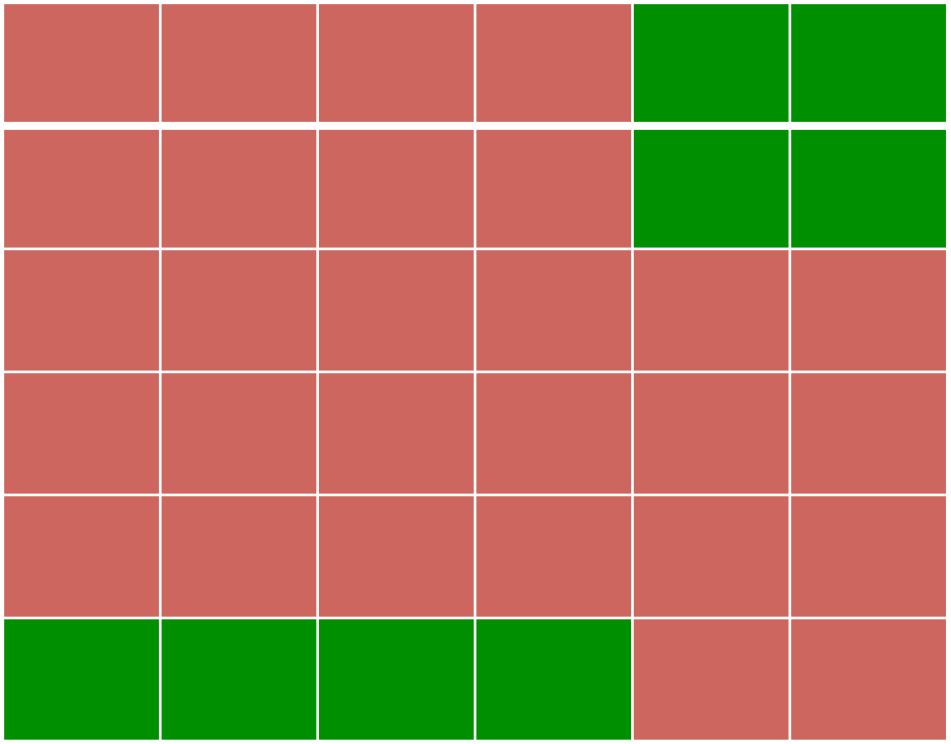
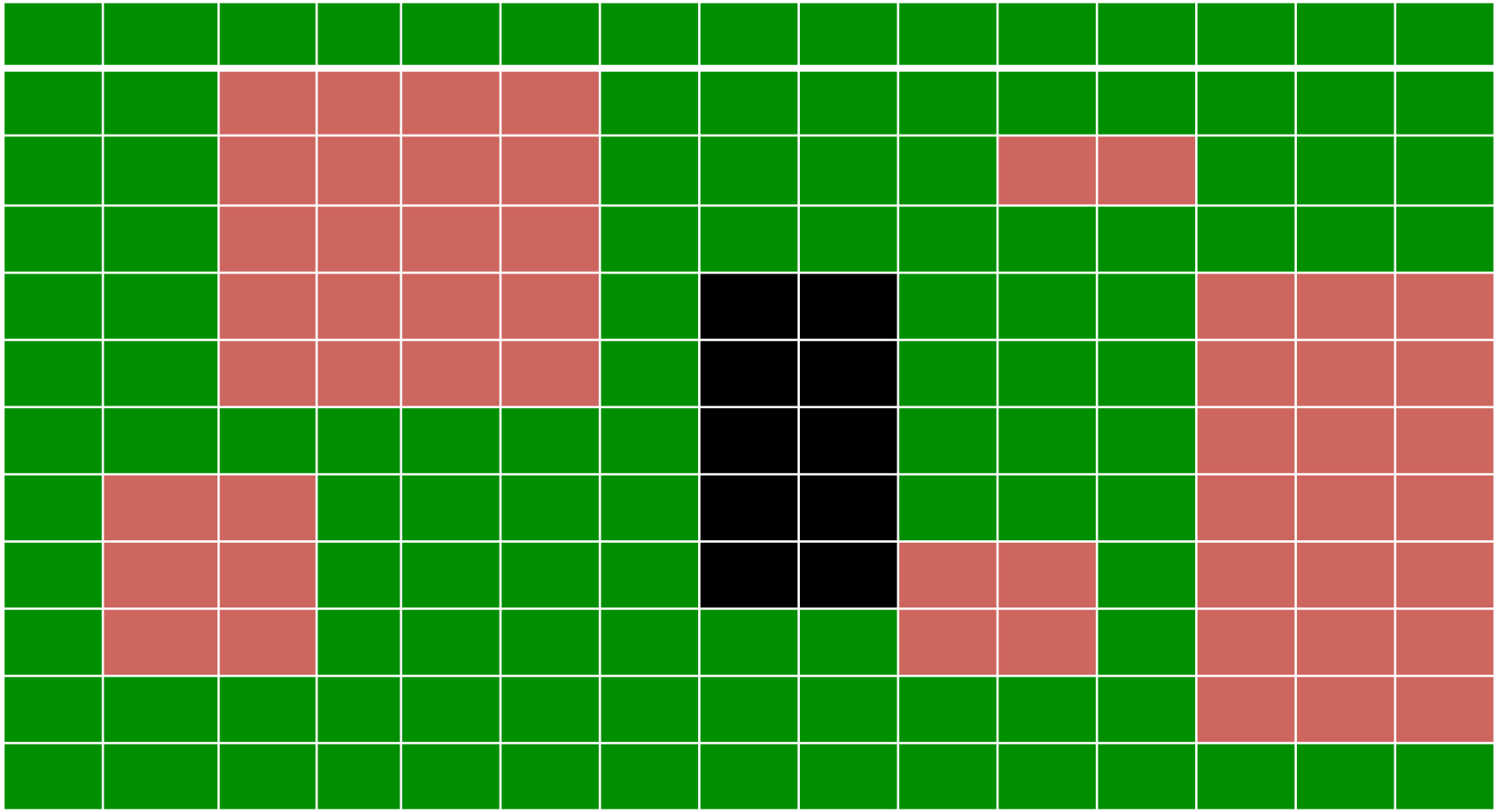
