Parallélisation et optimisation dans le contexte de la CFD

Application à des codes ”stencil”

Philippe Parnaudeau[†],
Ingénieur de Recherche - CNRS

[†] Institut Pprime, CNRS, UPR 3346, Pôle formation du MCIA
e-mail: philippe.parnaudeau@cnrs.pprime.fr - Web page: https://www.pprime.fr

La simulation numérique

- 3^{ieme} **pillier de la recherche**
- **Approche unique** envisageable dans certains contextes : **Univers, climat, médecine ...**
- **Approche complémentaire** moins onéreux et/ou plus rapide : **Nucléaire, aérodynamique ...**

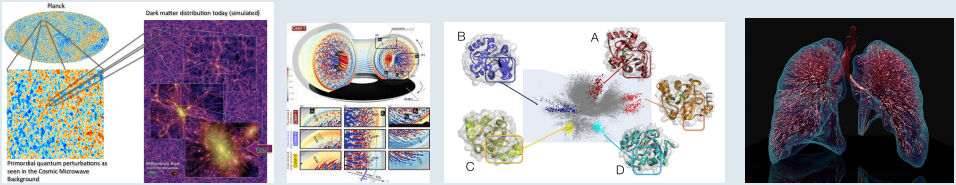
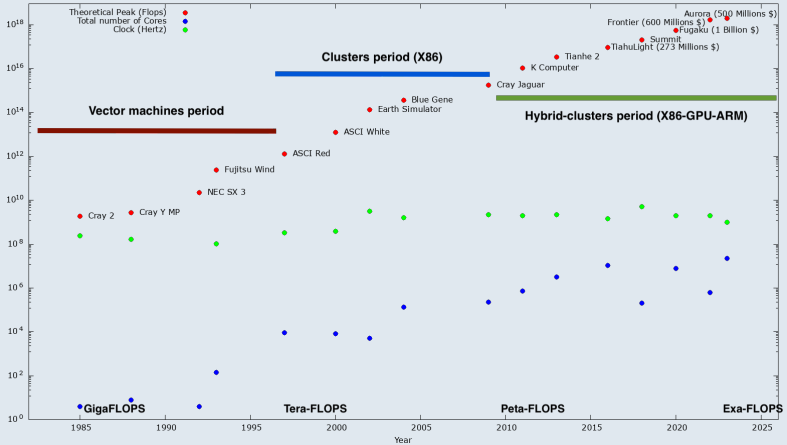


Figure: *Cosmologie-Projet Planck* [**Leclerc˙13**]; *Fusion plasmas ITER* [**Dif-Pradalier˙22**]; *Simulations de dynamique moléculaire COVID 19* [**nsf˙20**]; *Simulation du système pulmonaire* [**asme˙20**].

”Computer science” : De la théorie au transistor

- 1820 — Concept *machine programmable* **C. Babbage** et **A. Lovelace - Lady Byron** [**Lovelace˙1843**]
- 1850 — Algébre de Boole **G. Boole** [**Boole˙1854**]
- 1936 — **A**. Turing [**Turing˙36**] : Machine éponyme et premiers algorithmes modernes
- 1940 — **C**. Shannon [**Shannon˙40**] : Fondateur de la théorie de l’information
- 1947 — **AT&T** Bell laboratory, Lucent, Nokia... : Premier transistor, l’ère du supercalculateur

70 ans de supercalculateurs : Les 35 dernières années



Top supercomputer: loi de Moore

Architectures : Ordinateur

***Ordinateur* parallèle ⇔ Plusieurs CPU dans un ordinateur**
Une tâche s'éxécute sur l'intégralité (ou un sous-ensemble) de la plateforme.

Taxonomie de Flynn : 4 groupes majeurs

- **Single Instruction Single Data: Sequential computer** (Von Neumann)
- **Single Instruction Multiple Data: Vectorial computer**
  Packet of datas (Vector) can be addressed in the same CPU cycle (ex: Cray I and II)
- **Multiple Instruction Single Data: Pipeline computer**
  Successive operations overlap. Example IMB 360/91
- **Multiple Instruction Multiple Data: Multi-CPU computer**
  1 instruction/processor and for different datas
  Often, in recent period, same app. is divided in threads which are executed on CPU cores
  - **MIMD shared memory**: Symmetric Shared Memory (SMP) computer, SMP-Numa computer
  - **MIMD distributed memory**: Massively Parallel Processing (MPP) computer, *modern cluster*

app. : application

Architectures : CPU

- **Instruction Set Architecture (ISA)**
  Modèle ”open” et abstrait décrivant le fct ”logiciel” sur le processeur
  Quelques standards : X86, RISC, ARM ...

- **Micro-architecture ou computer organization ou μarch**
  L'implémentation d'une ISA, pas toujours ”open” (ex : AMD Zen3, Intel Xeon, Apple M2...)

