Parallelization and Optimization in the CFD context

An overview of stencil codes

Philippe Parnaudeau[†],
Senior Research Engineer - CNRS

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

Numerical simulation

- **Third pillar of science**
- **Some issues can only be addressed in this way**: universe, climate, medicine ...
- **Some other are complementarily (cheaper or faster) tackled**: nuclear, aerodynamics ...

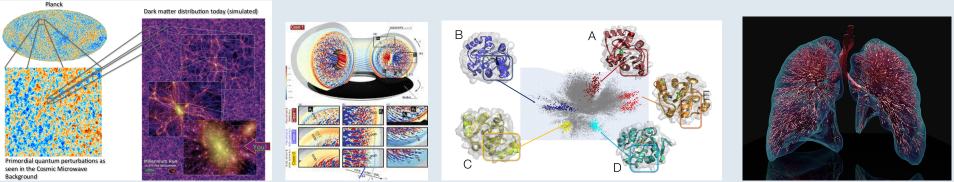
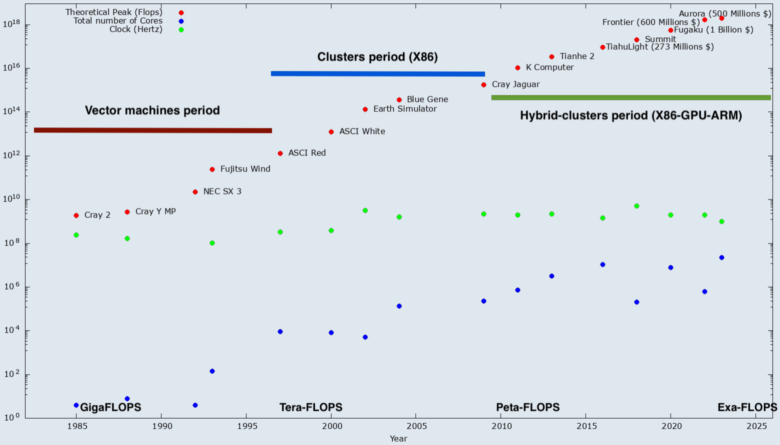


Figure: *Cosmology* [1]; *Fusion plasmas* [2]; *Molecular dynamics simulations* [3]; *Computational Lung Model* [4].

Computer science: From theory to first transistor

- 1820 First concepts of a programmable computer by **C. Babbage** and **A. Lovelace** [5]
- 1850 Boolean algebra by **G. Boole** [6]
- 1936 **A**. Turing [7]: concept of his naming machine and notions of algorithms
- 1940 **C**. Shannon [8]: father of science and information theory
- 1947 AT&T Bell laboratory, Lucent, Nokia...: first transistor - a new area for supercomputing world

70 years of supercomputers: The last 35 years



Top supercomputer: Moore's law

Computer Architectures

Parallel computer ⇔ Multiple-hybride CPU architecture computer

One task simultaneously on the whole (or a subset) part of a computer/cluster/supercomputer

Flynn's taxonomy: evolution in 4 major groups

- **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 application 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

CPU Architectures

- **Instruction Set Architecture (ISA)**

  Abstract model of how a software works on CPU

  The Standards of architecture (ex: X86, RISC, ARM ...)

- **Micro-architecture or computer organization or μarch**

  Implementation of ISA, but not always open (ex: AMD Zen3, Intel Xeon...)

- **Theoretical Peak Performance ($Peak_{Flops}$)**

  Maximum Floating-point operations by second (**Flop/s**):

$$Peak_{Flops} = Nb_{cpu} \text{ x } Nb_{core} \text{ x Clock x 2 FMA x } \frac{register_{size}}{64}$$

- **Theoretical Memory BandWidth (MB_{Bs})**

  Maximum rate at access processor memory (**Byte/s**)

- **Arithmetic Intensity (A.I.)**

  Floating-point operations / bytes in memory accessed (**Flop/Byte**)

- **Performance per Watt (P/W)**

  Application or architecture performance per 1 watt of power (**F/W**)

FMA: Fused Multiply-Add circuit

Optimize application/code*

Determine sequential or serial performance:
Application performance (FLOP/S) / theoretical peak performance

$\hookrightarrow$ **Reducing and/or optimizing** sequential/serial part

Determine scalability:
Identification of parallelization efficiency

$\hookrightarrow$ **Pratical speedup** or **parallelization gain**: $S_p = T_1/T_{N_{b_p}}$
$\hookrightarrow$ **VS Amdahl's law speedup/strong scaling**
CPU bound application or long time run application. **Ideal case** $S_p = N_{b_p}$
$\hookrightarrow$ **VS Gustafson's law speedup/weak scaling**
Memory bound application, ideal case $S_p = 1$

$\hookrightarrow$ **Reducing and/or optimizing** communication part

*Assuming application is already parallelized!

