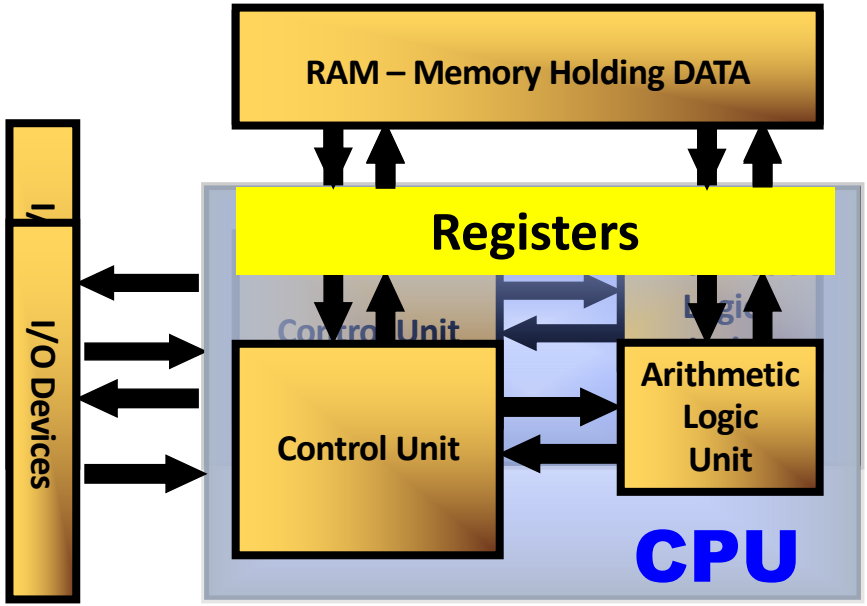
Computing 101

Ivan Girotto (ICTP)
igirotto@ictp.it

The Basic Model

John Von Neumann

```
int main( ){
  int c = 0;
  c = 2 + 3;
}
                                main.c
```




```
$gcc main.c -o main.x
```

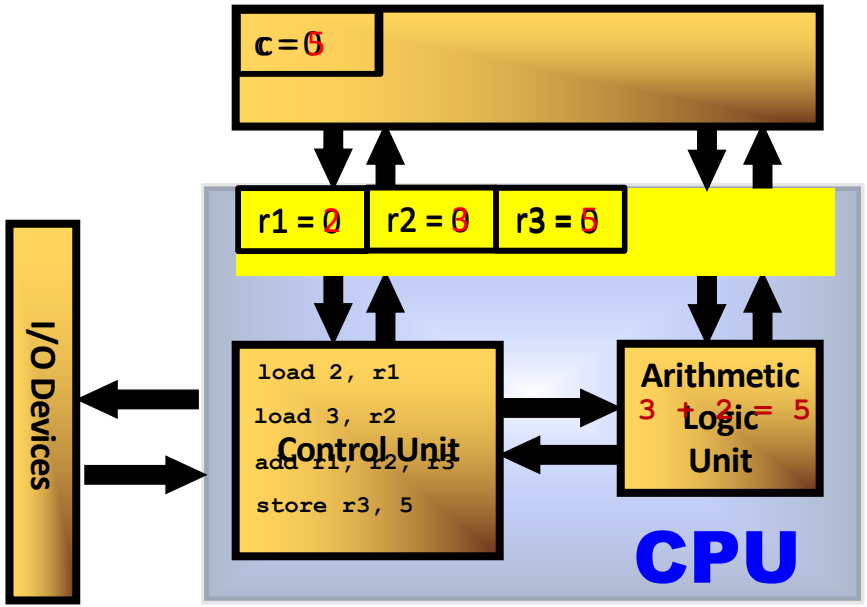
```
$./main.x
```

```
load 2, r1
load 3, r2
add r1, r2, r3
store r3, c
```

The Basic Model

John Von Neumann

```
int main( ){
  int c = 0;
  c = 2 + 3;
}
                          main.c
```




```
$gcc main.c -o main.x
```

```
$./main.x
```

```
load 2, r1
load 3, r2
add r1, r2, r3
store r3, c
```

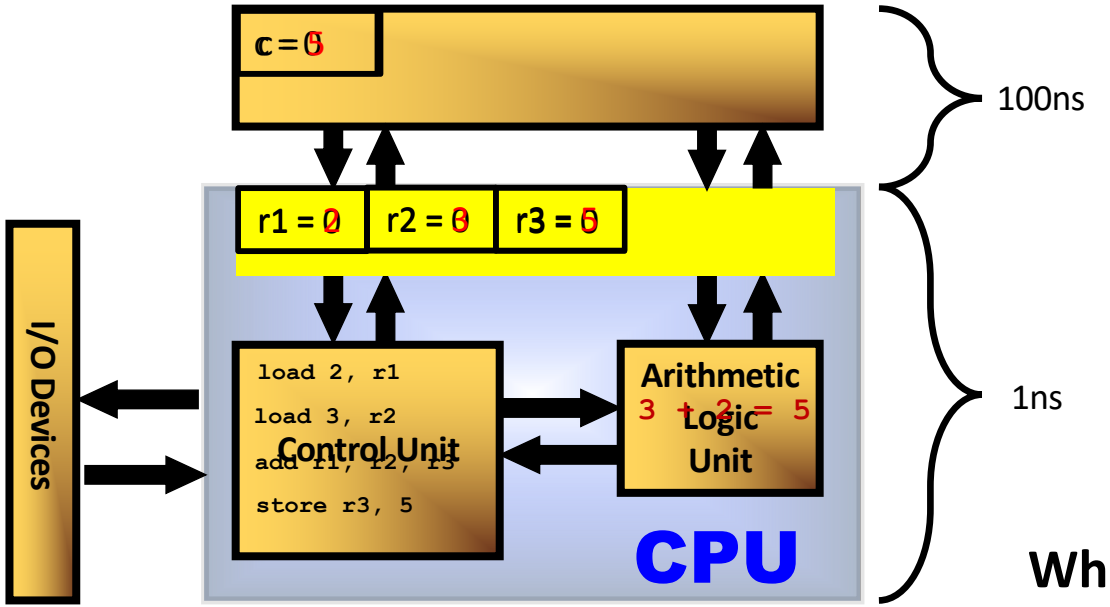
Performance assesment

- Let’s suppose memory data transfer is for free ...
- ... and that our processor runs at $1x10^9$ hertz ($1GH_z$) => x1 cycle every ns
- ... and that for every clock cycle a single operation is performed
- ... to execute 4 operations, x4 ns (10^{-9}) are required!
- Meaning that at full spead, our processor runs our code executing 10^9 operations x second (100% efficiency)
- If operations are considered double precision FP (64bits), normally this number is referred as FLOP/s
- Unluckly this is not the case...

The Basic Model

John Von Neumann

```
int main( ){
  int c = 0;
  c = 2 + 3;
}
                                main.c
```




```
$gcc main.c -o main.x
```

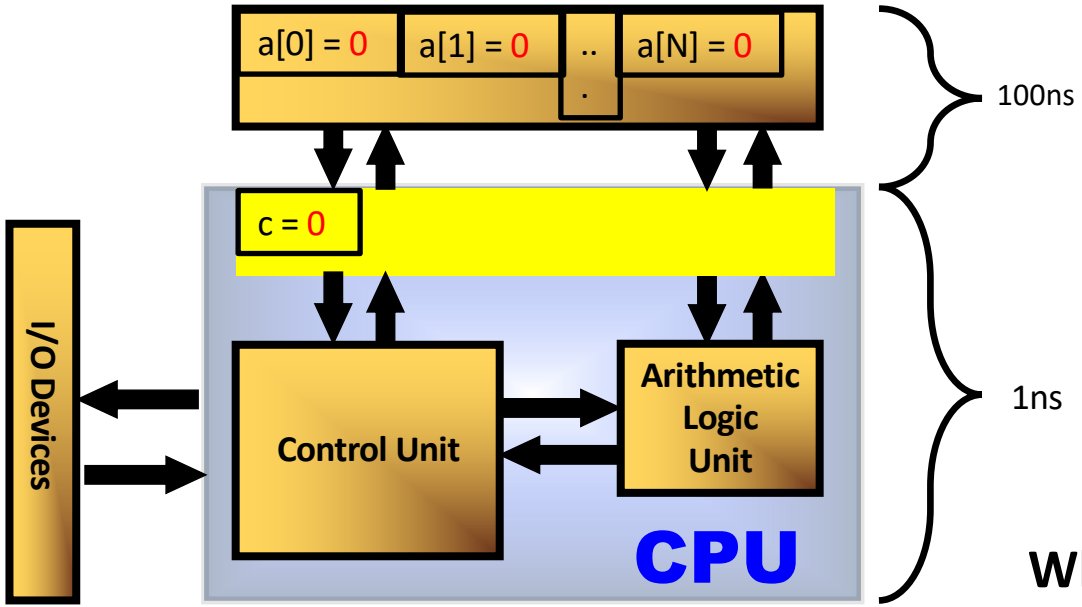
```
$./main.x
```

```
load 2, r1
load 3, r2
add r1, r2, r3
store r3, c
```

What is the number of OP/s here?

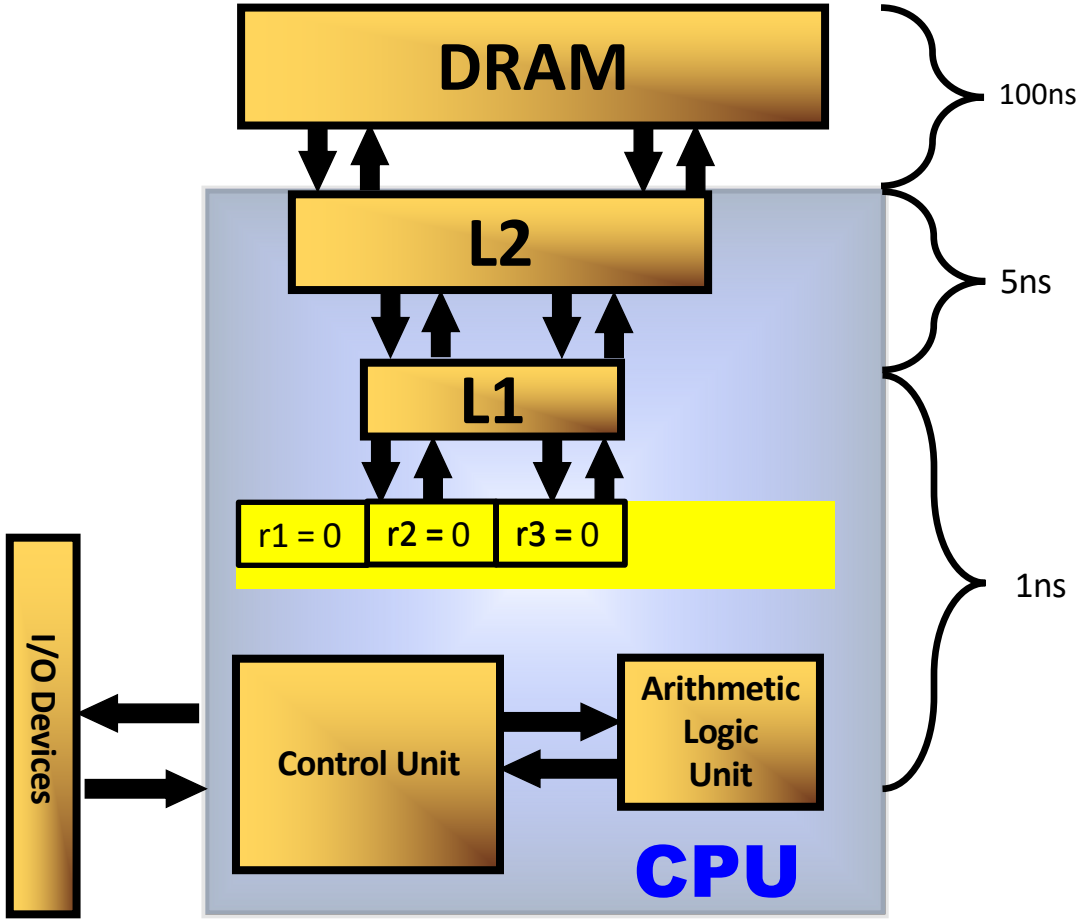
The Basic Model

```
int main( ){
  int a[N];
  for ( c = 0; c < N; c++ )
        a[c] = c;
}
```