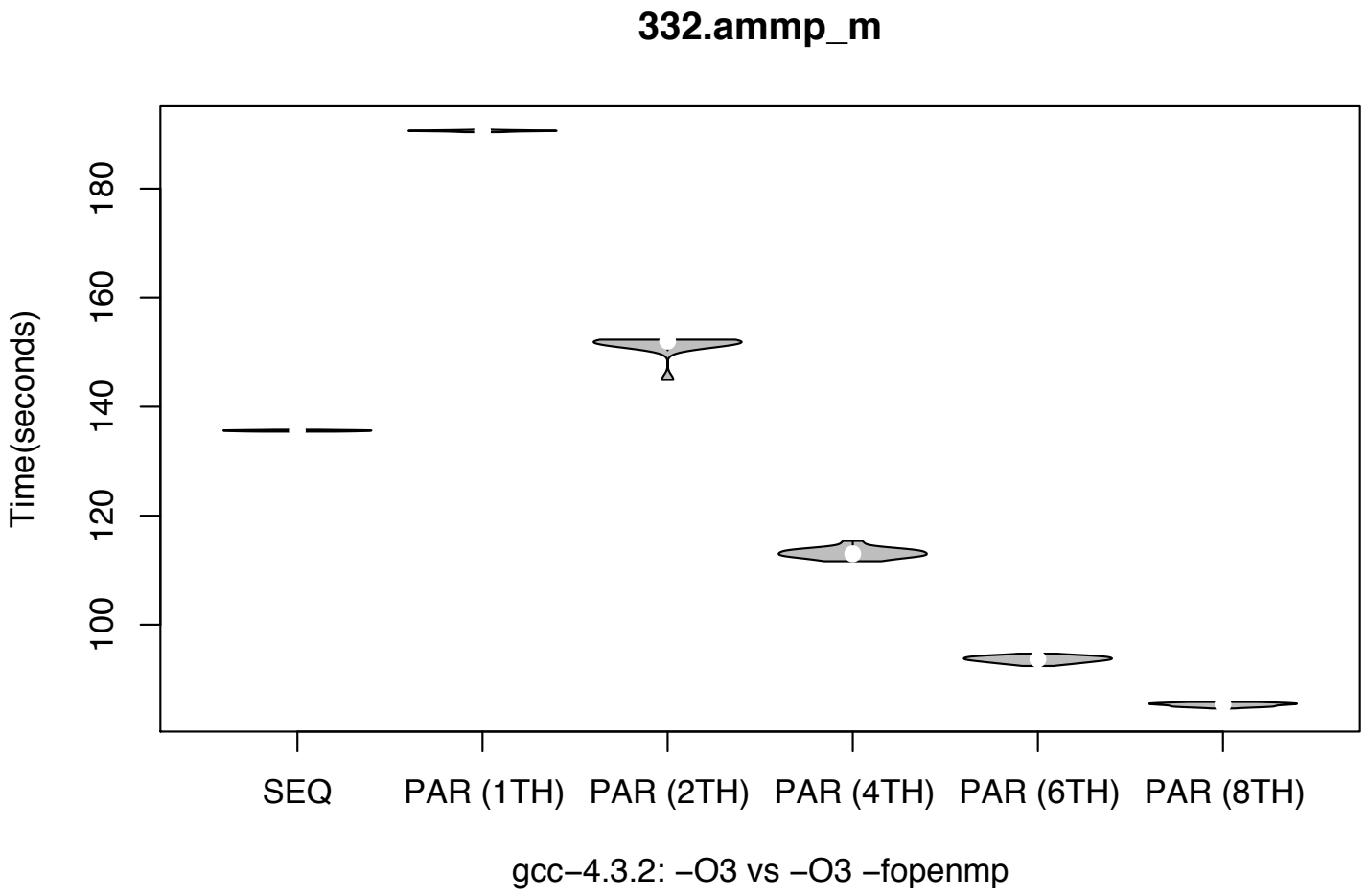