What is CFD ?

Various physical assumptions:

- Stationary or instationary
- Newtonian or non Newtonian fluid
- Compressible or incompressible flow
- Laminar/turbulent
- Single/multiphase flow
- With or without chemical species
- Fluid structure interaction
- **Today's seminar: focus Euler and Navier-Stokes equations on a cartesian mesh (*SCB* [9])**

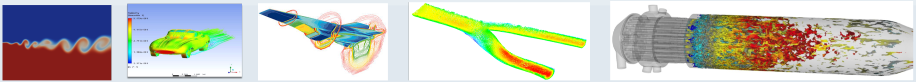


Figure: *Mixing layer*, *Miata (MX-5a)*, *hyper-X vehicle at Mach 7* [10], *blood flow*, *rocket engine* [11]

Optimization: CPU - Sequential and Vectorization

From less to more restrictive application performance

- **CPU-bound**: Application limited by central unit ”frequency”
  Optimization: CPU-SIMD
- **Cache-bound**: Cache size and frequency limits application
  Optimization: Instructions
- **Memory-bound**: Application limited by the speed of the system’s memory (RAM)
  Optimization:
  Adapt data structure, improve memory access patterns
  Optimization: Math libraries
- **I/O-bound**: Application limited by the BW with storage devive (not discussed here)
  Optimization: I/O libraries

Conclusion: Profiling and analysing application vs architecture

Arithmetic Intensity (A.I)

Programmer's perspective

- Some basic operations have a catastrophic latency
- **Stencils-PDE codes perform relatively poorly** $\Leftrightarrow$ **CFD codes** ~ 0.3F/B
- **No free lunch**: Optimization requires a heavy investment of both algorithms and programming

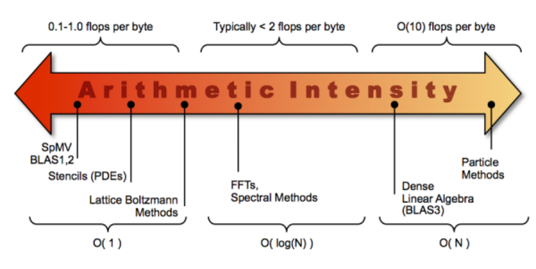


Figure: From Department Of Energy (DOE)

Roofline [12]

Definition

Assessing application's **performances** by :

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

A naive roofline model :

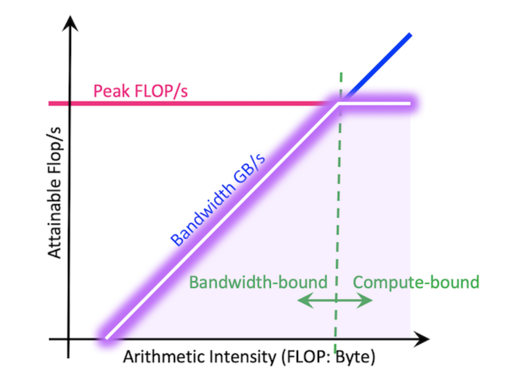
$$P_{\max} = \min\left(\text{Peak; A. I. x MB}\right)$$

Taking into account *memory hierarchy* addressing this naive scheme

Roofline

- Providing **application performance** vs **computer capabilities**
- **Essential for optimising the sequential (CPU) part**

Roofline [12]



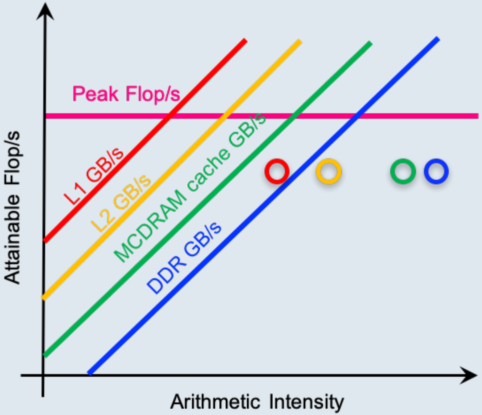


Figure: Roofline: naive view (left), more complex (right). From NERSC

Roofline [12]