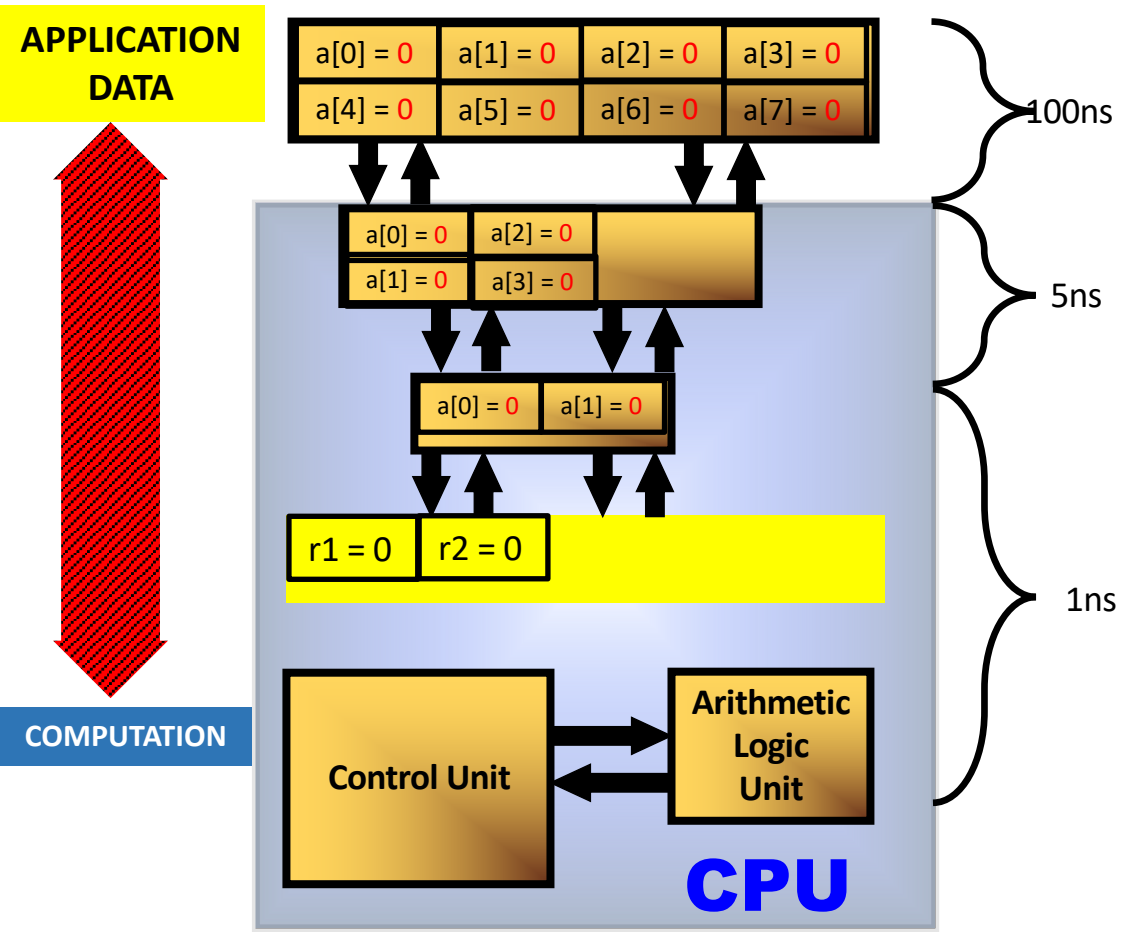
What is the number of OP/s here?

The Memory Hierarchy




```
int main( ){
  int a[N];
  for ( c = 0; c < N; c++ )
        a[c] = c;
}
```

The Memory Hierarchy



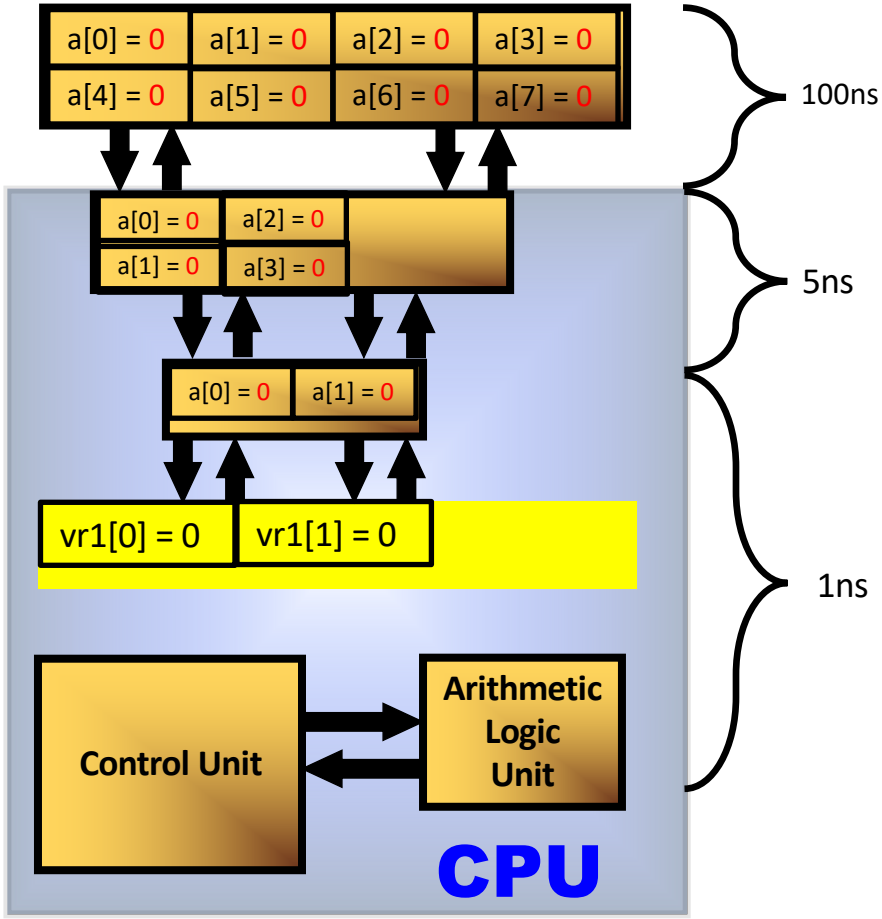

```
int main( ){
  int a[N];
  for ( c = 0; c < N; c++ )
        a[c] = c;
}
```

```
        set r2, 0
Loop: load r1, A(r2)
        store A(i), r1
        add r2, 1
        branch => loop
```

- D
- D

  blocks of fixed size, called *cache lines*.
- Operation of LOAD/STORE can lead at two different scenario:
  - *cache hit*
  - *cache miss*

Vectorization




```
int main( ){
  int a[N];
  for ( c = 0; c < N; c++ )
        a[c] = c;
}
```

Registers are capable to store more then one primitive type at a time

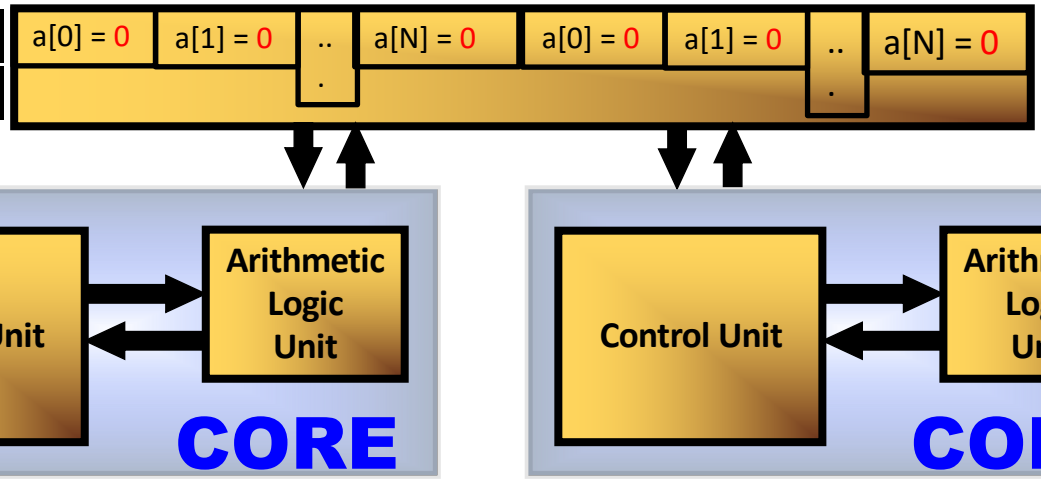
The vectorial ALU is capable to perform the same operation over multiple data of the same kind

Scalar Mode

Wrap up on modern computers

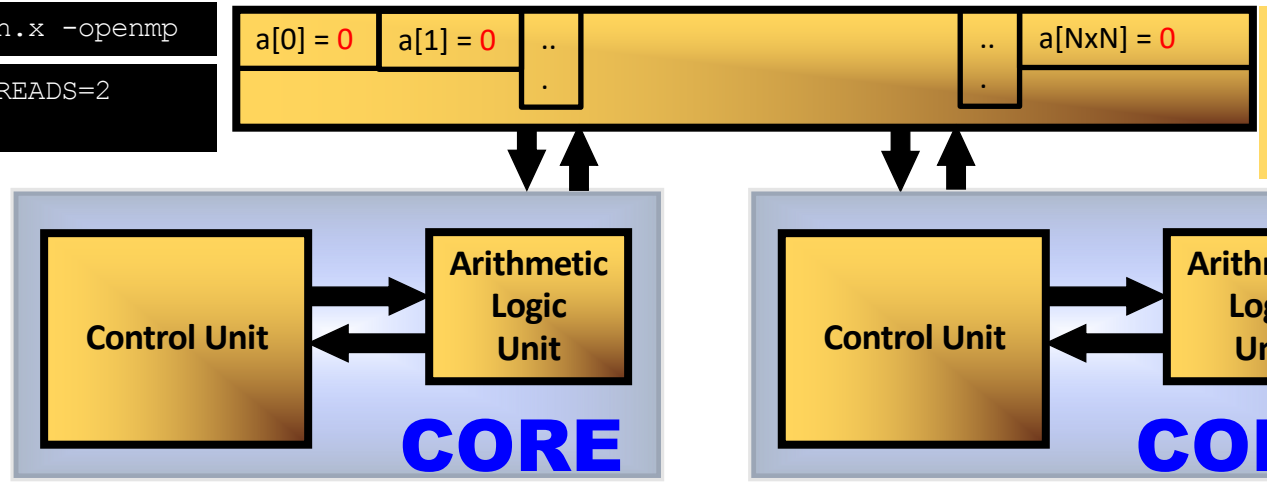
- Efficient data memory access patterns are crucial to best performances
  - design and development of computer algorithms that maximize data locality in time and space (data distance)
- Modern CPUs are equipped with vector registers and ALUs capable of multiple concurrent operations on multiple data
  - vectorization is aided by compilers, but requiring design and development of computer codes that maximize simple operations on contiguous data of the same type
- When all CPU component work at maximum speed that is called *peak of performance* (Floating point operations per seconds FLOP/s)
  - Tech-spec normally describe the theoretical peak
  - Benchmarks measure the real peak
  - Applications show the real performance value
- Real performance are also mostly related to the memory bandwidth (GBytes/s)

```
$gcc main.c -o main.x
$./main.x & ./main.x
```




```
int main( ){
   int a[N];
   for ( c = 0; c < N; c++
      ) a[c] = c;
}
```

```
$gcc main.c -o main.x -openmp
$export OMP_NUM_THREADS=2
$./main.x
```

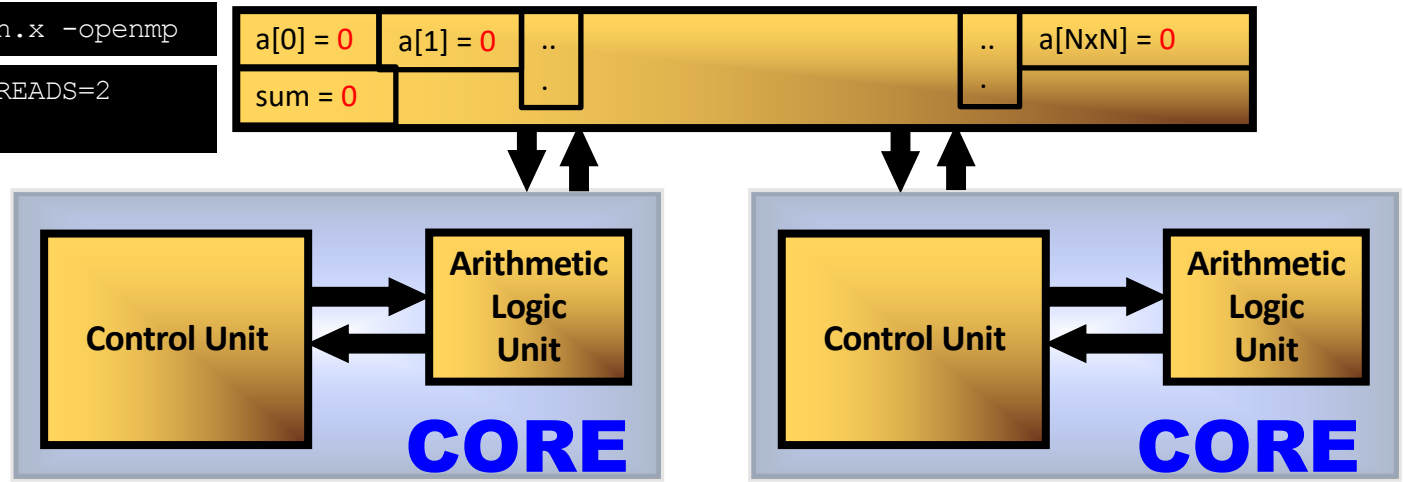



```
int main( ){
   int a[N];
   #pragma omp parallel for
   for ( c = 0; c < N; c++
      ) a[c] = c;
}
```