- **Theoretical Peak Performance ($\text{Peak}_{\text{Flops}}$)**
  Nbre Max d'opération en virgule flottante par seconde (**Flop/s**) :

$$\text{Peak}_{\text{Flops}} = \text{Nb}_{\text{cpu}} \text{ x } \text{Nb}_{\text{core}} \text{ x Clock x 2 FMA x } \frac{\text{register}_{\text{size}}}{64}$$

- **Theoretical Memory BandWidth**
  Vitesse maximale d'accès cache (L1, L2, L3) et la mémoire (**Byte/s**)

- **Arithmetic Intensity**
  Opérations en virgule flottante / mémoire accédée (**Flop/Byte**)

- **Performance / Watt**
  Performance d'une app. ou architecture / 1 watt (**F/W**)

FMA: Fused Multiply-Add circuit

Optimisation d'une application*

Déterminer la performance séquentielle :
Performance application (FLOP/S) / $\text{Peak}_{\text{Flops}}$

↪ **Réduction et/ou optimisation** partie séquentielle

Déterminer le passage à l'échelle *scalability* :
Identifier la ou les parties parallèles éfficaces

↪ **Accélération - speedup** ou **gain de la parallélisation**: $S_p = T_1/T_{N_{b_p}}$
↪ **VS Amdahl's law speedup/strong scaling**
CPU bound app. ou **App. long tps exec.** Ideal $S_p = N_{b_p}$
↪ **VS Gustafson's law speedup/weak scaling**
Memory bound app. Ideal $S_p = 1$

↪ **Réduction et/ou optimisation** de la partie communication

*Hypothèse : l'application est déjà parallélisée

Computer Fluid Dynamics ?

Des hypothèses physiques

- Stationnaire ou instationnaire
- Fluide newtonien ou non newtonien
- Ecoulement compressible ou incompressible
- Laminaire/turbulent
- Single/multiphase flow
- Avec ou sans espèces chimiques
- Interaction fluide-Structure

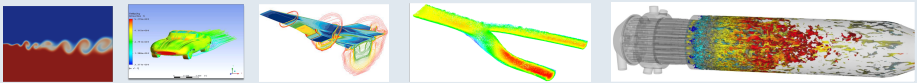


Figure: *Mixing layer*, *Miata (MX-5a)*, *hyper-X vehicle at Mach 7* [**Hyper-X**], *blood flow*, *rocket engine* [**Urbano˙16**]

Computer Fluid Dynamics ?

Les principales étapes

- Géometrie du problème (ex: CAO)
- Maillage du problème
- Hypothèses physiques et modèles mathématiques utilisés
- **Algorithmes et implémentation**
- **Calcul**
- Analyse

Remarques

- Les 2 premières parties sont -en production- les plus chronophages (pas notre sujet)
- **Séminaire: jeu d'Eq. maillage cart. pour des problèmes multiphasique et/ou turbulent.**

A retenir

- Mauvais choix structure des données $\Longrightarrow$ **Ré-écriture du code**
- Performance médiocre (CPU) $\Longrightarrow$ **Optimisation** CPU - sequentiel
- Performance médiocre (CPU) $\Longrightarrow$ **Parallelisation** CPU - mémoire partagée
- Performance médiocre (supercalculateur) $\Longrightarrow$ **Parallelisation** - Optimisation communications
- **Nombreuses librairies HPC** peuvent aider - **Attention au support**
- **Language n'est NI le problème NI la solution** ! $\Longrightarrow$ Le projet dicte le choix du langage!

Optimisation: CPU - Sequentiel et Vectorielle

Limites et résolutions

- **CPU-bound**: App. limitée fréquence du CPU
  Optimisation : CPU-SIMD
- **Cache-bound**: App. limitée taille et/ou fréquence ”cache” des CPU
  Optimisation : Instructions
- **Memory-bound**: Appl. limitée débit de la RAM
  Optimisation : Adapter structure données, améliorer le schémas d’accès à la mémoire
  Optimisation: Math libraries
- **I/O-bound**: App. limitée bande passante des E/S (pas abordé)
  Optimisation: I/O libraries

Conclusion : Profilage et analyse de l’app. / l’architecture

Arithmetic Intensity (A.I) - Intensité Arithmétique

Avoir à l'esprit pour un programmeur

- Les **opérations de base** ont des latences plus ou moins importantes
- **Les codes *stencils* ont, en général, performance médiocre** $\Leftrightarrow$ **CFD codes** $\sim$ 0.3F/B
- **L'optimisation == investissement LOURD** dans le choix des algorithmes et mise en œuvre.

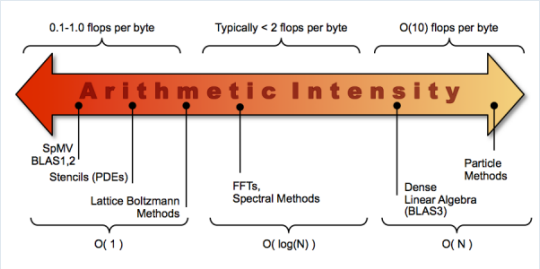


Figure: From Department Of Energy (DOE)

Roofline [**Williams˙09**]

Definition

Evaluer les **performances** de l'app. avec :

Arithmetic Intensity = F(**locality, bandwidth, different parallelization paradigms**).