Do-it-yourself roofline: A messy and time-consuming job (but often essential!)
Interesting for a short part of code

Some friendly tools

- **Intel Advisor**
- Empiral roofline tool
- Roofline Visualizer
- NVIDIA NVProf / NSight (GPU)
- **Papi library** (by-hand)

In-house CFD code performance on Intel Xeon 6248

- I.A $\simeq$ 0.3 F/B
- $\simeq 7\%$ of the **theoretical peak performance** (not so bad)
- $\simeq 20\%$ of the **theoretical memory bandwidth** (not so good)

Conclusion: Our CFD code is limited by the memory-bound

Parallelization - Shared memory

Definitions

- **Process**: Executable part of a programme, divided into one or more **threads**
- **Thread**: Smallest part of a process managed independently by O.S
- **Core**: The smallest part of the processor in a modern CPU

OpenMP

- Industry standard API for shared memory parallel programming
- Based on pragma directives to be inserted in the code
- Implementation in all modern compilers and now on GPU (but not with same directive)
- Alternatives: OpenACC, OpenCL, Pthreads

Poisson equation

The Poisson equation is a key point for CFD applications for incompressible case

Problem

Solve: $-\nabla u = f$, in Ω
With: $f(x, y) = 2(x(x-1) + y(y-1))$
Dirichlet boundary condition: $u|_{\partial\Omega} = 0$
Exact solution: $u(x, y) = xy(x-1)(y-1)$
Initial condition: White noise

Resolution

- Finite Difference
- Jacobi

OpenMP: Tactic

Fine Grain (FG)

- **OpenMP splits the loop into multiple threads**
- (+++): Simple to use (pragma)
- (+++): Easy to maintain the code
- (—): Low performance (number of parallel region)
- (—): Difficult with complex (depency) loop

```
DO j= 1,ny
DO i= 1,nx
  u_new(i,j)= c0 * ( c1*(u(i+1,j)+u(i−1,j)) &
              + c2*(u(i,j+1)+u(i,j−1)) − f(i,j))
ENDDO
ENDDO
```

Listing: Jacobi-SEQ

```
!$OMP PARALLEL DO PRIVATE(i,j) &
!$$OMP SHARED (ny,nx,u,u_new,f,c0,c1,c2)
DO j= 1,ny
DO i= 1,nx
  u_new(i,j)= c0 * ( c1*(u(i+1,j)+u(i−1,j)) &
              + c2*(u(i,j+1)+u(i,j−1)) − f(i,j))
ENDDO
ENDDO
 !$OMP END PARALLEL DO
```

Listing: Jacobi-OMPFG

```
DO j= 1,ny
DO i= 1,nx
 u(i,j)= omega*(c0*(c1*(u(i+1,j)+u(i-1,j)) &
            +c2*(u(i,j+1)+u(i,j-1)) &
            −f(i,j)))+ (1.−omega)*u(i,j)
ENDDO
ENDDO
DO j= ny,1,−1
DO i= nx,1,−1
 u(i,j)= omega*(c0*(c1*(u(i+1,j)+u(i−1,j)) &
            +c2*(u(i,j+1)+u(i,j−1)) &
            −f(i,j)))+ (1.−omega)*u(i,j)
ENDDO
ENDDO
```

Listing: SSOR-loop nest

OpenMP: Tactic

Coarse Grain (CG)

- **Domain decomposition**
- **(+++)**: Great performance
- (—): Communication management

```

!$OMP PARALLEL PRIVATE (rang,jdeb,jfin)
    rang=OMP_GET_THREAD_NUM()
    nbproc=OMP_GET_NUM_THREADS()
    jdeb=1+(rang*ny)
    jfin=(ny+rang*ny)
!$OMP END PARALLEL
```