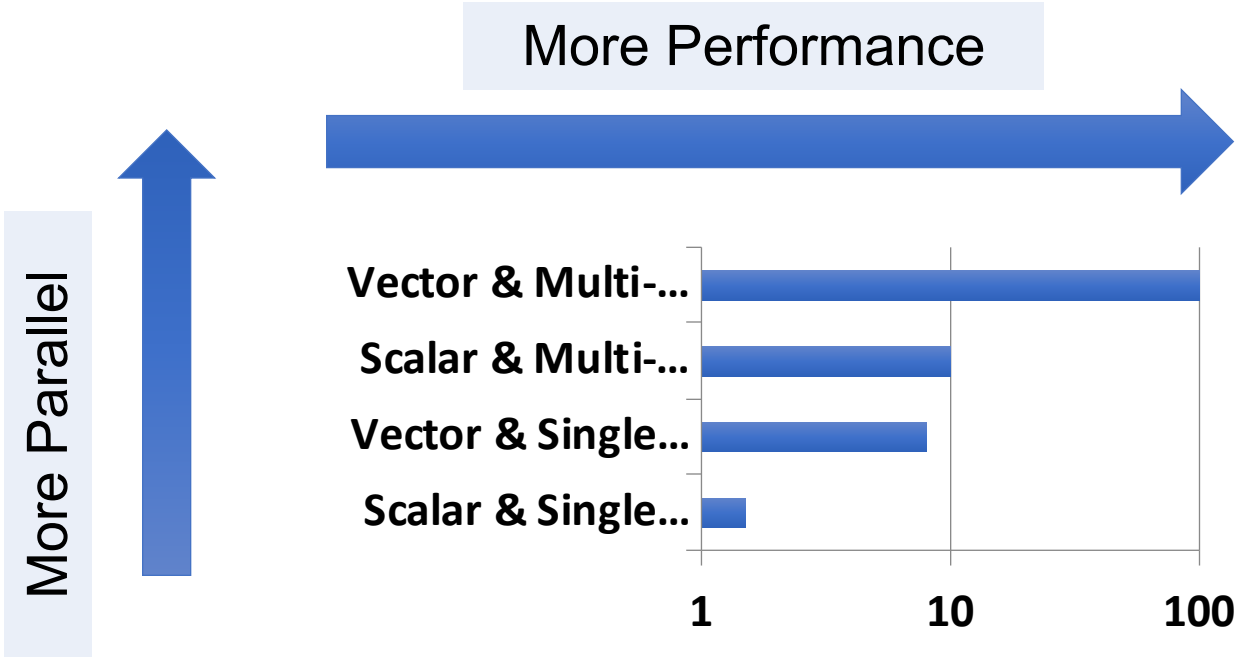
```
int main( ){
 int a[N], sum;
 #pragma omp parallel for
 for ( c = 0; c < N; c++ )
      sum += a[c];
}
```

```
$gcc main.c -o main.x -openmp
```

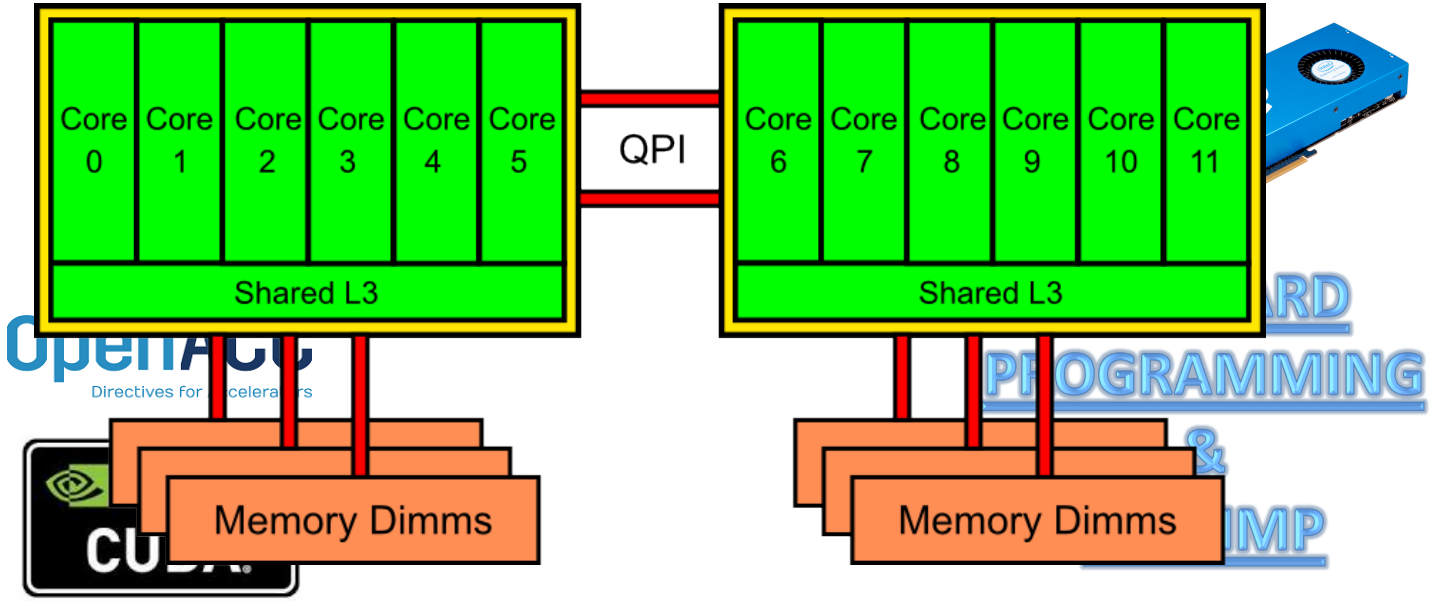
```
$export OMP_NUM_THREADS=2
$./main.x
```



Threading and Vectorization

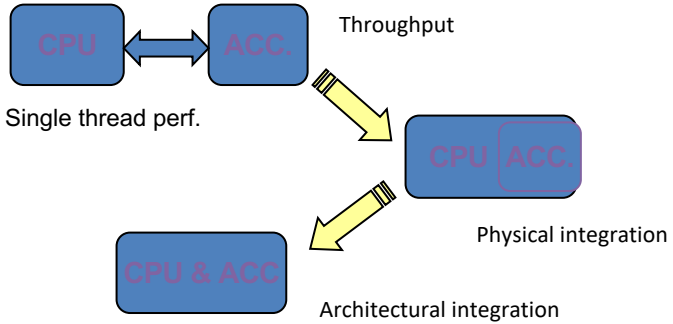


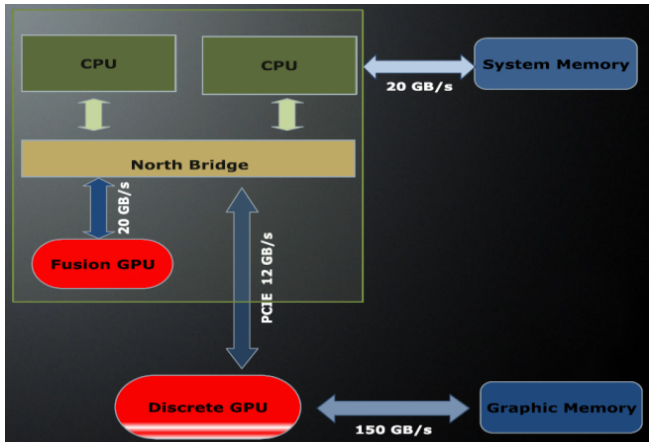
Multiple Socket CPUs + Acceleretors



Accelerated co-Processors

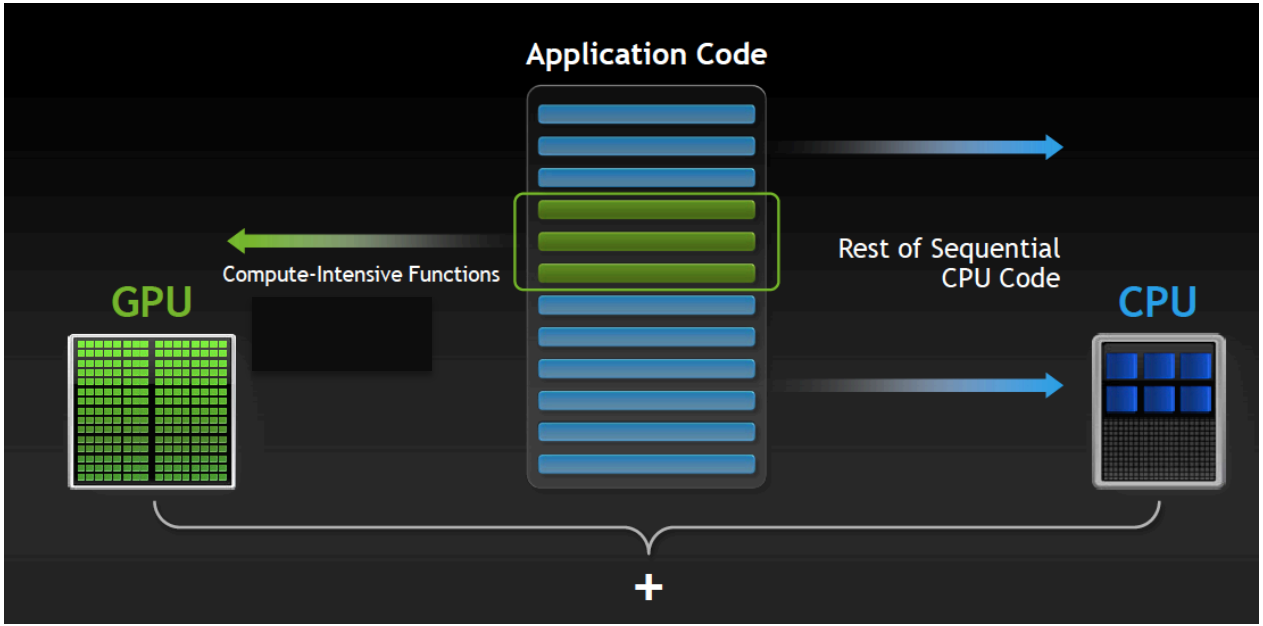
- A set of simplified execution units that can perform few operations (with respect to standard CPU) with very high efficiency. When combined with full featured CPU can accelerate the “nominal” speed of a system.

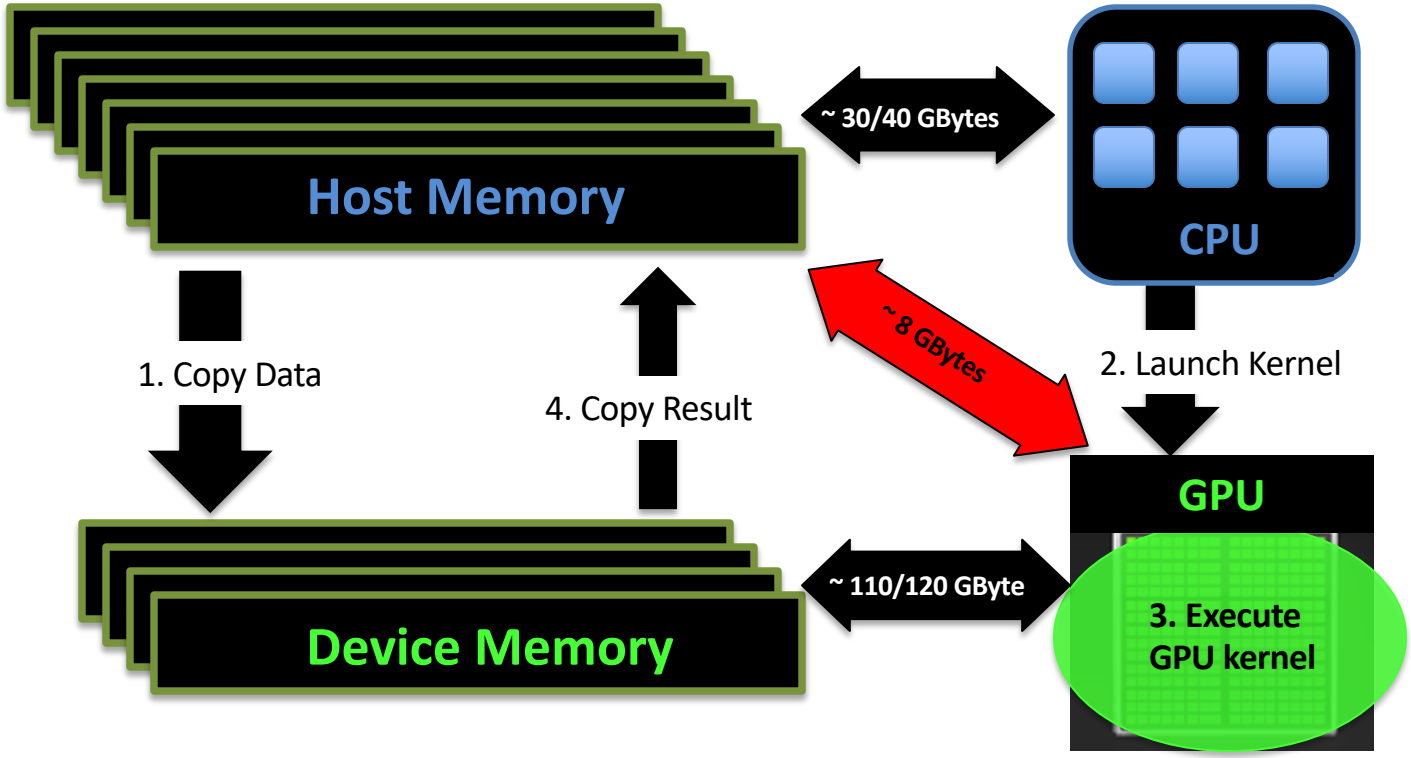




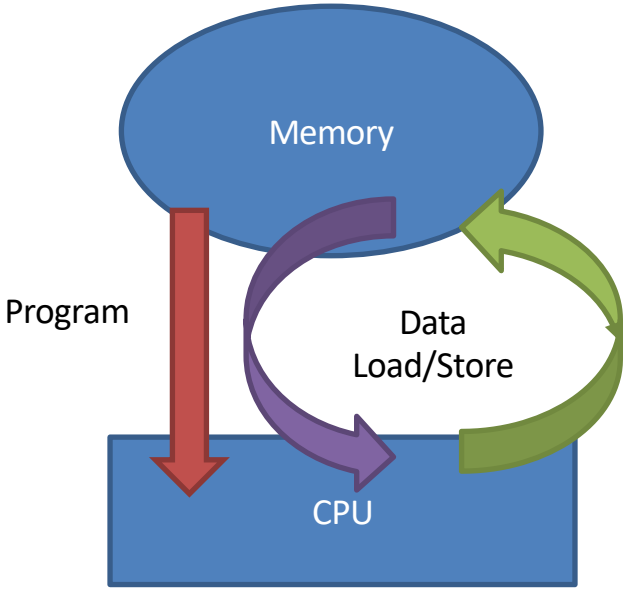
- Main approaches to accelerators:
  - Task Parallelism (MIMD) → MIC
  - Data Parallelism (SIMD) → GPU