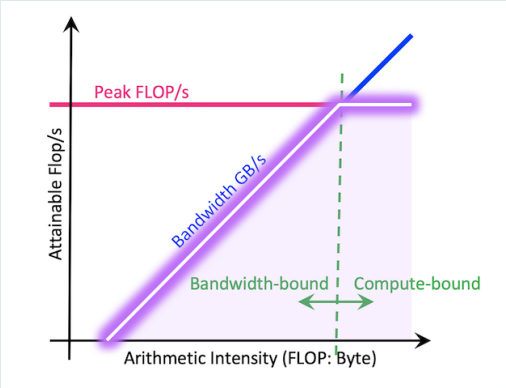
Un modèle "roofline" naif :

$$\text{Peak}_{\max} = \min\left(\text{Peak}_{theo}; \text{A. I. x Mem. Band.}\right)$$

Roofline

- Fournit **Performance de l'app.** vs **Performance la plateforme de calcul**
- **Essentiel pour optimiser la partie séquentielle (CPU) de l'app.**

Roofline [**Williams'09**]



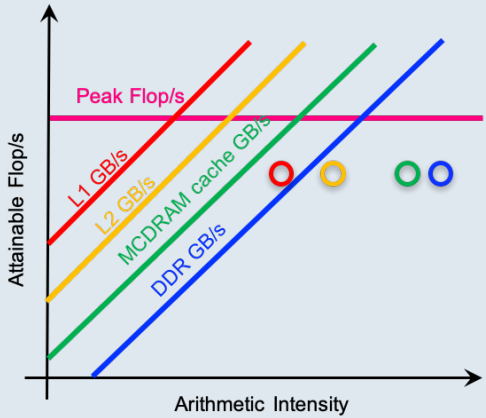


Figure: Roofline: naive view (left), more complex (right). From NERSC

Roofline [**Williams˙09**]

Faire soi-même la "roofline" : Travail **fastidieux et chronophage**, mais **essentiel !**
Conseil : Uniquement pour des courtes partie de l'app.

Quelques outils

- **Intel Advisor**
- Empiral roofline tool
- Roofline Visualizer
- NVIDIA NVProf / NSight (GPU)
- **Papi library** (à la main)

App. maison "CFD" : Performance sur plateforme Intel Xeon 6248

- I.A $\simeq 0.3$ F/B
- $\simeq 7\%$ **Pic performance théorique** (pas si mauvais)
- $\simeq 20\%$ **bande passante théorique de la mémoire** (pas si bon)

Conclusion: Notre app. est limitée par la bande passante mémoire

Parallelisation - ”Shared memory” - mémoire partagée

Definitions

- **Process**: Partie exécutable d’un programme, divisée en un ou plusieurs **thread**
- **Thread**: Plus petite partie d’un processus gérée de manière indépendante par l’O.S
- **Core**: Plus petite partie dédiée au calcul d’un processeur moderne

OpenMP

- API standard de l’industrie pour la programmation parallèle à mémoire partagée
- Basé sur des directives pragma à insérer dans le code
- Intégré langages modernes (C, C++, Fortran, Python-PyOMP); Implémenté compilateurs modernes et sur GPU (directives diff.)
- Alternatives: OpenACC, OpenCL, Pthreads

Exemple : Equation de Poisson

Résoudre "Poisson" est un point essentiel en CFD (incompressible)

Problem

Résoudre : $-\nabla u = f$, in Ω
Avec : $f(x, y) = 2(x(x-1) + y(y-1))$
CL Dirichlet : $u|_{\partial\Omega} = 0$
Solution exacte : $u(x, y) = xy(x-1)(y-1)$
Condition initiale : White noise

Résolution

- Différence finie
- Jacobi

OpenMP : Tactique

Fine Grain (FG) - Grain Fin

- **OpenMP divise la boucle en plusieurs threads**
- **(+++)** : Simple à utiliser (pragma)
- **(+++)** : Facile de maintenir l'app.
- **(—)** : Performance médiocre (nbre de "parallel region")
- **(—)** : Difficile avec code complexe/gros et certain langage. Dépendance dans les boucles

```
1  DO j= 1,ny
2  DO i= 1,nx
3    u_new(i,j)= c0 * ( c1*(u(i+1,j)+u(i-1,j)) &
4                     + c2*(u(i,j+1)+u(i,j-1)) - f(i,j))
5  ENDDO
6  ENDDO
```

Listing: Jacobi-SEQ

```
1  !$OMP PARALLEL DO PRIVATE(i,j) &
2  !$$OMP SHARED (ny,nx,u,u_new,f,c0,c1,c2)
3  DO j= 1,ny
4  DO i= 1,nx
5    u_new(i,j)= c0 * ( c1*(u(i+1,j)+u(i-1,j)) &
6                     + c2*(u(i,j+1)+u(i,j-1)) - f(i,j))
7  ENDDO
8  ENDDO
9   !$OMP END PARALLEL DO
```