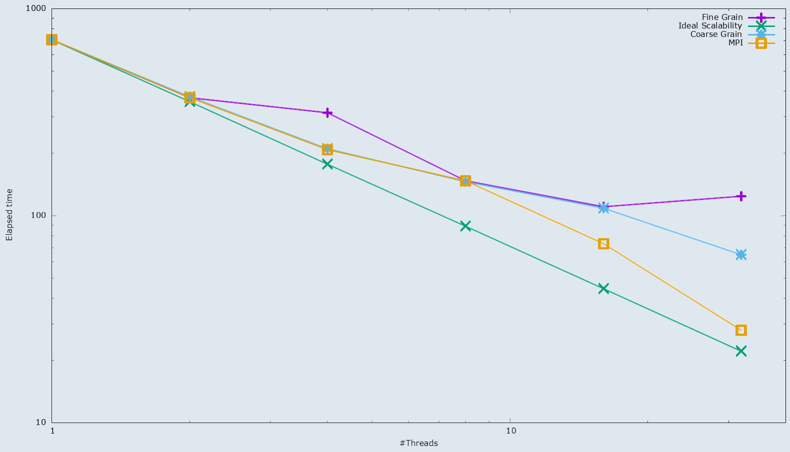
Listing: Domain decomposition

```
DO j= jdeb,jfin
DO i= 1,nx
   u_new(i,j) =c0*(c1*(u(i+1,j)+u(i-1,j)) &
            +c2*(u(i,j+1)+u(i,j-1)) &
            -f(i,j))
ENDO
ENDO

DO j= jdeb,jfin
DO i= 1,nx
   u(i,j)=u_new(i,j)
ENDO
ENDO
!$omp barrier
!$omp flush
```

Listing: Jacobi-OMPCG

OpenMP: Strong-Scaling Performance test



OpenMP: Conclusion

- **OpenMP (FG)**: The way to start (quickly) on simple cases: First results in few hours
- **OpenMP (CG)**: For more complex cases and great performances
- **OpenMP (TASK)**: Group of instructions (tasks) are defined and work in parallel. The number of task is unknown in advance (not presented)

Parallelization - Distributed memory

Definition

- **Decomposition domain** is based on Scharwz (1870) work, for today only non-overlaping approach
- **Non-blocking point-to-point (P2P)** communication (Stencil-code)
- **Non-blocking collective** communication (FFT-code)

Message Passing Interface (MPI)

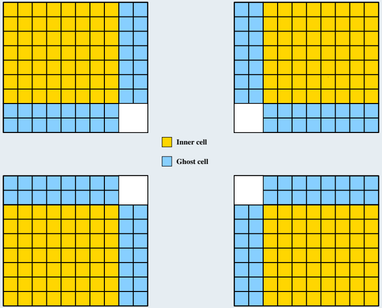
- **Thread-safe**: Threads accessing memory concurrently
- Defines syntax and semantics of library routines
- 2 major open-source MPI implementations: **OpenMPI and MPICH2** or MVAPICH2

MPI: Tactic

Non-blocking P2P communication

- **SCB uses a 5-point-stencil** per direction
- Add "**ghost points**" at each subdomain
- **Exchange data** between neighbors

- **(+++)**: Low cost communication
- **(+++)**: Great performance
- **(—)**: Not easy adapted for implicit problem



(a) Domain communication

MPI: Tactic

SCB [9]: Finite Volume code

Hyperbolic system:

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

$\mathbf{W} = (\rho, \rho\mathbf{u}, E, \alpha)^\intercal$: State vector
$\mathbf{A} = (\rho\mathbf{u}, \rho\mathbf{u} \otimes \mathbf{u} + P\mathbf{1}, \alpha\mathbf{u})^\intercal$: Flux vector
$\mathbf{S} = (0, \mathbf{0}, 0, -(K + \alpha))^\intercal$: Source term
On a cartesian grid, with explicit time integration
Numerical flux are compute at cell-vertex with various schemes

- HLLC with or without Muscl-Hancock
- WENO with or without Muscl-Hancock
- JST

MPI: Tactic

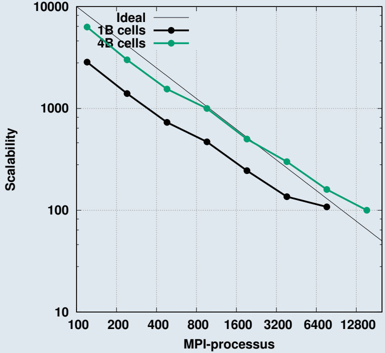
Non-blocking P2P communication

```
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
```

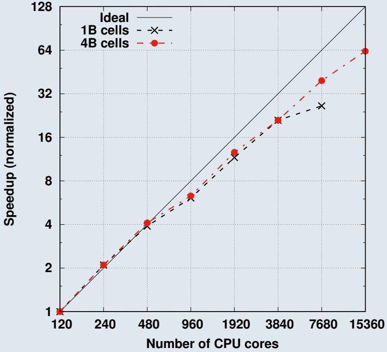
Listing: Hybride MPI-OpenMP implementation