The General Concept of Accelerated Computing

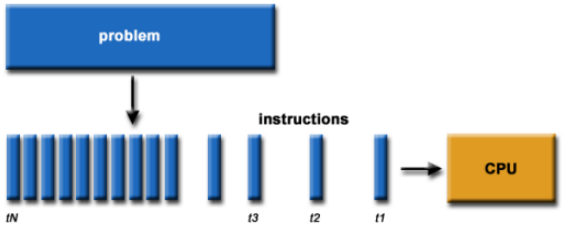



Host Memory
~ 30/40 GBytes
CPU
1. Copy Data
4. Copy Result
~ 8 GBytes
2. Launch Kernel
GPU
Device Memory
~ 110/120 GByte
3. Execute GPU kernel

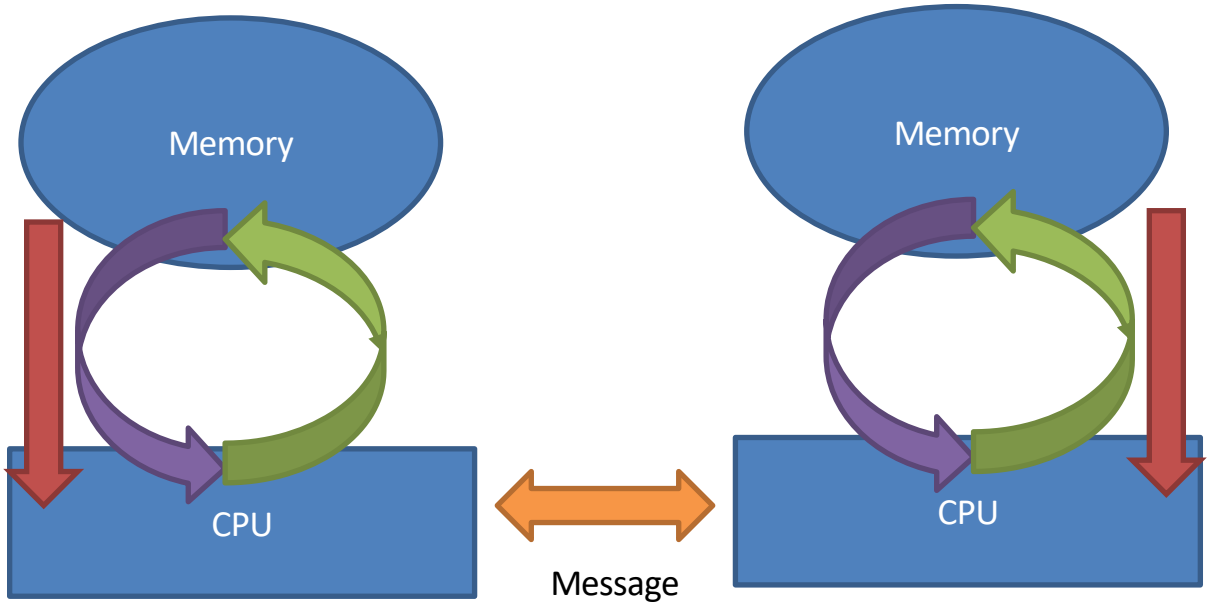
Serial Programming



A problem is broken into a discrete series of instructions.
Instructions are executed one after another.
Only one instruction may execute at any moment in time.

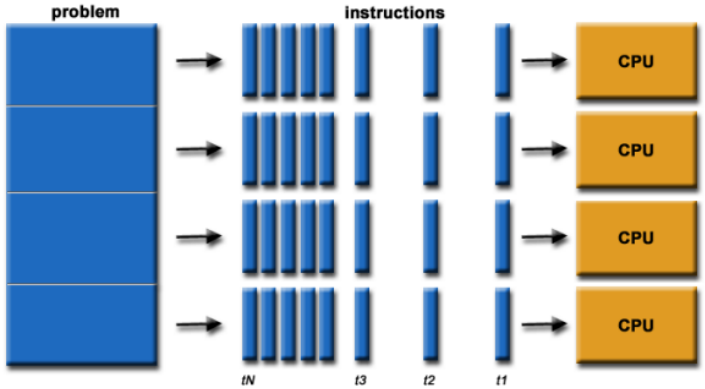


Parallel Programming



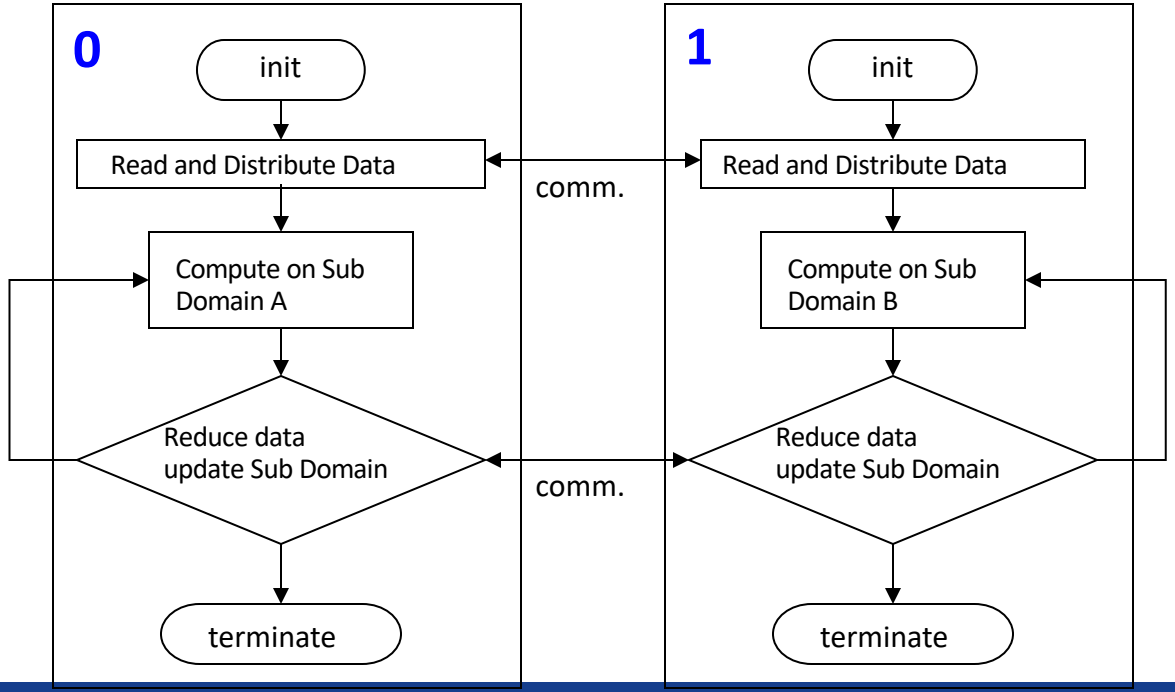
Concurrency

The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently



- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control / coordination mechanism is employed

What is a Parallel Program

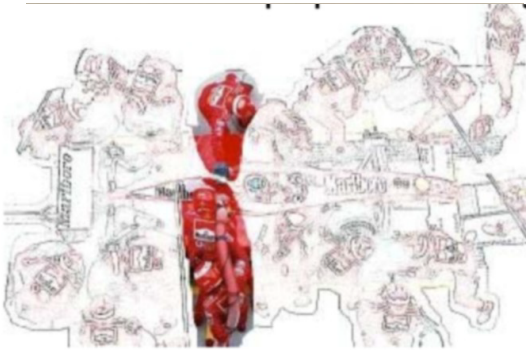


Fundamental Steps of Parallel Design

- Identify portions of the work that can be performed concurrently
- Mapping the concurrent pieces of work onto multiple processes running in parallel
- Distributing the input, output and intermediate data associated within the program
- Managing accesses to data shared by multiple processors
- Synchronizing the processors at various stages of the parallel program execution

Type of Parallelism

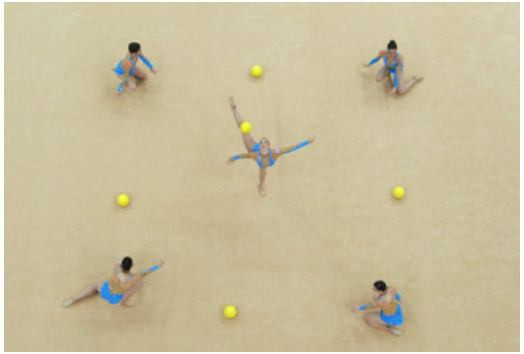
- **Functional (or task) parallelism**: different people are performing different task at the same time



- **Data Parallelism**: different people are performing the same task, but on different equivalent and independent objects

Process Interactions

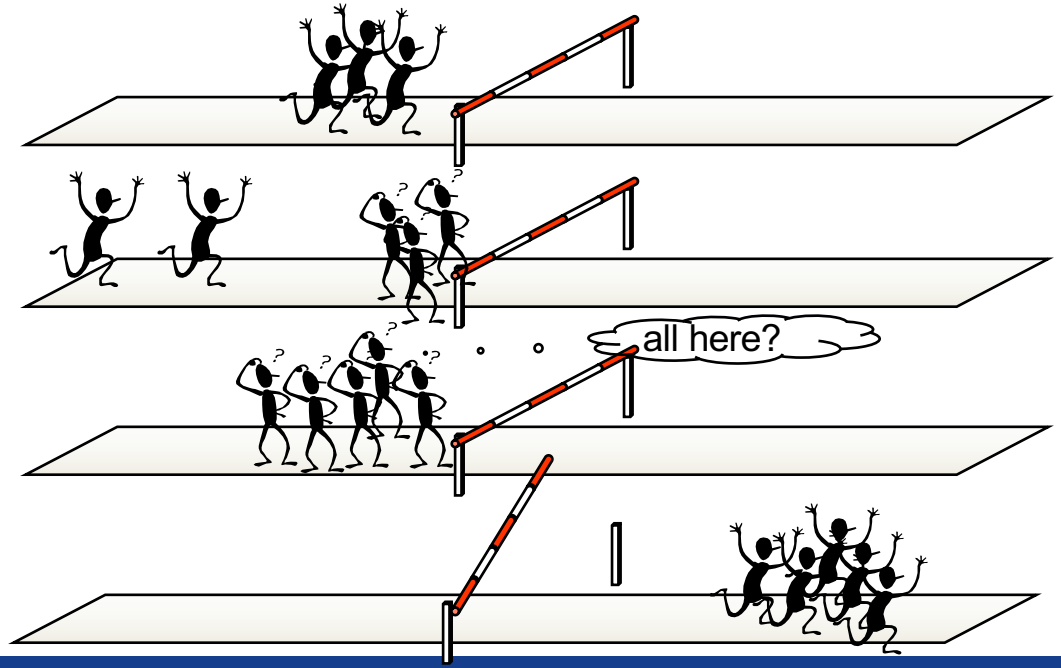
- The effective speed-up obtained by the parallelization depend by the amount of overhead we introduce making the algorithm parallel
- There are mainly two key sources of overhead:
  1. Time spent in inter-process interactions (**communication**)
  2. Time some process may spent being idle (**synchronization**)



Load Balancing

- Equally divide the work among the available resource: processors, memory, network bandwidth, I/O, ...
- This is usually a simple task for the problem decomposition model
- It is a difficult task for the functional decomposition model

Effect of Load Unbalancing



Minimizing Communication

- When possible reduce the communication events:
  - group lots of small communications into large one
  - eliminate synchronizations as much as possible. Each synchronization level off the performance to that of the slowest process

Overlap Communication and Computation

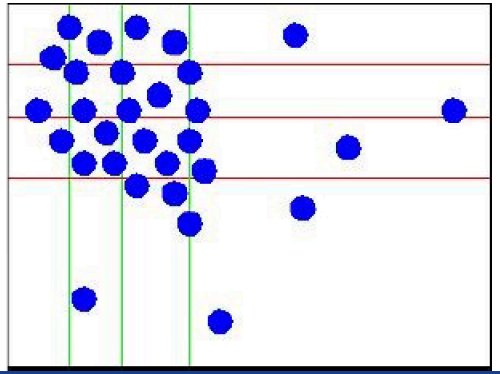
- When possible code your program in such a way that processes continue to do useful work while communicating
- This is usually a non trivial task and is afforded in the very last phase of parallelization
- If you succeed, you have done. Benefits are enormous

Granularity

- Granularity is determined by the decomposition level (number of task) on which we want divide the problem
- The degree to which task/data can be subdivided is limit to concurrency and parallel execution
- Parallelization has to become “topology aware”
  - coarse grain and fine grained parallelization has to be mapped to the topology to reduce memory and I/O contention
  - make your code modularized to enhance different levels of granularity and consequently to become more “platform adaptable”