Listing: Jacobi-OMPFG

```
1  DO j= 1,ny
2  DO i= 1,nx
3   u(i,j)= omega*(c0*(c1*(u(i+1,j)+u(i-1,j)) &
4                    +c2*(u(i,j+1)+u(i,j-1)) &
5                    -f(i,j)))+ (1.-omega)*u(i,j)
6  ENDDO
7  ENDDO
8  DO j= ny,1,-1
9  DO i= nx,1,-1
10  u(i,j)= omega*(c0*(c1*(u(i+1,j)+u(i-1,j)) &
11                   +c2*(u(i,j+1)+u(i,j-1)) &
12                   -f(i,j)))+ (1.-omega)*u(i,j)
13 ENDDO
14 ENDDO
```

Listing: SSOR-loop nest

OpenMP : Tactique

Coarse Grain (CG) - Grain Grossier

- **Décomposition de domaine**
- **(+++)**: Bonne performance
- **(—)**: Gestion des communications
- **(—)** : Difficile à maintenir

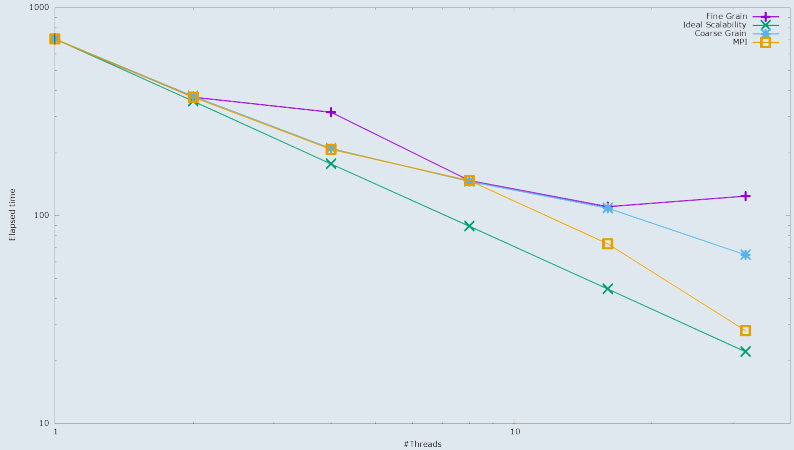
```
1
2   !$OMP PARALLEL PRIVATE (rang,jdeb,jfin)
3       rang=OMP_GET_THREAD_NUM()
4       nbproc=OMP_GET_NUM_THREADS()
5       jdeb=1+(rang*ny)
6       jfin=(ny+rang*ny)
7   !$OMP END PARALLEL
```

Listing: Domain decomposition

```
1   DO j= jdeb,jfin
2   DO i= 1,nx
3       u_new(i,j) =c0*(c1*(u(i+1,j)+u(i-1,j)) &
4                   +c2*(u(i,j+1)+u(i,j-1)) &
5                   -f(i,j))
6   ENDO
7   ENDO
8
9   DO j= jdeb,jfin
10  DO i= 1,nx
11      u(i,j)=u_new(i,j)
12  ENDO
13  ENDO
14  !$omp barrier
15  !$omp flush
```

Listing: Jacobi-OMPCG

OpenMP :
”Strong Scaling Test” - Test Passage à l’échelle ”fort”



OpenMP : Conclusion

• **OpenMP (FG)** : La façon de commencer (rapidement) sur des cas simples : Premiers résultats en quelques heures

• **OpenMP (CG)** : Pour des cas plus complexes et d'excellentes performances

• **OpenMP (TASK)** : Un groupe d'instructions (tâches) est défini et travaille en parallèle. Le nombre de tâches est inconnu à l'avance (non présenté).

Parallelisation - ”Distributed memory” - Mémoire distribuée

Definition

- **Decomposition de domaine** : Théorie décomposition de Scharwz (1870).
Séminaire: limité au cas ”sans recouvrement”
- **Non-blocking point-to-point (P2P)** communication (”Stencil”-code)
- **Non-blocking collective** communication (FFT-code)

Message Passing Interface (MPI)

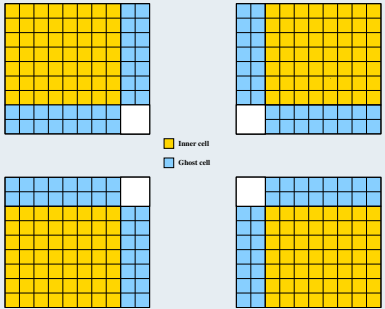
- **Thread-safe** : Threads accédant simultanément à la mémoire
- Définit la syntaxe et la sémantique des routines de la bibliothèque
- 2 implementations ppales (open-source) : **OpenMPI and MPICH2** or MVAPICH2

MPI : Tactique

Non-blocking P2P communication

- **SCB** 5 **à** 9**-point "stencil" /** direction
- Ajout "**ghost points**" à chaque sous-domaine
- **Échange de données** entre voisins

- **(+++)** : Communication à faible coût
- **(+++)** : Performance peuvent être excellente
- **(—)** : Problème implicite



(a) Domain communication

MPI : Tactique

SCB [**Dubois˙21**]: Volume fini

Un des système hyperbolique résolu :

$$\frac{\partial \mathbf{W}}{\partial t} + \nabla \cdot \mathbf{A} + \mathbf{S}\nabla \cdot \mathbf{u} = \mathbf{0}$$

$\mathbf{W} = (\rho, \rho\mathbf{u}, E, \alpha)^\intercal$: Vecteur état
$\mathbf{A} = (\rho\mathbf{u}, \rho\mathbf{u} \otimes \mathbf{u} + P\mathbf{1}, \alpha\mathbf{u})^\intercal$: Vecteur flux
$\mathbf{S} = (0, \mathbf{0}, 0, -(K + \alpha))^\intercal$: Terme source
Sur une grille cartésienne, avec intégration explicite en temps
Les flux sont calculés ”cell-vertex” avec diff. sch.