- Line 2: Unique parallel zone declare: Better performance!
- Line 4-5-6: Good vectorization and cache optimization
- Line 16: P2P - W1 exchange between element N and S
- Line 16: Size of message element

MPI-P2P: Strong-Scaling performance test



Strong-Scaling: Scalability



Strong-Scaling: Speedup

MPI: Tactic

Non-blocking collective communication

- **Computation 1D** math. operator along a direction
- Transposition via **collective communication**
- **(+++)**: Implicit problem (FFT's schemes)
- **(—)**: Cost communication

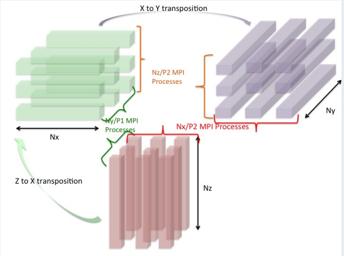


Figure: Pencil MPI communication schemes

MPI: Tactic

GPS [13]: FFT code

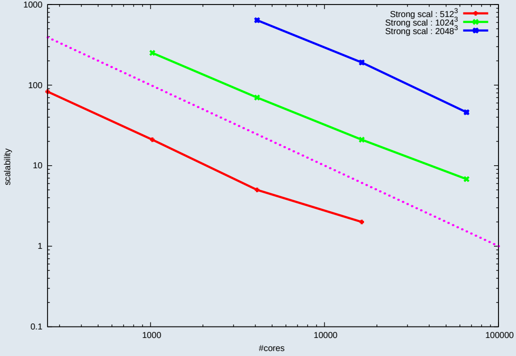
Considering the **dimensionless Gross-Pitaevskii Equation (GPE)** with a **rotation term**, in the case of a **stationary state**:

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

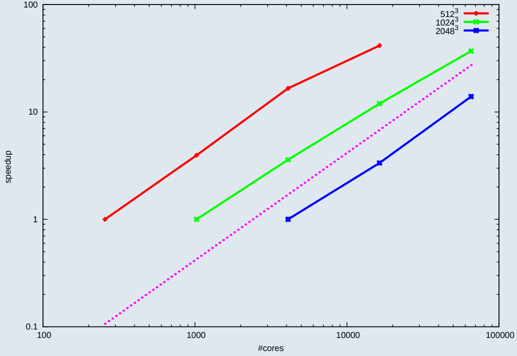
where μ **is called the chemical potential** of the condensate and

- ϕ: Stationary wave function
- V: Magnetic trap, is quadratic, quartic etc.
- β: Interaction between particles inside the condensate
- ΩL_z: Angular momentum

MPI Non-blocking collective: Strong-Scaling performance test



Strong-Scaling: Scalability



Strong-Scaling: Speedup

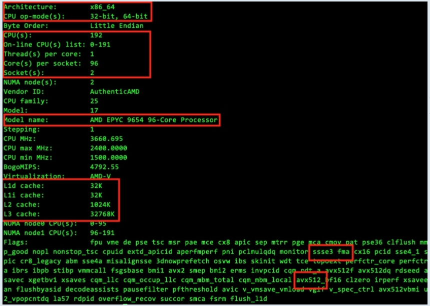
MPI: Conclusion

- **P2P communication**: Start with **send/recv** and **step/step non-blocking**
- **P2P communication**: Easy to start, but challenging to optimize
- **Collective communication**: Only used when needed (reduced operation)
- **Collective communication**: Today, asynchronous implementations are really efficient
- **Basic recommendation**: Limit communication and especially collective

Conclusion: optimization and parallelization

- New generation supercomputers are hybrid (CPU+GPU)
- X86-64 architecture is no longer the archi-dominant (ex: ARM on Apple)
- Need to merge 2 or 3 parallelism paradigms
- Need to think about maintainability and sustainability
- Flops/Watt is a real challenge for developpers!

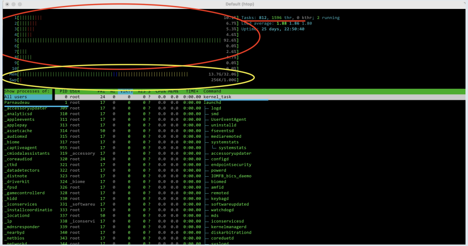
LSCPU:
CPU information



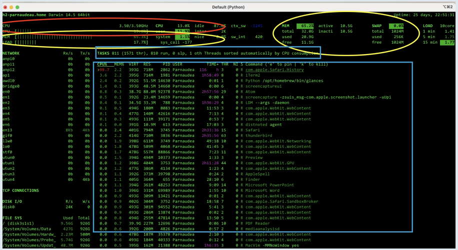
lscpu

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

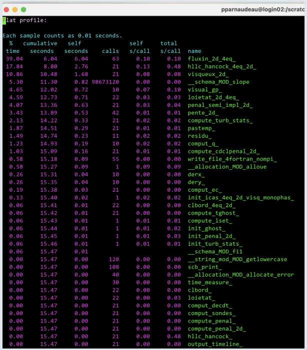
GNU Debuger (GDB)