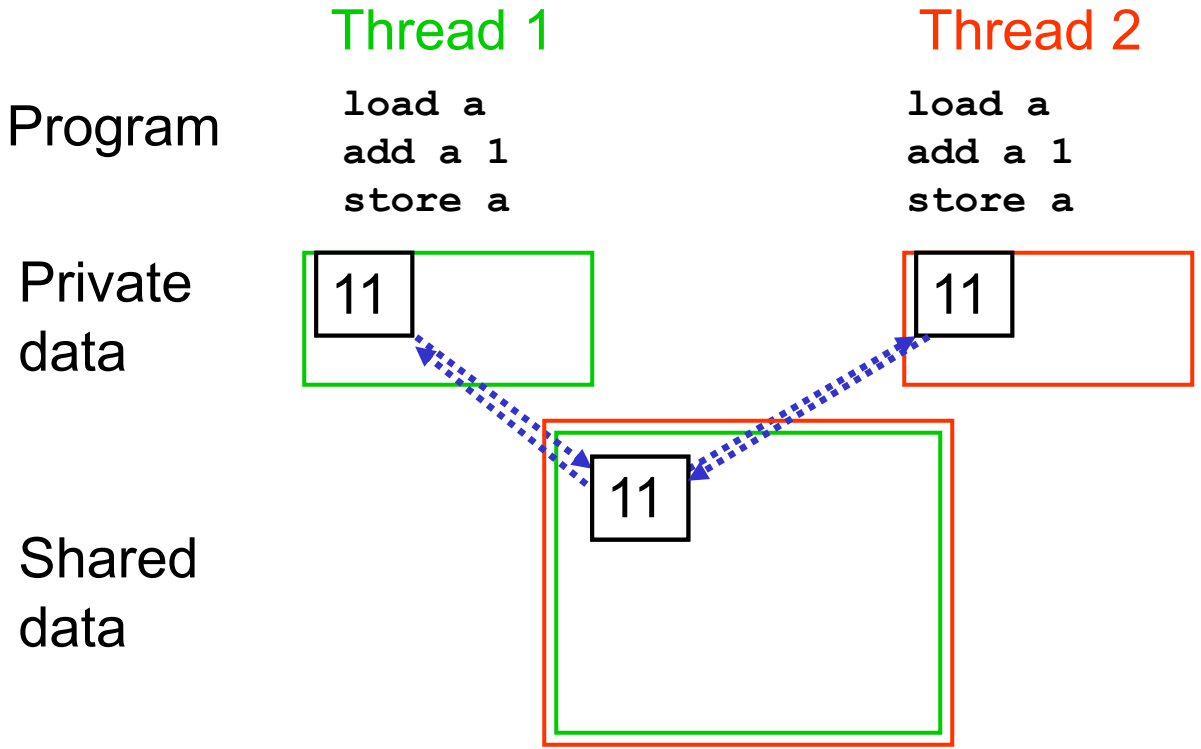
Limitations of Parallel Computing

- Fraction of serial code limits parallel speedup
- Degree to which tasks/data can be subdivided is limit to concurrency and parallel execution
- Load imbalance:
  - parallel tasks have a different amount of work
  - CPUs are partially idle
  - redistributing work helps but has limitations
  - communication and synchronization overhead



Shared Resources

- In parallel programming, developers must manage exclusive access to shared resources
- Resources are in different forms:
  - concurrent read/write (including parallel write) to shared memory locations
  - concurrent read/write (including parallel write) to shared devices
  - a message that must be send and received


Thread 1
Thread 2
Program
load a
add a 1
store a
load a
add a 1
store a
Private data
11
11
Shared data
11

Fundamental Tools of Parallel Programming

MPI Program Design

- Multiple and separate processes (can be local and remote) concurrently that are coordinated and exchange data through “messages”

  => a “share nothing” parallelization
- Best for coarse grained parallelization
- Distribute large data sets; replicate small data
- Minimize communication or overlap communication and computing for efficiency

  => Amdahl's law: speedup is limited by the fraction of serial code plus communication

Phases of an MPI Program

1. Startup
   - Parse arguments (mpirun may add some!)
   - Identify parallel environment and rank of process
   - Read and distribute all data
2. Execution
   - Proceed to subroutine with parallel work (can be same of different for all parallel tasks)
3. Cleanup

CAUTION: this sequence may be run only once

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm,
ierr)

  print *, "After Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_FINALIZE(ierr)

  end program bcast
```

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  end program bcast
```

P_0

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_1

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_2

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_3

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)

  end program bcast
```

P_0

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)

  call MPI_COMM_SIZE(comm, ncpus, ierr)

  call MPI_COMM_RANK(comm, myrank, ierr)
```

P_0

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)

  call MPI_COMM_SIZE(comm, ncpus, ierr)

  call MPI_COMM_RANK(comm, myrank, ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  end program bcast
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

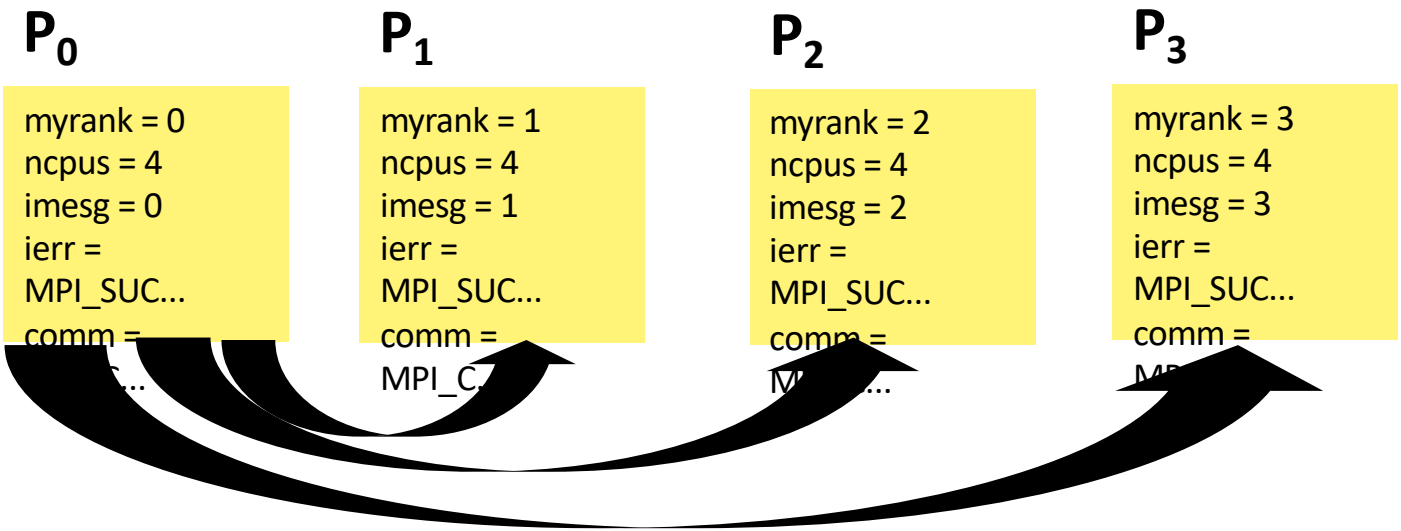
P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)



call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm,
ierr)

  print *, "After Bcast operation I'm ", myrank, &
       " and my message content is ", imesg


  end program bcast
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm,
ierr)

  print *, "After Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  call MPI_FINALIZE(ierr)

end program bcast
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm,
ierr)

  print *, "After Bcast operation I'm ", myrank, &
       " and my message content is ", imesg

  call MPI_FINALIZE(ierr)

end program bcast
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUCC
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

Programming Parallel Paradigms

- Are the tools we use to express the parallelism for on a given architecture (see also SPMD, SIMD, etc...)
- They differ in how programmers can manage and define key features like:
  - parallel regions
  - concurrency
  - process communication
  - synchronism

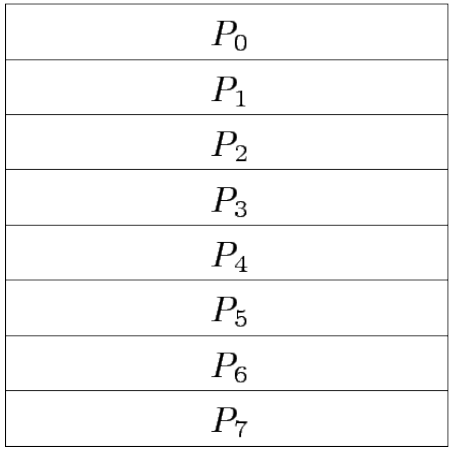
Parallelism - 101

- there are two main reasons to write a parallel program:
  - access to larger amount of memory (aggregated, going bigger)
  - reduce time to solution (going faster)

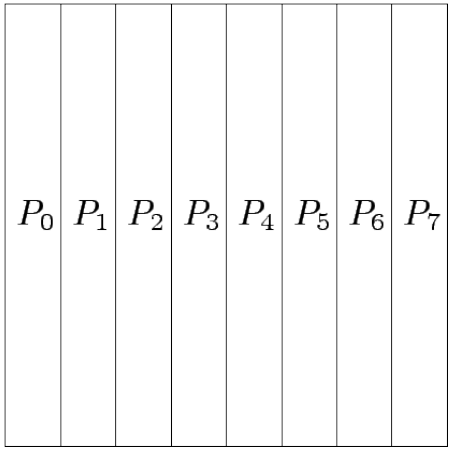
Static Data Partitioning

The simplest data decomposition schemes for dense matrices are 1-D block distribution schemes.

row-wise distribution



column-wise distribution

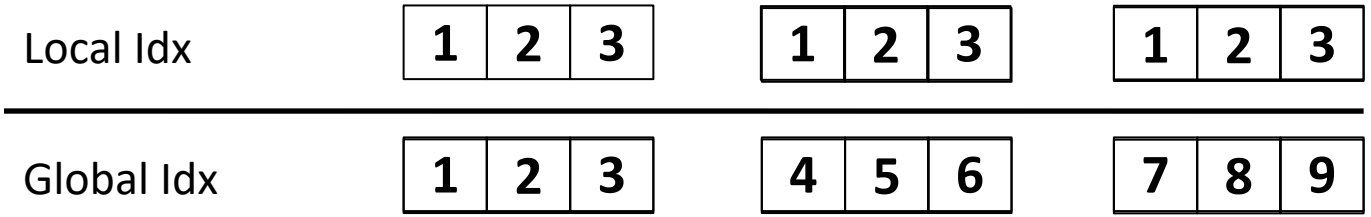


Distributed Data Vs Replicated Data

- Replicated data distribution is useful if it helps to reduce the communication among process at the cost of bounding scalability
- Distributed data is the ideal data distribution but not always applicable for all data-sets
- Usually complex application are a mix of those techniques => distribute large data sets; replicate small data

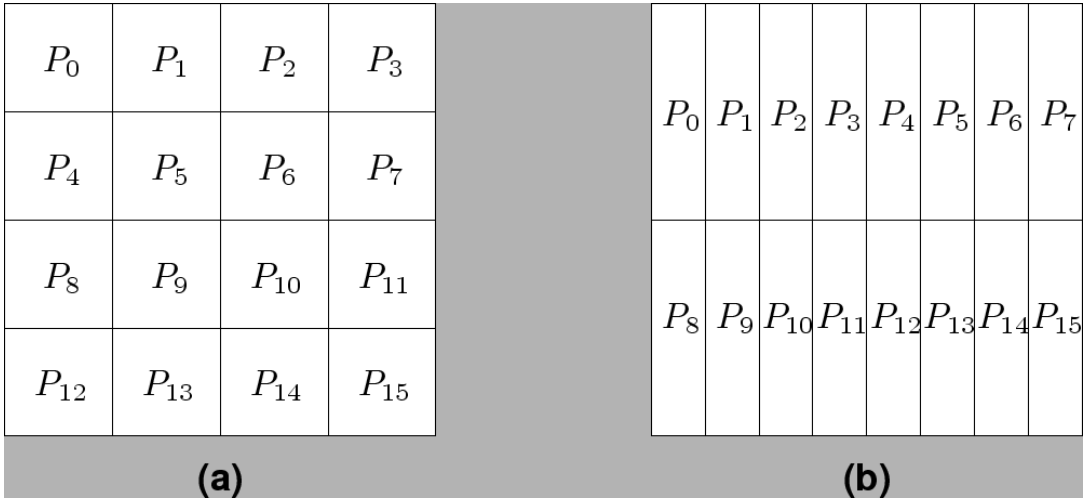
Global Vs Local Indexes

- In sequential code you always refer to global indexes
- With distributed data you must handle the distinction between global and local indexes (and possibly implementing utilities for transparent conversion)



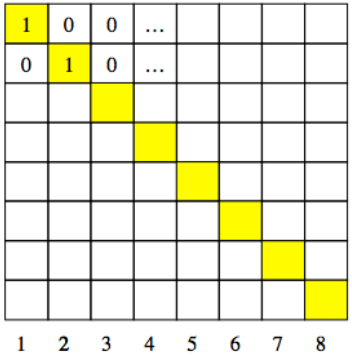
Block Array Distribution Schemes

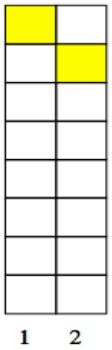
Block distribution schemes can be generalized to higher dimensions as well.