- HLLC avec ou sans Muscl-Hancock
- WENO-TENO avec ou sans Muscl-Hancock
- JST

MPI : Tactique

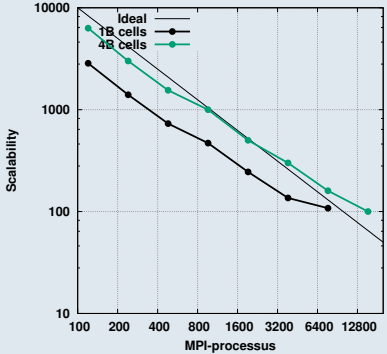
Non-blocking P2P communication - Communication Point à Point non-bloquante

```
1  DO ndt=1,ndtmax
2  !$OMP PARALLEL IF(ijmax.gt.256) default(none)
3  !$OMP DO SCHEDULE (runtime) PRIVATE (i,j,k) COLLAPSE(2)
4     DO k=kmin,kmax
5     DO j=jmin,jmax
6     DO i=imin,imax
7        RI1=w1(i,j,k)-w1(i-1,j,k)
8        sl=dmax(0.0,dmin(Ri1,1.0))+dmin(0,dmax(1,Ri1))
9        W1(i,j,k)= W1(i-1,j,k)+1/4*sl*(W1(i-1,j,k)-W1(i-2,j,k))+1/4*sl*(W1(i,j,k)-W1(i-1,j,k))
10    ENDDO
11    ENDDO
12    ENDDO
13 !$OMP END DO
14    CALL BOUNDARY (W1)
15 !$OMP END PARALLEL
16    CALL MPI_SENDRECV(W1, imax*kmax, MPI_DOUBLE_PRECISION,neib_mpi(N),tag, &
17                      W1, imax*kmax, MPI_DOUBLE_PRECISION,neib_mpi(S),tag, &
18                      comm, status, err_mpi)
19 ENDDO
```

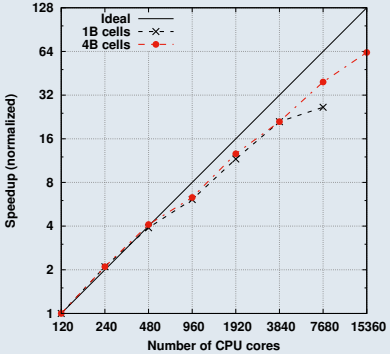
Listing: Hybride MPI-OpenMP implementation

- Line 2: Zone parallel unique : Meilleur performance!
- Line 4-5-6: Bonne vectorisation et optimisation du cache
- Line 16: P2P - W1 échange avec élément N - S
- Line 16: Taille du message

MPI-P2P:
”Strong Scaling Test” - Test Passage à l’échelle ”fort”



Strong-Scaling : Passage à l’échelle



Strong-Scaling : Speedup

MPI: Tactique

Non-blocking collective communication - Communication collective non-bloquante

- **Calcul 1D** d'un op. math. suivant 1 direction
- Transposition via **communication collective**
- **(+++)**: Problème implicite (FFT and co)
- **(—)**: Coût des communications +/- élevé

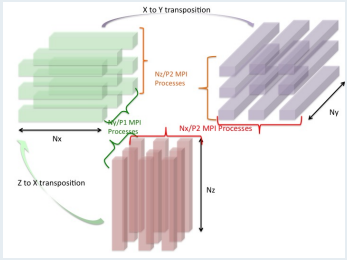


Figure: Pencil MPI communication schemes

MPI: Tactic

GPS [**Parnaudeau˙15**]: FFT code

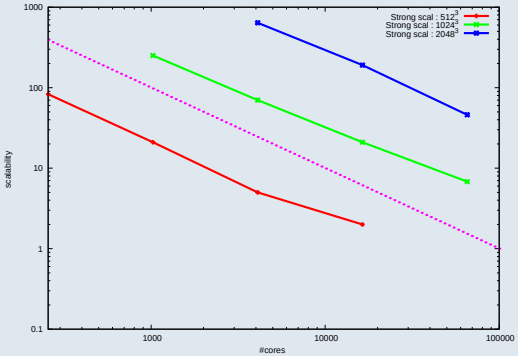
Soit l' **Equation Gross-Pitaevskii (EGP)** adimensionnée avec un **terme de rotation**, pour un cas **stationaire**:

$$\mu\phi(\mathbf{x}) = \Big(-\frac{1}{2}\Delta + \mathbf{V}(\mathbf{x}) + \beta|\phi(\mathbf{x})|^2 - \Omega L_z \Big)\phi(\mathbf{x}) \text{ with} ||\phi||_0^2 = 1$$