- **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

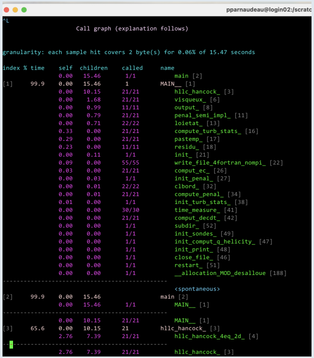
GNU Profiler (GPROF)



```
pparnaudeau@login02:/scratc
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 39.04      6.04     6.04       63     0.10     0.10  fluxin_2d_4eq_
 17.84      8.80     2.76       21     0.13     0.48  hllc_hancock_4eq_2d_
 10.86     10.48     1.68       21     0.08     0.08  visqueux_2d_
  5.30     11.30     0.82 98673120     0.00     0.00  __schema_MOD_slope
  4.65     12.02     0.72       10     0.07     0.10  visual_gp_
  4.59     12.73     0.71       22     0.03     0.03  loietat_2d_4eq_
  4.07     13.36     0.63       21     0.03     0.04  penal_semi_impl_2d_
  3.43     13.89     0.53       42     0.01     0.01  pente_2d_
  2.13     14.22     0.33       21     0.02     0.02  compute_turb_stats_
  1.87     14.51     0.29       21     0.01     0.01  pastemp_
  1.49     14.74     0.23       11     0.02     0.02  residu_
  1.23     14.93     0.19       10     0.02     0.02  comput_q_
  1.03     15.09     0.16       21     0.01     0.01  compute_cdclpenal_2d_
  0.58     15.18     0.09       55     0.00     0.00  write_file_4fortran_nompi_
  0.58     15.27     0.09        1     0.09     0.09  __allocation_MOD_alloue
  0.26     15.31     0.04       10     0.00     0.00  derx_
  0.26     15.35     0.04       10     0.00     0.00  dery_
  0.19     15.38     0.03       21     0.00     0.00  comput_ec_
  0.13     15.40     0.02        1     0.02     0.02  init_icas_4eq_2d_visq_monophas_
  0.06     15.41     0.01       22     0.00     0.00  clbord_4eq_2d_
  0.06     15.42     0.01       21     0.00     0.00  compute_tghost_
  0.06     15.43     0.01        1     0.01     0.01  compute_lset_
  0.06     15.44     0.01        1     0.01     0.02  init_ghost_
  0.06     15.45     0.01        1     0.01     0.03  init_penal_2d_
  0.06     15.46     0.01        1     0.01     0.01  init_turb_stats_
  0.06     15.47     0.01                             __schema_MOD_fi1
  0.00     15.47     0.00      120     0.00     0.00  __string_mod_MOD_getlowercase
  0.00     15.47     0.00      108     0.00     0.00  scb_print_
  0.00     15.47     0.00       40     0.00     0.00  __allocation_MOD_allocate_error
  0.00     15.47     0.00       30     0.00     0.00  time_measure_
  0.00     15.47     0.00       22     0.00     0.00  clbord_
  0.00     15.47     0.00       22     0.00     0.03  loietat_
  0.00     15.47     0.00       21     0.00     0.00  comput_decdt_
  0.00     15.47     0.00       21     0.00     0.00  comput_sondes_
  0.00     15.47     0.00       21     0.00     0.00  compute_penal_
  0.00     15.47     0.00       21     0.00     0.00  compute_penal_2d_
  0.00     15.47     0.00       21     0.00     0.48  hllc_hancock_
  0.00     15.47     0.00       21     0.00     0.00  output_timeline_
```