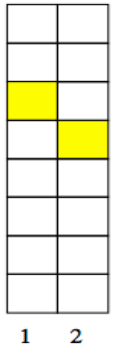


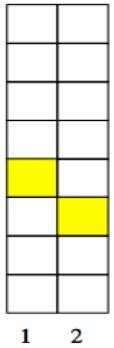
Degree to which tasks/data can be subdivided is limit to concurrency and parallel execution!!

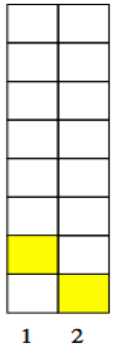
Collaterals to Domain Decomposition /1







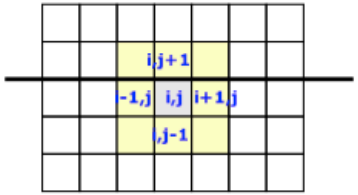




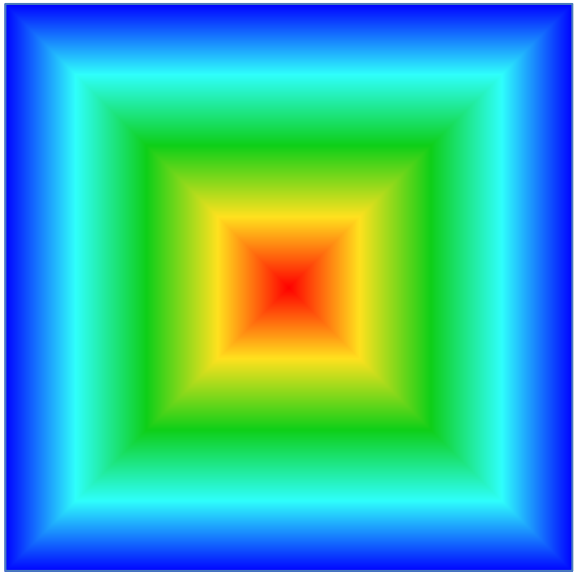
Are all the domain's dimensions always multiple of the number of tasks/processes we are willing to use?

Again on Domain Decomposition

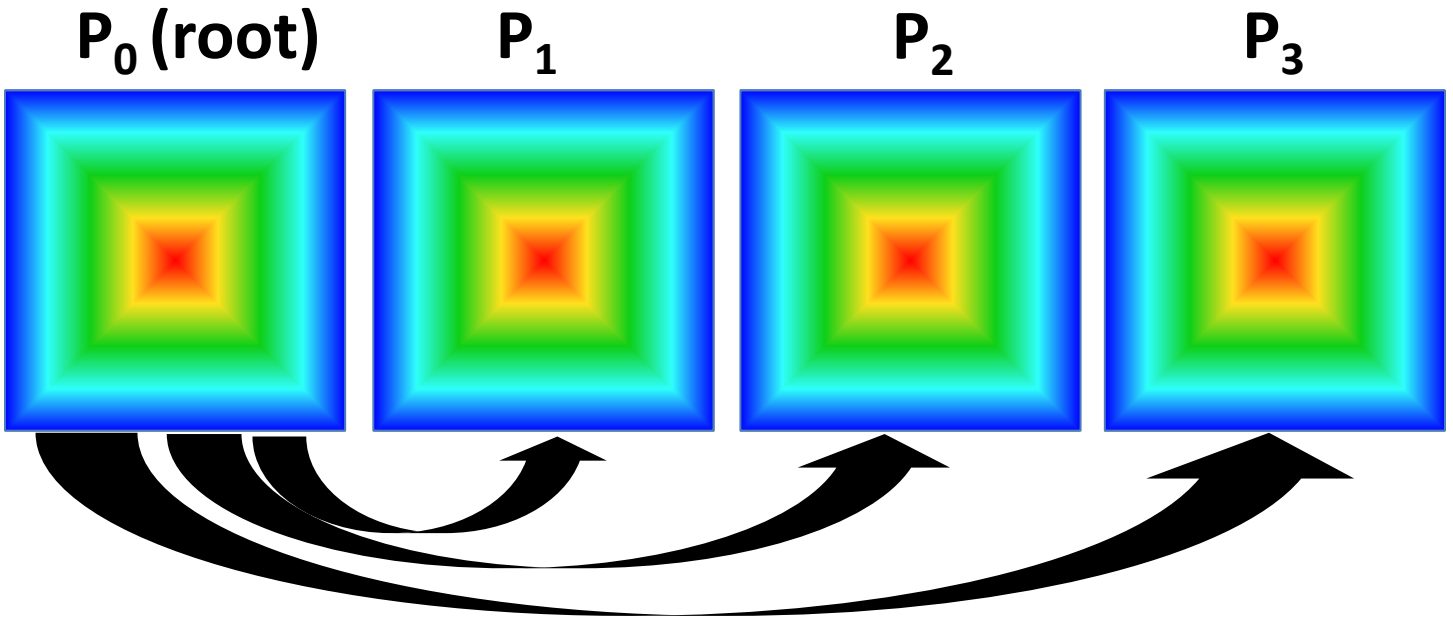
sub-domain boundaries

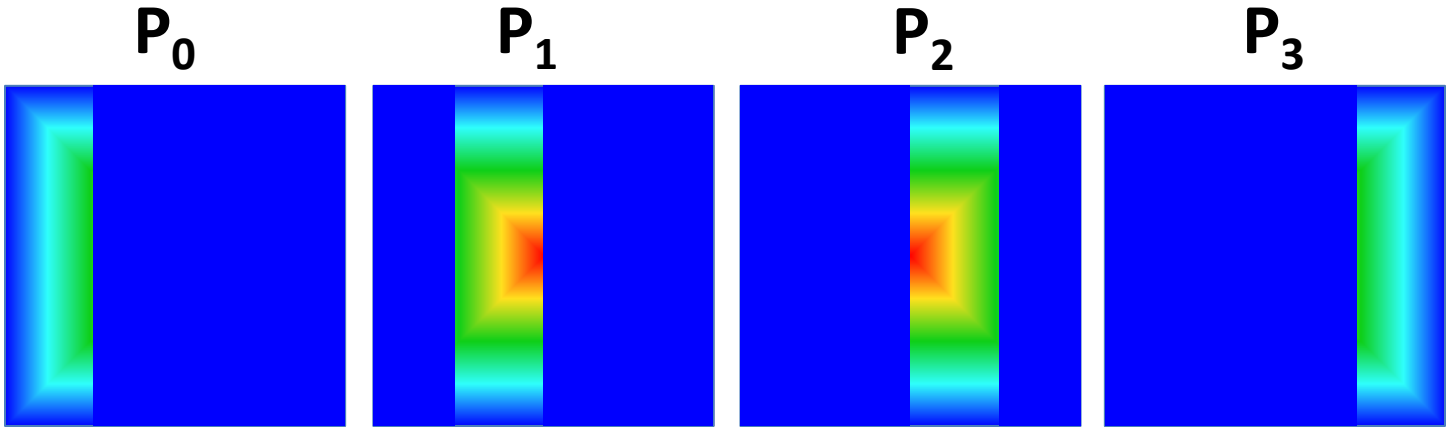


P_0



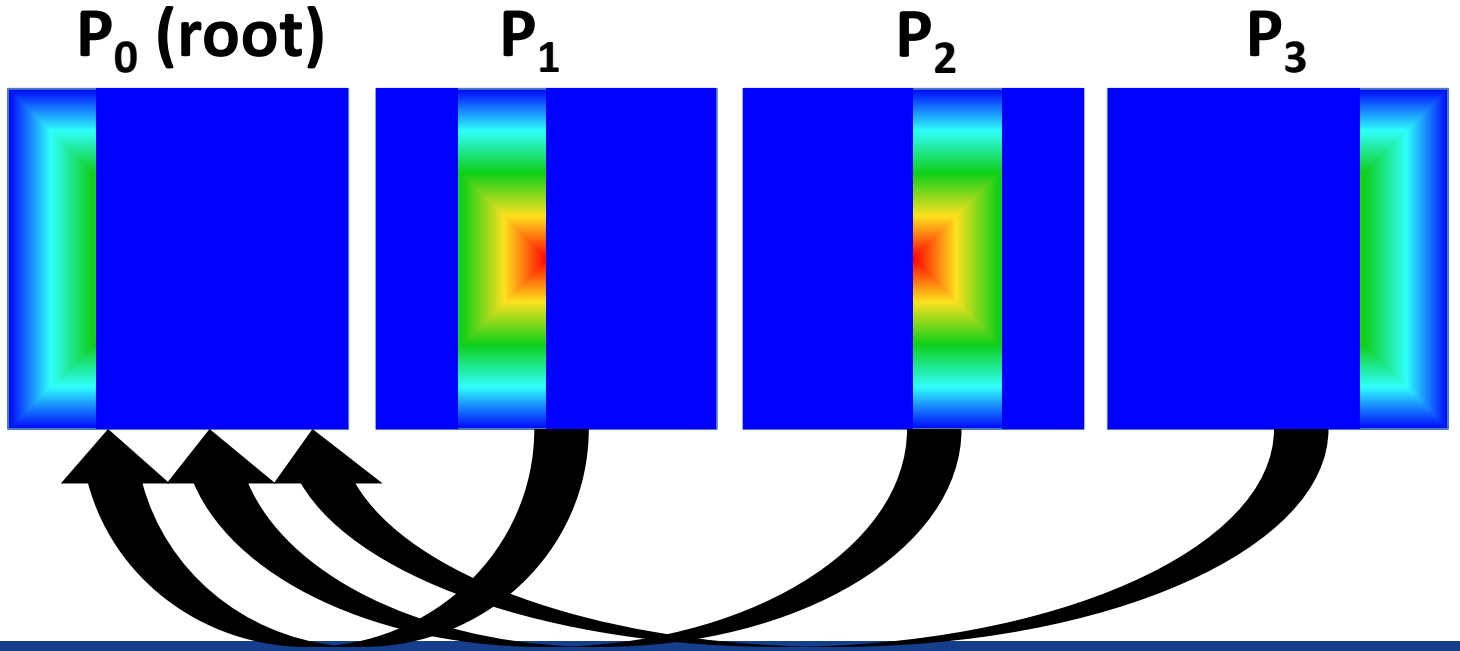
call MPI_BCAST(...)





call evolve(dtfact)

call MPI_Gather(..., ..., ...)

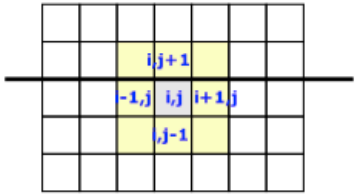


Replicated data

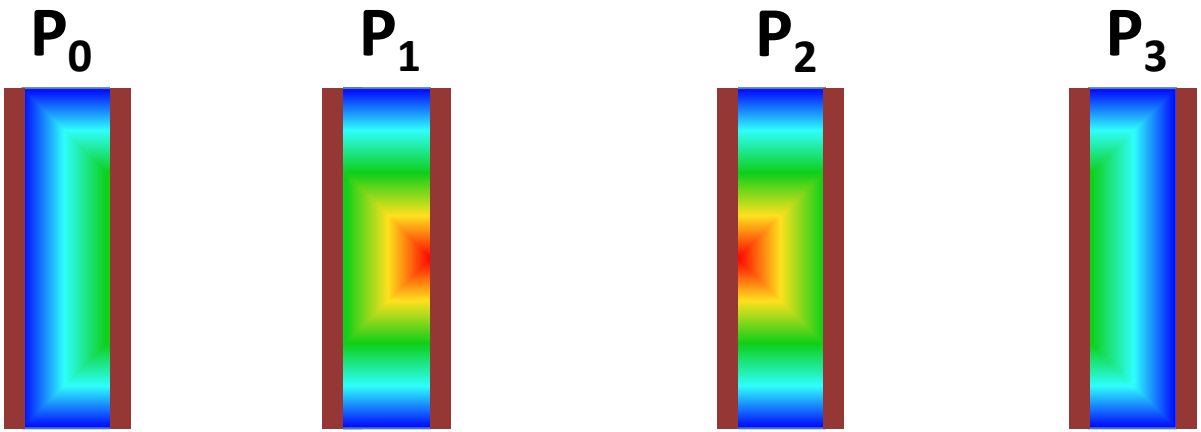
- Compute domain (and workload) distribution among processes
- Master-slaves: P_0 drives all processes
- Large amount of data communication
  - at each step P_0 distribute data to all processes and collect the contribution of each process
- Problem size scaling limited in memory capacity