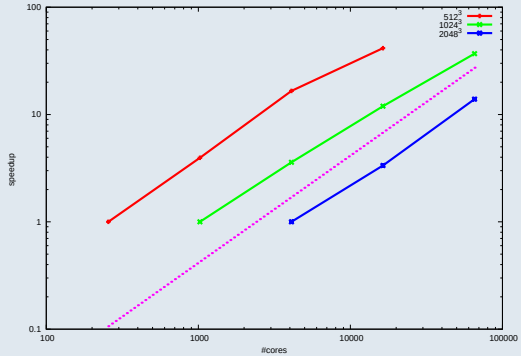
où μ **est le potentiel chimique** du condensat/nuage et

- ϕ: fonction d'onde stationaire
- V: Trappe magnétique, qui peut prendre plusieurs formes etc.
- β: Interaction entre particles dans le nuage
- ΩL_z: Moment angulaire

MPI Non-blocking collective: Strong-Scaling performance test - Test Passage à l'échelle "fort"



Strong-Scaling: Passage à l'échelle



Strong-Scaling: Speedup

MPI: Conclusion

- **P2P communication** : Commencer par **send/recv** et **step/step non bloquant**.
- **P2P communication**: Début simple, mais difficile à optimiser
- **Collective communication**: A n'utiliser que si impératif (opération de réduction et les limiter)
- **Collective communication**: Aujourd'hui, certaines sont asynchrones et cela importe un gain, pas toujours facile à utiliser
- **Recommendation**: Limiter au maximum les communications, utiliser les threads, nouvelles normes apportent des réponses (one side communication)

Conclusion: optimisation et parallelisation

• Nouvelle génération de supercalculateur est hybride (CPU+GPU)

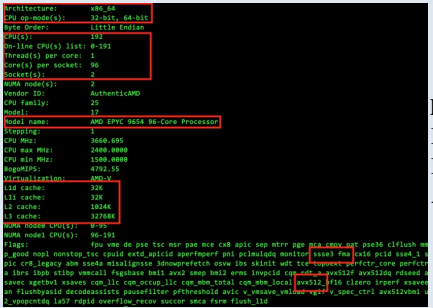
• Architecture X86-64 est et sera de moins en moins archi-dominant (ex: ARM on Apple (Mx), Fujitsu (A64FX), Nvidia (Grace))

• Nécessité d'utiliser au moins 2 or 3 paradigmes

• Toujours avoir à l'esprit maintenabilité et développement futur

• Flops/Watt est le vrai "mur" pour les développeurs !
Attention le Flops/Watt du constructeur n'a rien à voir avec celui de votre app.

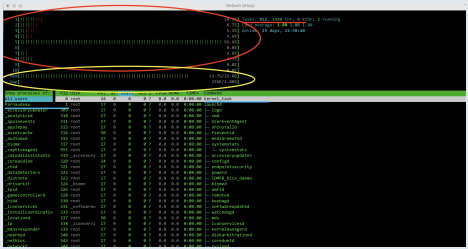
LSCPU:
CPU information



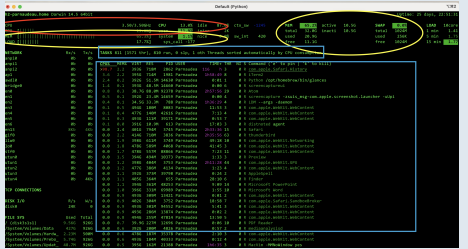
lscpu

Line 1-2-13: Architecture information
Line 4-5-6-7-8: Number of socket, cores and threads per node
Line 9-24-25: Memory policy: Non Uniform Memory Architecture
AVX512: Vectoriel support (SIMD optimisation)

HTOP or GLANCES:
System monitoring core usage, memory usage, process information



htop



glances

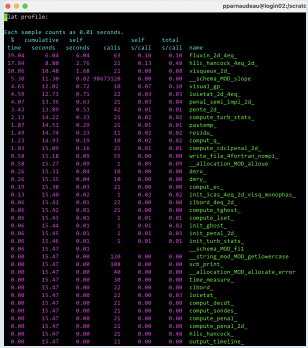
GNU Debuger (GDB)

- **Compile** the program with options: -g
- **Serial/Sequential/OpenMP**: gdb Binary_name
- **Distributed**: MPIRUN_Command_name xterm -e gdb Binary_name (Nb proc. < 10)
- **Some GUI for GDB**

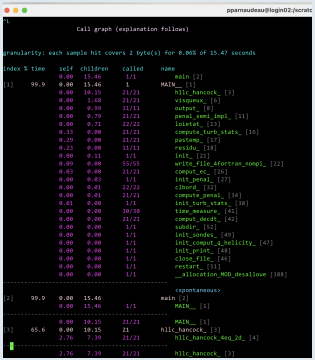
Useful commands

Command	Argument	Explain
b	file:line	Breakpoint in file at line
n	binary	Execute binary
p	variable	Display variable value
n		Execute next instruction
c		Continue the program instruction
quit		Quit gdb

GNU Profiler (GPROF)