Gprof: Flat view



```
pparnaudeau@login02:/scratc
^L
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.06% of 15.47 seconds

index % time    self  children    called     name
                0.00   15.46       1/1           main [2]
[1]     99.9    0.00   15.46       1         MAIN__ [1]
                0.00   10.15      21/21          hllc_hancock_ [3]
                0.00    1.68      21/21          visqueux_ [6]
                0.00    0.99      11/11          output_ [8]
                0.00    0.79      21/21          penal_semi_impl_ [11]
                0.00    0.71      22/22          loietat_ [13]
                0.33    0.00      21/21          compute_turb_stats_ [16]
                0.29    0.00      21/21          pastemp_ [17]
                0.23    0.00      11/11          residu_ [18]
                0.00    0.11       1/1           init_ [21]
                0.09    0.00      55/55          write_file_4fortran_nompi_ [22]
                0.03    0.00      21/21          comput_ec_ [26]
                0.00    0.03       1/1           init_penal_ [27]
                0.00    0.01      22/22          clbord_ [32]
                0.00    0.01      21/21          compute_penal_ [34]
                0.01    0.00       1/1           init_turb_stats_ [38]
                0.00    0.00      30/30          time_measure_ [41]
                0.00    0.00      21/21          comput_decdt_ [42]
                0.00    0.00       1/1           subdir_ [52]
                0.00    0.00       1/1           init_sondes_ [49]
                0.00    0.00       1/1           init_comput_q_helicity_ [47]
                0.00    0.00       1/1           init_print_ [48]
                0.00    0.00       1/1           close_file_ [46]
                0.00    0.00       1/1           restart_ [51]
                0.00    0.00       1/1           __allocation_MOD_desalloue [188]
-----------------------------------------------
                                                 <spontaneous>
[2]     99.9    0.00   15.46                 main [2]
                0.00   15.46       1/1           MAIN__ [1]
-----------------------------------------------
                0.00   10.15      21/21          MAIN__ [1]
[3]     65.6    0.00   10.15      21         hllc_hancock_ [3]
                2.76    7.39      21/21          hllc_hancock_4eq_2d_ [4]
-----------------------------------------------
                2.76    7.39      21/21          hllc_hancock_ [3]
```

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

Cost of instruction latencies

Operation	cost / CPU cycle	SIMD Optimization
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

Memory hierarchy

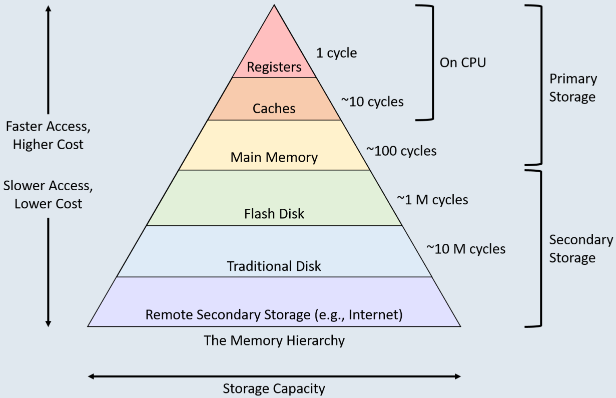


Figure: From Durganshu Mishra's blog

Return to mainpages

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

- **All modern CPUs (since the mid of 90s) have vector instructions**
  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: Can be vectorized **but in another loop**
- Line 5: Cannot be vectorized: **Transcendental function**
- Line 6: Cannot be vectorized: **Anti dependence**
- Line 7: Cannot be vectorized: **Conditional test**
- Line 8: Cannot be vectorized: **Function called**

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

Simple rules:

- **Always**: Specified the size of the loop
- **Avoided**: I/O or called function or conditional test in a computational loop
- **Avoided**: Loop dependence
- **Avoided**: Pointers
- **Avoided**: Too small inner-loop
- **Recommendation**: Check compiler optimization report and documentation

Gfortran compiler optimizations option:

- *-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

Instruction optimization: Memory caching (1/2)

- What is the difference between cache and RAM memory ? Memory hierarchy latency
- Cache Management Policy: **Spatial locality of data**
- Cache Management Policy: **Temporal locality of data**
- **Avoid cache conflicts**

```
DO k=kmin,kmax
DO j=jmin,jmax
DO i=imin,imax
W1(i,j,k)=W0(i,j,k)*W3(i,j,k))
W2(i,j,k)=0.5*(W1(i,j,k)*W1(i,j,k))
W3(i,j,k)=0.5*(W2(i,j,k)+W2(i+1,j,k))
W4(i,j,k)= W4(i,j,k)/W0(i,j,k)
ENDDO
ENDDO
ENDDO
```