Collaterals to Domain Decomposition /2

sub-domain boundaries

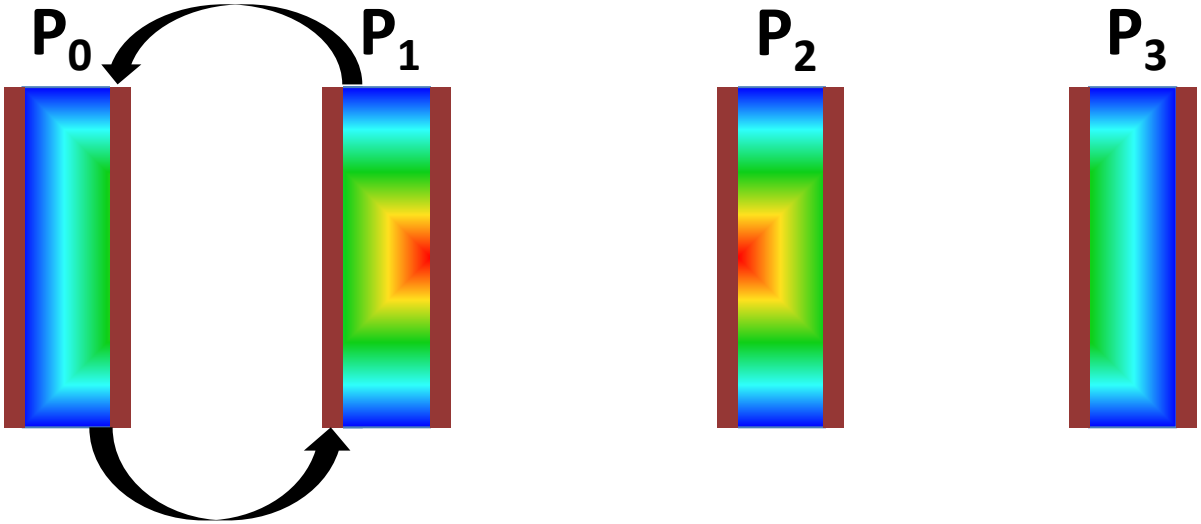


The Transport Code - Parallel Version

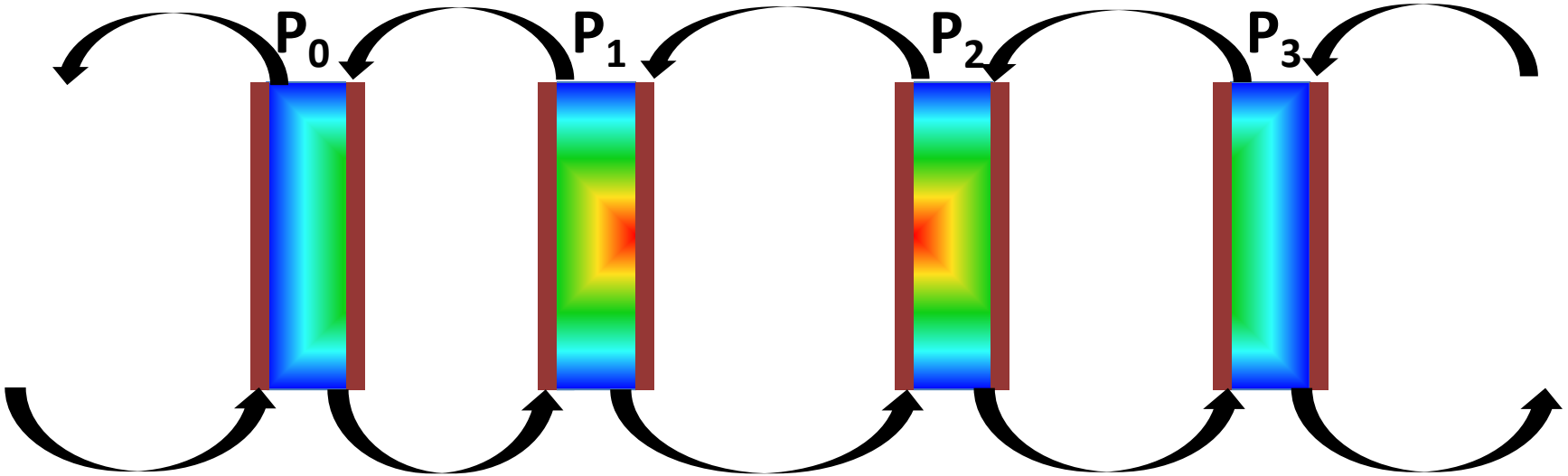


call evolve(dtfact)

Data exchange among processes



proc_down = mod(proc_me - 1 + nprocs , nprocs)

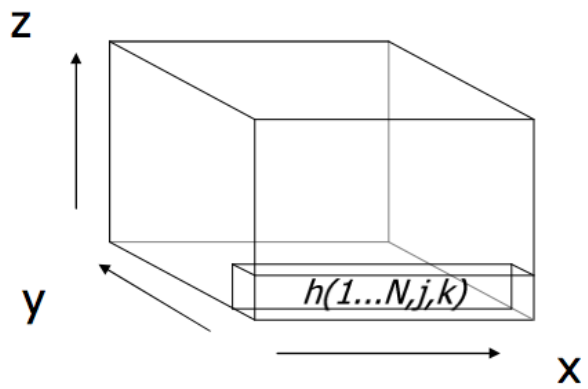


proc_up = mod(proc_me + 1 , nprocs)

Distributed Data

- Global and Local Indexes
- Ghost Cells Exchange Between Processes
  - Compute Neighbor Processes
- Parallel Output

Multidimensional FFT



$$f(x,y,z) = \frac{1}{N_z N_y N_x} \sum_{z=0}^{N_z-1} \Big(\sum_{y=0}^{N_y-1} \Big(\underbrace{\sum_{x=0}^{N_x-1} F(u,v,w)\, e^{-2\pi i \frac{xu}{N_x}}}_{\text{DFT long x-dimension}} \Big) e^{-2\pi i \frac{yv}{N_y}} \Big)}_{\text{DFT long y-dimension}} e^{-2\pi i \frac{zw}{N_z}}$$

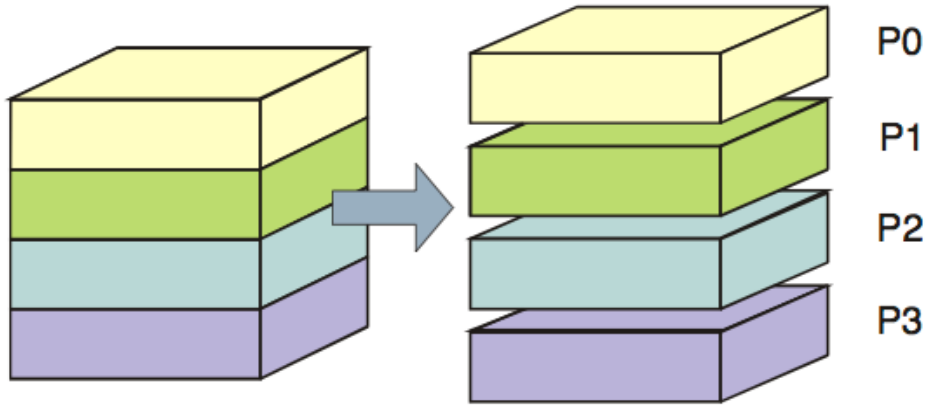
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{DFT long z-dimension}}$$

1) For any value of ***j*** and ***k*** transform the column (1…N, j, k)

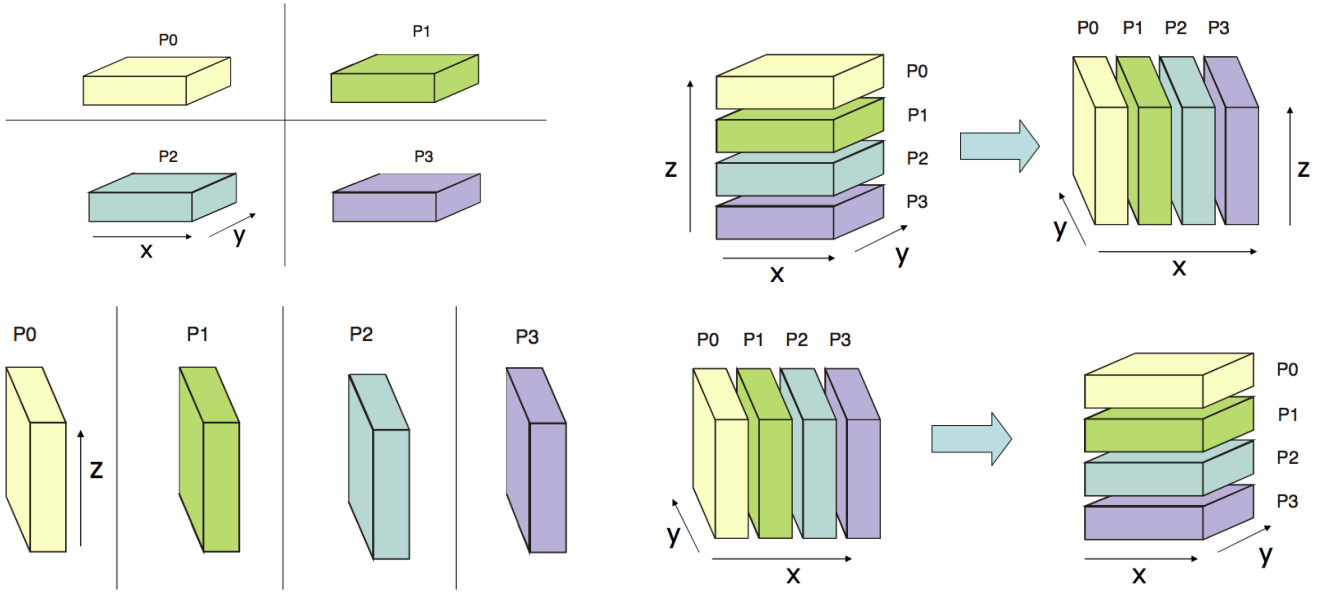
2) For any value of ***i*** and ***k*** transform the column (i, 1…N, k)

3) For any value of ***i*** and ***j*** transform the column (i, j, 1…N)

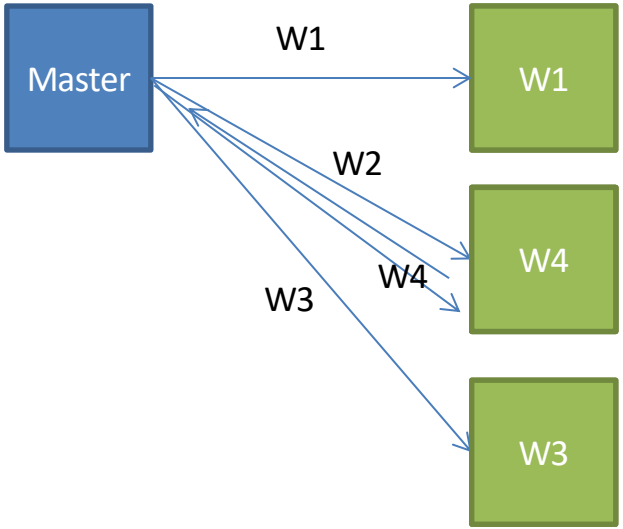
Parallel 3DFFT / 1



Parallel 3DFFT / 2



Master/Slave



Task Farming

- Many independent programs (tasks) running at once
  - each task can be serial or parallel
  - "independent" means they don't communicate directly
  - Processes possibly driven by the mpirun framework

```
[igirotto@localhost]$ more my_shell_wrapper.sh
#!/bin/bash
#example for the OpenMPI implementation
./prog.x --input input_${OMPI_COMM_WORLD_RANK}.dat

[igirotto@localhost]$ mpirun -np 400 ./my_shell_wrapper.sh
```

Easy Parallel Computing

- Farming, embarrassingly parallel
  - Executing multiple instances on the same program with different inputs/initial cond.
  - Reading large binary files by splitting the workload among processes
  - Searching elements on large data-sets
  - Other parallel execution of embarrassingly parallel problem (no communication among tasks)
- Ensemble simulations (weather forecast)
- Parameter space (find the best wing shape)

Three nice reason to use python in scientific programing

NumPy:

http://www.numpy.org/

SciPy:

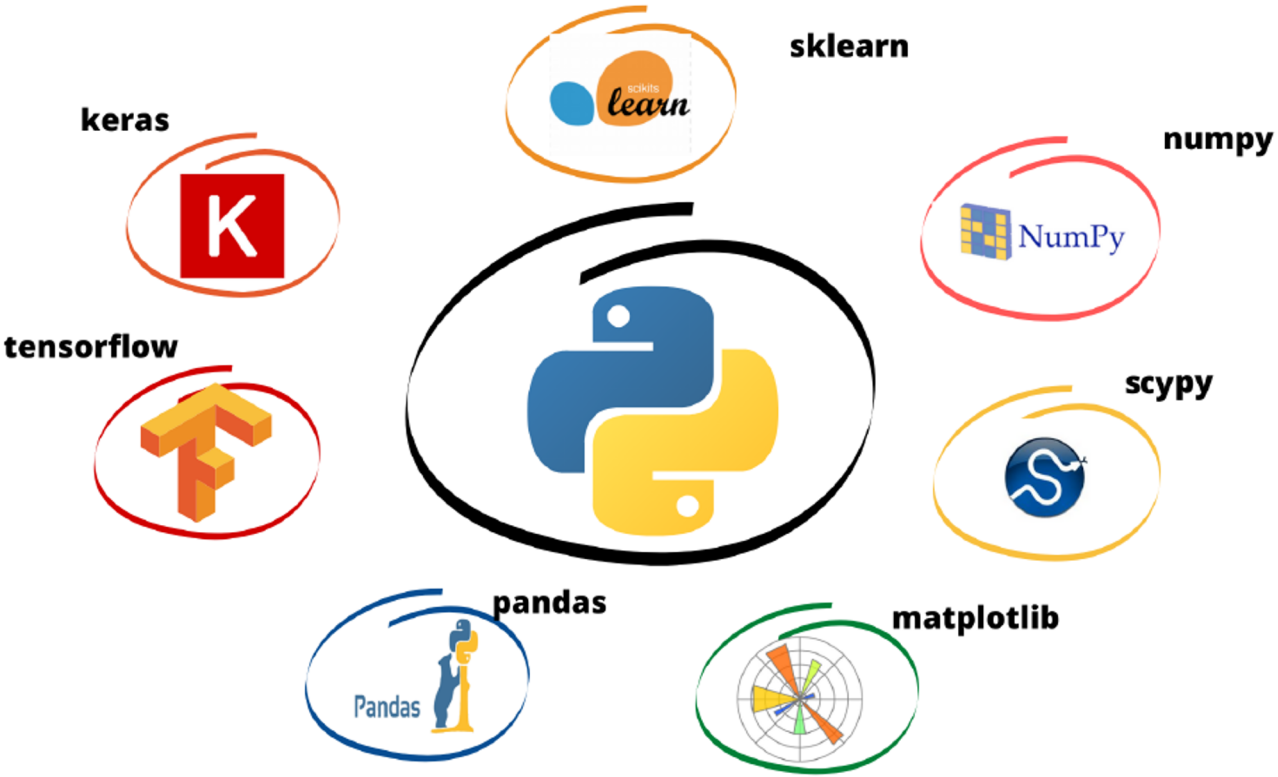
http://www.scipy.org/

MatPlotLib:

http://matplotlib.org/

All are open source!