Tableau de bord des CPU actifs, vue du gestionnaire système de la machine



Version séquentielle vs version parallèle d'un programme



Distinguer entre CPU, cœur, socket, processeur

Terme	Interlocuteur	Signification
Processeur	Architecte machine	Terme historique: le composant matériel de la machine qui exécute un programme
CPU	Architecte système	Central Processing Unit = un composant physique ou logique de la machine qui exécute un processus
Socket	Fabricant des puces électroniques	circuit intégré qui peut contenir un ou plusieurs cœurs
Cœur (physique)	Terme commercial	Processeur intégré au sein d'une socket
Cœur logique	Terme technico commercial	processeur logique = CPU vu par le système. (SMT, Hyperthreading)

Plan

- HPC vs. Calculs intensifs
- Algorithmique et programmation
- Amélioration des performances logicielles
- Parallélisme
- Compilation et optimisation de code
- Analyse statistique des performances
- Conclusion

La compilation optimisante

- L'une des branche les plus actives en recherche en compilation depuis les 70's
- Objectif: optimiser la génération de codes exécutables
  - Temps d'exécution
  - Taille de code
  - Consommation d'énergie
  - Obscurcir le code

Avancées architecturales des processeurs pour les performances

- Enrichir le jeu d'instructions assembleur avec des instructions spéciales qui améliorent les performances de certains logiciels
  - Instructions SIMD
  - Instructions multimédia
  - Instructions flottantes
  - Instructions spéculatives
- Parallélisme explicit d'instructions (processeurs VLIW)

Améliorations micro-architecturales des processeurs

- Exécution pipelinée
- Parallélisme implicite des instructions
  - Processeurs superscalaires: parallélisation à la volée des instructions, exécution dans le désordre, renommage des registres, vectorisation de certaines opérations mémoire
- Plusieurs niveaux de caches, partagés ou privés
  - L1I, L1D
  - L2
  - L3
  - Mémoires NUMA, UMA
- Prédiction des branchements
- Exécution spéculative

Un compilateur moderne

- Des centaines d'options de compilation
  - Pour les experts
- Certaines options sont regroupées dans des niveaux -O1, -O2, -O3, etc.
  - Ce sont des niveaux d'optimisation standards
  - Temps de compilation croissant
  - Révélation de bugs dans les programmes
  - Chaque programme peut nécessiter un choix propre judicieux des options de compilation

Trouver la bonne combinaison des options de compilation

- Il est très difficile de préciser avec exactitude l'effet de chaque technique d'optimisation de code sur la performance d'un programme.
  - Parfois, optimiser une partie d'un code peut en dégrader une autre
- Trouver la meilleure combinaison des optimisations de code est un problème indécidable! (prouvé par S. Touati et D. Barthou)

Ecrire un « bon » programme

- Un bon programme est dans l'ordre
  - Un programme correct
  - Un programme efficace
  - Autres: code maintenable, code lisible, etc

Programme bien écrit pour la compilation optimisante

- Les compilateurs ne sont pas des logiciels parfaits
  - Beaucoup d'options de compilation ne s'activent pas sur des programmes
  - Ou bien s'activent, mais leur effet n'est pas positif
  - Aussi, beaucoup de méthodes automatiques d'optimisation de code ne sont pas incorporés dans les compilateurs
- Aider le compilateur à bien optimiser le code en modifiant le programme source, ou en l'écrivant d'une façon claire pour le compilateur