Gprof: Flat view



Gprof: Grap view

- **Compile/link** the program with options: -g -pg
- **Execute** the program in standard way
- Execution **generates profiling files in execution directory**
- To obtain profiling report generation: **gprof Binary_name gmon.out.MPI_Rank gprof.out.MPI_Rank**

Other tools

- **Intel offers a wide and attractive range of tools**

- Intel Advisor: Help for design code for efficient vectorization, threading, and offloading to accelerators

- Intel Inspector: Locate and debug threading, memory, and persistent memory errors

- Intel Trace Analyzer and Collector (ITAC): Help for efficient MPI application

- Intel VTune™ Profiler: Analysing and optimizing performance of code for several architecture

- **Cray offers a wide and attractive range of tools**

- Cray Performance and Analysis Tools: Help to design code for efficient vectorization, threading, and offloading to accelerators

Scientific and math libraries

- The Netlib math library
  - **BLAS-1-2-3**: (vector and matrix operations) - Fortran
  - **CBLAS** - C
  - **LAPACK**: Solve linear equation systems
  - **ScaLAPACK**: Distributed version of Lapack
- Intel Library: MKL
  - **Netlib, FFTW ...**
- AMD Optimized CPU Libraries: AOCL
  - **Netlib, FFTW ...**
- NVidia GPU Libraries: CUDA-X
  - **Netlib, FFTW ...**
- I/O Libraries
  - **HDF5, Netcdf, Adios2**

Return to mainpages

Latence de différentes instructions

Operation	coût / cycle CPU	SIMD Optimisation
ADD, OR, SUB, MUL, FMA	2	Excellent
L1-Read	4	Excellent
If, wrong branch	[10;20]	Good
L2-Read	10	Good
DIV, SQRT	[20;40]	Poor
Function callecd (Language and method dependent)	$>$ [30;60]	Poor
L3-Read	[60;70]	Very poor
EXP, LOG, SIN, COS..	$>$100	Very poor
RAM-NUMA-Read	[100;500]	No gain!
Allocation/deallocation	[200;500]	No gain!
Kernel call	$>$ 1000	

Table: Cost of instruction latencies:
from A. Fog, *Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs*

Return to mainpages

hiérarchie de la mémoire

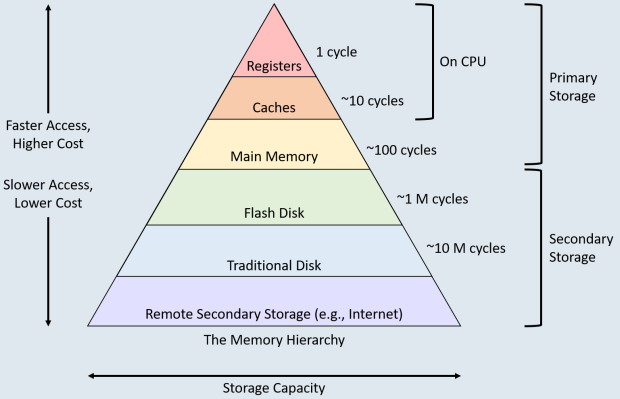


Figure: From Durganshu Mishra's blog

Return to mainpages

Single Instruction Multiple Data and vectorization pipelining (1/2)

- **Les CPUs-HPC possèdent un jeu d'instructions vectorielles**
  ex: Streaming SIMD Extension (SSE) ∈ Advanced Vector eXtensions AVX ∈ AVX512
- **Without SIMD**: Instruction/instruction **vs with**: Grouped instructions (vector) in same CPU cycle
- **Use compilers options** or **using dedicated libraries** or **using pragma approach (not presented)**
- **Recommendation:** ***IDRIS SIMD course***

```
1  DO k=kmin,kmax
2  DO j=jmin,jmax
3  DO i=imin,imax
4     W1(i,j,k)= 0.5*(W1(i+1,j,k)+W1(i,j,k))
5     W2(i,j,k)= sqrt(W1(i,j,k)*W1(i,j,))
6     W2(i,j,k) = 0.5*(W2(i-1,j,k)+W2(i,j,k))
7     if (W2(i,j,k) > Lim) print *,"Value W2:", w2(i,j,k)
8     W3=user_func(W2(i,j,k))
9  ENDDO
10 ENDDO
11 ENDDO
```

Listing: Fortran examples

- Line 4: Vectorisable **mais dans une autre boucle**
- Line 5: Ne peut pas être vectorisée: **Fonction transcendante**
- Line 6: Ne peut pas être vectorisée: **Anti dependence**
- Line 7: Ne peut pas être vectorisée: **Test conditionnel**
- Line 8: Ne peut pas être vectorisée: **Appel de fonction**

Single Instruction Multiple Data and vectorization pipelining (2/2)

Règles simples

- **Toujours**: specifier la taille de la boucle
- **Eviter**: E/S ou fonction appelée ou test conditionnel
- **Eviter**: dépendace dans une boucle
- **Eviter**: les pointeurs
- **Eviter**: les boucles internes trop petites
- **Recommendation**: vérifier les options du compilateur et sa documentation

Exemples avec Gfortran

- *-O3* : Maximum optim. (take care) enabled by default
- *-march=native* (AVX1,AVX2, AVX512...): Leave compiler selection to CPU optimization
- *-fopt-info-vec-all*: Vectorization informations
- All compilers (Intel, Cray ...) have an equivalent options: Read the compiler documentation