Listing: Fortran source

- Array sizes $\propto$ to cache size: **Cache conflicts appears**
- **Spatial locality**: Do not change loop order
- **Temporal locality**: W3 is re-used

Instruction optimization: Memory caching (2/2)

Simple rules

- **Contiguous memory**: Ordering in loop index (langage dependent) - spatial locality
- **Reducing latency (data locality)**: Improving cache misses reoder iteration loop
- **Tolerate latency (prefetching)**: Optimizing data locally (closed to CPU)
- Point of view: Perhaps more complex optimization and architecture dependent

Gfortran Compiler: optimization options

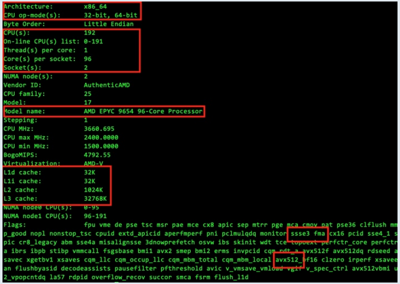
- Compiler optimizations: Prefetching, loop unrolling, cache-aware
- *–fopt-info-note*: Optimization report
- Read the compiler documentation and use the pragma directive optimization carefully

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
- **Single node performance**

  $Peak_{Flops}$ = 2 x 96 x 2.4 x 2 x 16 = 14.74 TFlop/s

Amdahl's law - Strong Scaling

Predicts a theoretical speed-up obtained by parallelizing an application for a cst size problem.

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

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

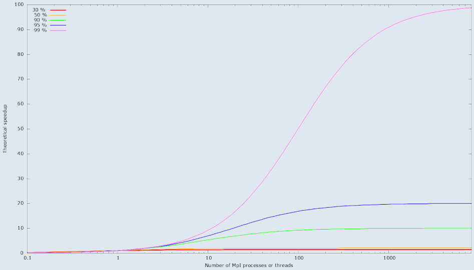


Figure: Amdalh's law, $P_{para} \in [30; 99]\%$

Return to mainpages

Gustafson-Barsis's law - Weak Scaling

Predicts a theoretical speed-up obtained by parallelizing an application where the size of each subdomain is fix.

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

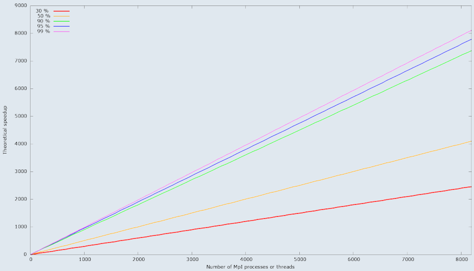


Figure: Gustafson's law

Return to mainpages

References I

[1] F. Leclercq, A. Pisani, and B.D. Wandelt. "Cosmology: from theory to data, from data to theory". In: https://arxiv.org/pdf/1403.1260 (2013).

[2] G. Dif-Pradalier et al. "Transport barrier onset and edge turbulence shortfall in fusion plasmas". In: Communications Physics 229-5 (2022).

[3] Argonne National Laboratory. AI accelerating drug discovery to fight COVID-19. 2020.

[4] John Kosowatz. Computational Lung Model May Guide How Ventilators are Used. 2020.

[5] A.A. Lovelace. "Notes by A.A.L. [August Ada Lovelace]". In: Taylor's Scientific Memoirs p 666-731 (1843).

[6] G. Boole. "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities". In: Macmillan vol. 45 (1854).

[7] A. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: Proceedings of the London Mathematical Society 45 (1936).

References II

[8] C. Shannon. “A Symbolic Analysis of Relay and Switching Circuits”. PhD thesis. Massachusetts Institute of Technology, Dept. of Electrical Engineering, 1940.

[9] R. Dubois, E. Goncalves, and P. Parnaudeau. “High performance computing of stiff bubble collapse on CPU-GPU heterogeneous platform”. In: Comput. Math. Appl. 99 (2021), pp. 246–256.

[10] NASA Identifier: NIX-ED97-43968-1. Hyper-X at mach 7. 2009.

[11] A. Urbano et al. “Exploration of combustion instability triggering using Large Eddy Simulation of a multiple injector liquid rocket engine”. In: Combustion and Flame 169 (2016).

[12] S. Williams, A. Waterman, and D. Patterson. “Roofline: an insightful visual performance model for multicore architectures”. In: Communications of the ACM (2009).

[13] P. Parnaudeau, J.M. SacEpee, and A. Suzuki. An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic ISC-15, Frankfurt, Poster session. 2015.