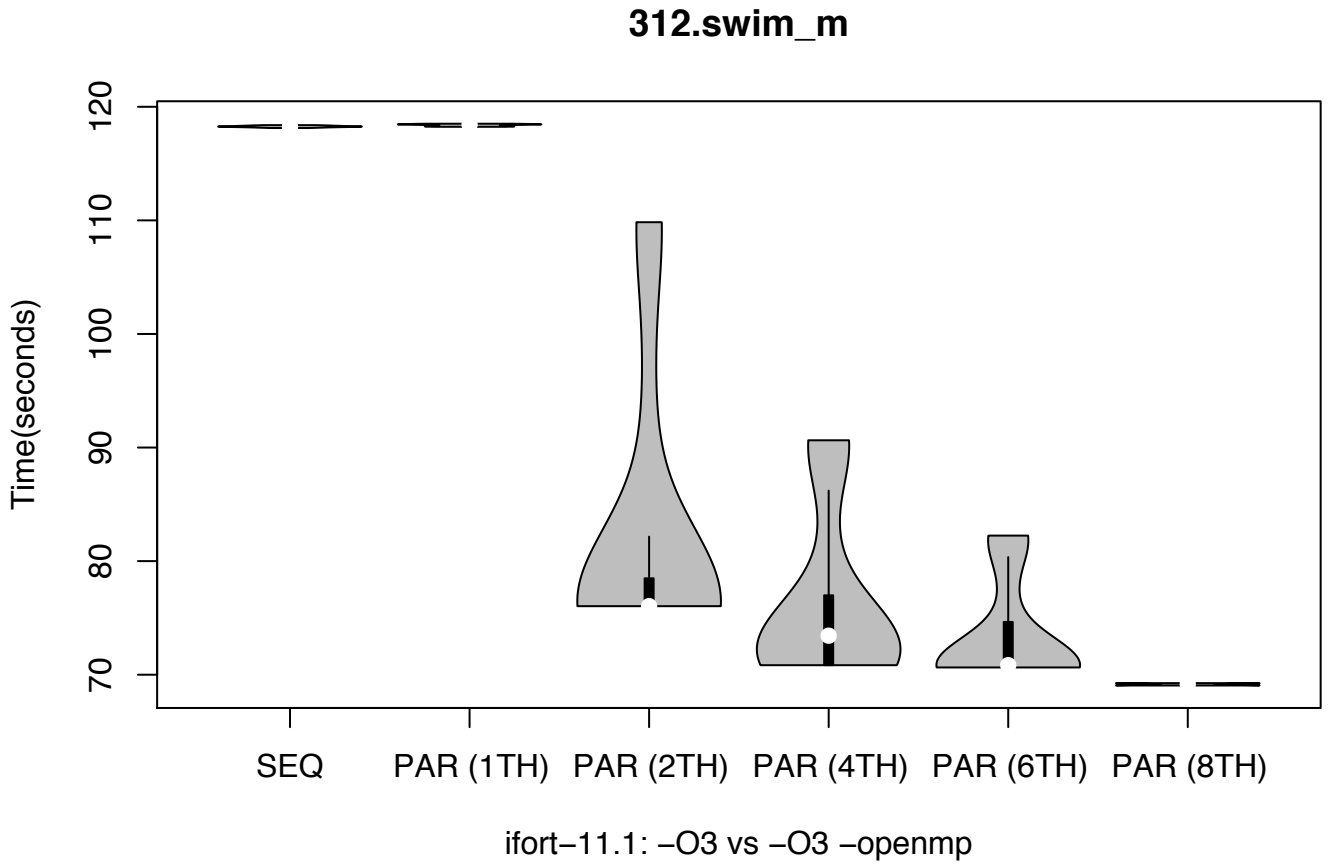
Comment rendre votre programme difficilement optimisable par les compilateurs

- Utilisation massive des pointeurs
  - Alias
  - Calculs arithmétiques de pointeurs
  - Passage de tableaux comme paramètres
- Introduire du code assembleur dans le code source (`asm` en C)
- Pointeurs de fonctions
- Appels à des fonctions externes (de librairie) au sein des boucles

Plan

- HPC vs. Calculs intensifs
- Algorithmique et programmation
- Amélioration des performances logicielles
- Parallélisme
- Compilation et optimisation de code
- Analyse statistique des performances
- Conclusion

Variabilité des performances des programmes



Analyse des performances

- Quelle métrique choisir?
  - Meilleure performance
  - Performance moyenne
  - Performance médiane
  - Pire performance
  - Distribution statistique
- Un algorithme peut avoir *n* codes binaires différents, choisir le meilleur code parmi *n*

Protocole statistique pour décider du code le plus efficace

- Test de student pour comparer les performances moyennes de deux programmes
  - Test de normalité pour les petits échantillons de mesure
- Test de Wilcowon Mann-Whitney pour comparer les performances médianes de deux programmes
  - Test de Komogorov-Smirnov pour les petits échantillons de mesure
- Modélisation statistique par mélange de gaussiennes
  - Métriques de performances moins standards et plus pointues: calculer la probabilité qu'un *seule* exécution soit la plus efficace

Conclusions et perspective

- HPC et calculs intensifs
- Axe de recherche et développement informatique-Physique pour le HPC pour compléter celui des calculs intensifs
- Optimiser des logiciels de calculs
  - Compilation, mesures des performances, réécrire des bouts de programmes, analyse des performances
- Conseils, expertises et collaborations:
  - Sid.Touati@unice.fr