Libraries:

HPC libraries (BLAS-1,2,3) dedicated to performing basic vector and matrix operations

Return to mainpages

Optimisation des instructions
Optimisation des cache (1/2)

- Quelle est la différence entre la mémoire cache et la mémoire RAM ? Memory hierarchy latency
- Politique de gestion du cache : **Localité spatiale des données**
- Politique de gestion du cache : **Localité temporelle des données**
- **Avoid cache conflicts**

```
1  DO k=kmin,kmax
2  DO j=jmin,jmax
3  DO i=imin,imax
4  W1(i,j,k)=W0(i,j,k)*W3(i,j,k))
5  W2(i,j,k)=0.5*(W1(i,j,k)*W1(i,j,k))
6  W3(i,j,k)=0.5*(W2(i,j,k)+W2(i+1,j,k))
7  W4(i,j,k)= W4(i,j,k)/W0(i,j,k)
8  ENDDO
9  ENDDO
10 ENDDO
```

Listing: Fortran source

- Taille du tableau $\propto$ taille du cache : **Des conflits de cache apparaissent**
- **Localité spatiale**: Conserver l'ordre des boucles
- **Localité temporelle**: W3 est ré-utilisé

Optimisation des instructions
Optimisation des cache (2/2)

Règles simples

- **Mémoire contiguë**: ordre index des boucles (fct du langage) - localité spatiale
- **Réduction de la latence (localité des données)** : Amélioration des conflits de cache
- **Tolérer la latence (prefetching)** : Optimisation localité (proche CPU)
- Point de vue : Optimisation complexe et architecture dépendant

Gfortran Compiler: optimization options

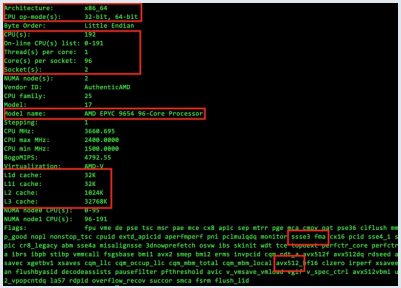
- Compiler optimizations: Prefetching, loop unrolling, cache-aware
- *–fopt-info-note*: Optimization report
- Read the compiler documentation and use the pragma directive optimization carefully

Libraries:

Libraries (Blas, Lapack) dedicated to performing cache optimization

Return to mainpages

CPU Architecture: Theoretical Peak Performance



lscpu on Austral supercomputer node

Return to mainpages

- **1 Austral Node (Criann supercomputer)**
  - Architecture: X86_64
  - 2 sockets with AMD EPYC 9654, 2.4 Ghz (Milan)
  - 96 cores per socket and no hyperthreading
  - L1 cache 32kB, L3 cache 32MB
  - AVX512 units, FMA (Fused Multiply-Add)
- **Single node performance**

  $\text{Peak}_{\text{Flops}}$ = 2 x 96 x 2.4 x 2 x 16 = 14.74 TFlop/s

Amdahl's law - Strong Scaling - Test Passage à l'échelle "fort"

Accélération théorique parallélisation d'une app. pour un problème de taille de cst.

$$S_{p_{th}} = \frac{1}{1 - P_{para} + \frac{P_{para}}{N_{b_p}}},$$

$$\lim_{N_{b_p} \to \infty} S_{p_{th}} = \frac{1}{1 - P_{para}} = \frac{1}{P_{seq}}$$

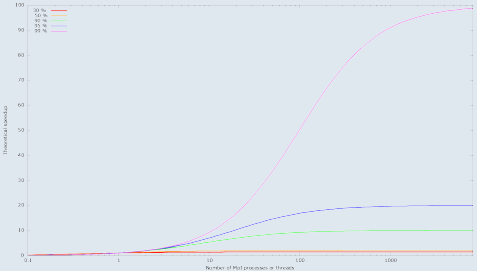


Figure: Amdalh's law, $P_{para} \in [30; 99]\%$

Return to mainpages

Gustafson-Barsis's law - Weak Scaling- Test Passage à l'échelle "faible"

Accélération théorique parallélisation pour une app. pour un pbme où la taille de chaque sous-domaine est fixe.

$$S_{p_{th}} = 1 - P_{para} + (P_{para})N_{b_p} = 1 + (N_{b_p} - 1)P_{para}$$

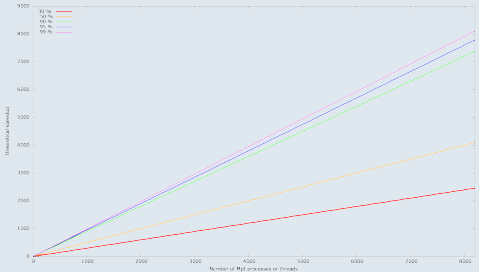


Figure: Gustafson's law

Return to mainpages

References I

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed d
that should have been added to the final page this extra page has been added to receive it.
If you rerun the document (without altering it) this surplus page will go away, because LaTeX no
knows how many pages to expect for this document.