And more


sklearn
keras
numpy
tensorflow
scypy
pandas
matplotlib
Pandas

Nice reason to use python in scientific programing

NumPy provides functionality to create, delete, manage and operate on large arrays of type raw" data (like Fortran and C/C++ arrays).

SciPy extends NumPy with a collection of useful algorithms like minimization, Fourier transforms, regression and many other applied mathematical techniques.

Both packages are add-on packages (not part of the Python standard library) containing Python code and compiled with (fftpack, BLAS).

MatPlotLib is a nice library to plot (but there are others as Seaborn or Bokeh).

How to import libraries?

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as pp
import matplotlib.pyplot as plt
```

Numpy

NumPy is the fundamental package for scientific computing in Python.

- A powerful N-dimensional array object.
- Sophisticated (broadcasting) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.
- Operations on matrices and vectors in NumPy are very efficient because they are linked to compiled in BLAS/LAPACK code.

NumPy functionality:

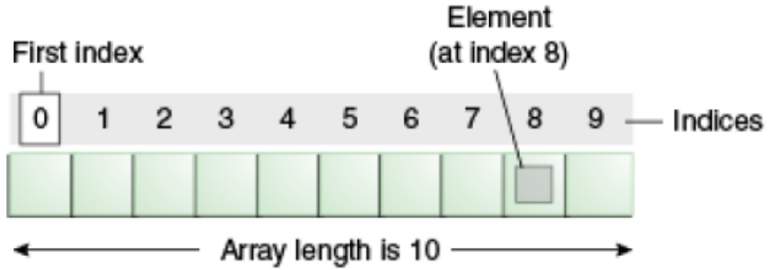
- Polynomial mathematics.
- Statistical computations.
- Pseudo random number generators.
- Discrete Fourier transforms.
- Size / shape / type testing of arrays.

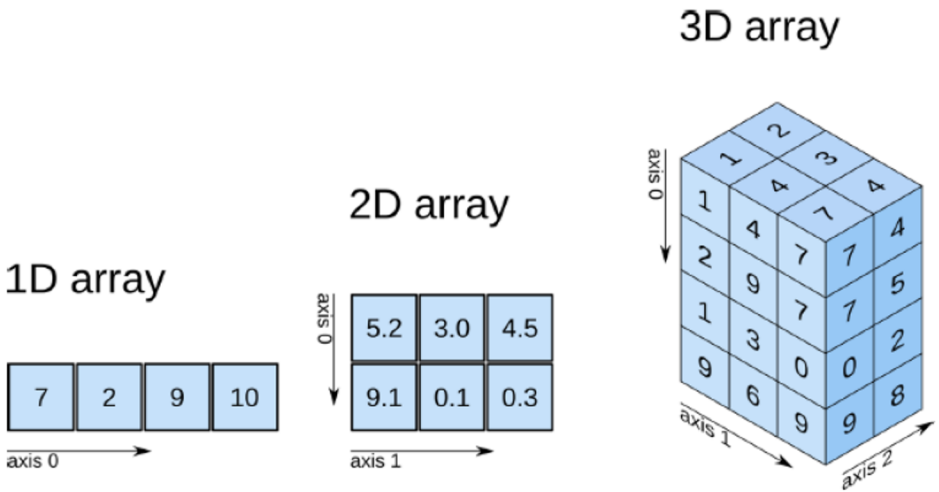
Fundamental thing from Numpy: np.array

There are 5 general mechanisms for creating arrays:

- Conversion from other Python structures (e.g., lists, tuples).
- Intrinsic numpy array creation objects (e.g., arange, ones, zeros, etc.)
- Reading arrays from disk, either from standard or custom formats.
- Creating arrays from raw bytes through the use of strings or buffers.
- Use of special library functions (e.g., random).

How is an Array?





Examples of how to create arrays


```
x = np.array([2, 3, 1, 0])
print(x)
[2 3 1 0]

np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.arange(2, 10, dtype=np.float)
array([2., 3., 4., 5., 6., 7., 8., 9.])

np.arange(2, 3, 0.1)
array([2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

Special Functions

```
a = np.zeros((5, 2))
print(a)
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]

b = a.reshape((2, 5))
print(b)
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

Linear Algebra Operations

```
x = np.array([[1.,2.,3.,4.],[1.,2.,3.,4.]])
print(x)
[[1. 2. 3. 4.]
 [1. 2. 3. 4.]]

b= x.T
print(b)
[[1. 1.]
 [2. 2.]
 [3. 3.]
 [4. 4.]]
```

Simple Linear Algebra Operations

```
v=np.array([-1, 2,5])
w=np.array([1, 2,3])
print(v.dot(w))
18
print(np.dot(v, w))
18

#-----------

v1=np.array([[-1, 2,5],[1, 3,5]])
w1=np.array([1,10,3])
print(v1.dot(w1))
[34 46]
print(np.cross(v,w))
[-4  8 -4]
```

```
x=np.array([1, 2,3])
y=np.array([11, -2,5])
print(x + y)
[12  0  8]
print(np.add(x, y))
[12  0  8]
#----------------

print(x - y)
[-10   4  -2]
print(np.subtract(x, y))
[-10   4  -2]
```

Opening a txt or cvs file


```
import numpy as np

file_name_you_want = np.loadtxt(fname,delimiter=" ")

print "First column element: ",file_name_you_want[0]
#to get the full column:

Transpose_your_file = file_name_you_want.T

print "First column: ", Transpose_your_file[0]
```

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

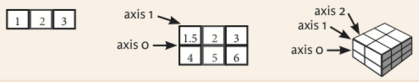
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
 array([[-0.5,  0. ,  0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
 array([[ 2.5,  4. ,  6. ],
        [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
 array([[ 0.66666667,  1.        ,  1.        ],
        [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
 array([[  1.5,   4. ,   9. ],
        [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
 array([[ 7.,  7.],
        [ 7.,  7.]])
```

Subtraction	
Subtraction	
Addition	
Addition	
Division	
Division	
Multiplication	
Multiplication	
Exponentiation	
Square root	
Print sines of an array	
Element-wise cosine	
Element-wise natural logarithm	
Dot product	

Comparison

```
>>> a == b
 array([[False,  True,  True],
        [False, False, False]], dtype=bool)
>>> a < 2
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison	
Element-wise comparison	
Array-wise comparison	

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]
 3
>>> b[1,2]
 6.0
```

Select the element at the 2nd index

Select the element at row 0 column 2 (equivalent to `b[1][2]`)

Slicing

```
>>> a[0:2]
 array([1, 2])
>>> b[0:2,1]
 array([ 2.,  5.])
>>> b[:1]
 array([[1.5, 2., 3.]])
>>> c[1,...]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
>>> a[ : :-1]
 array([3, 2, 1])
```

Select items at index 0 and 1

Select items at rows 0 and 1 in column 1

Select all items at row 0 (equivalent to `b[0:1, :]`)

Same as `[1,:,:]`

Reversed array `a`

Boolean Indexing

```
>>> a[a<2]
 array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
 array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
 array([[ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5],
        [ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5]])
```

Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```

Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
 array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays

Stack arrays vertically (row-wise)

Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)

Create stacked column-wise arrays

Create stacked column-wise arrays

Splitting Arrays

```
>>> np.hsplit(a,3)
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science Interactively

SciPy

One of the core packages. SciPy is built on top of NumPy and implements many specialized scientific computation tools:

- Manly user-friendly.
- Efficient numerical routines such as routines for numerical integration and optimization.
- Clustering.
- Fourier transforms.
- Numerical integration, interpolations. data I/O, LAPACK.

Also:

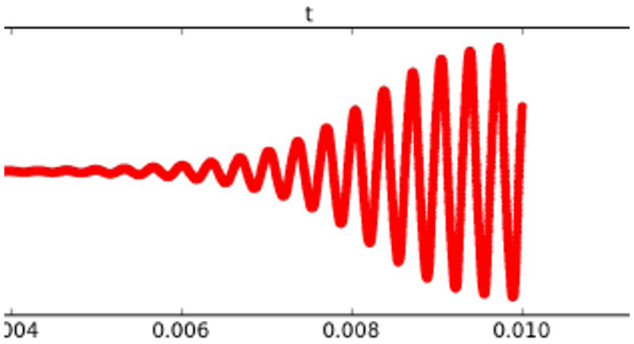
- sparse matrices, linear solvers, optimization.

- signal processing.

- statistical functions

Nice for numerical integration of equations

```
from scipy.integrate import odeint
dy/dt = func(y, t0, ...)
sol = odeint(func, X0, t)
```



https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

Or for example for a simple fit

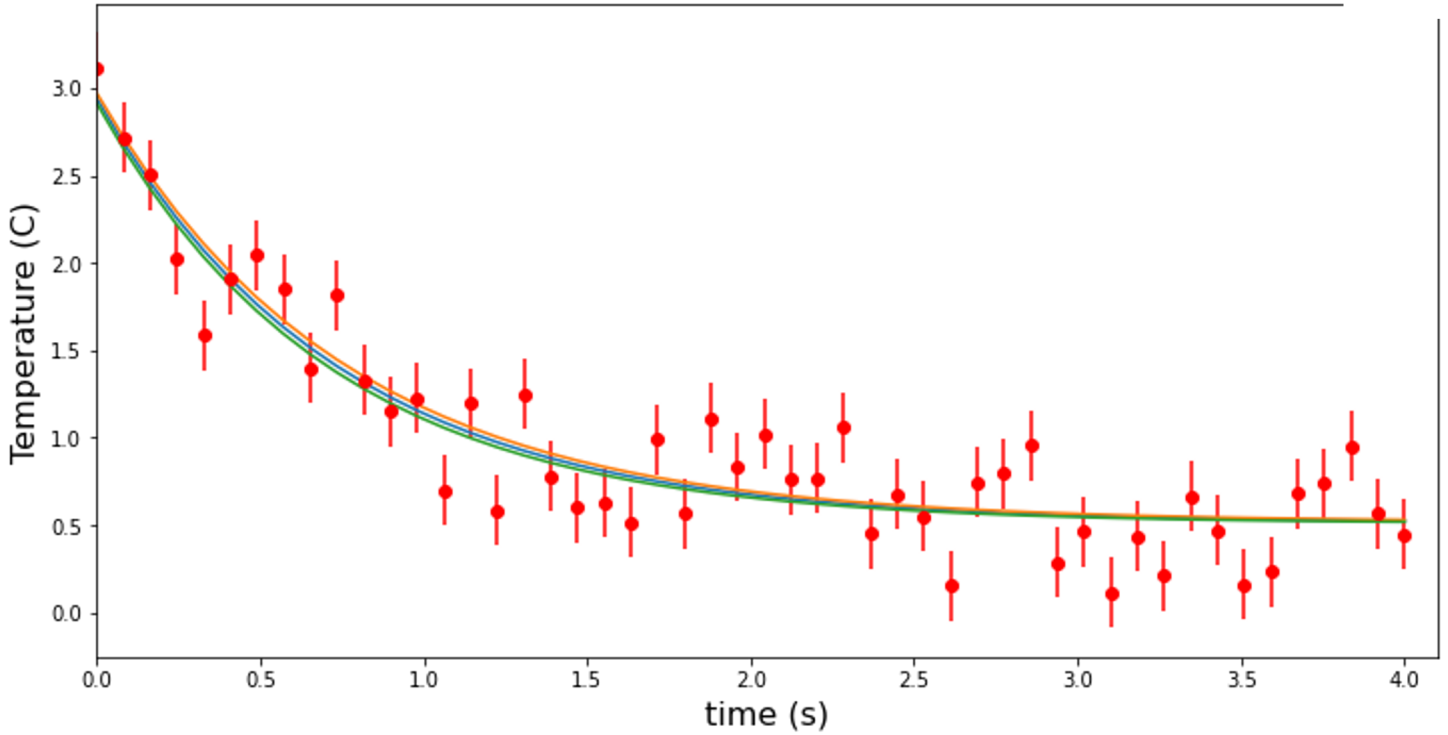
```
import numpy as np
import matplotlib.pyplot as pp
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
   return a*np.exp(-b*t) + c

t = np.linspace(0,4,50)
temp = fitFunc(t, 2.5, 1.3, 0.5)
noisy = temp + 0.25*np.random.normal(size=len(temp))
fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)

pp.figure(figsize=(12, 6))
pp.ylabel('Temperature (C)', fontsize = 16)
pp.xlabel('time (s)', fontsize = 16)
pp.xlim(0,4.1)
pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
pp.show()
```

And the result!



With the Fit parameters

[2.595658 1.74438726 0.69809511]

And covariance matrix:

[[0.02506636 0.01490486 -0.00068609]

2 [0.01490486 0.04178044 0.00641246]

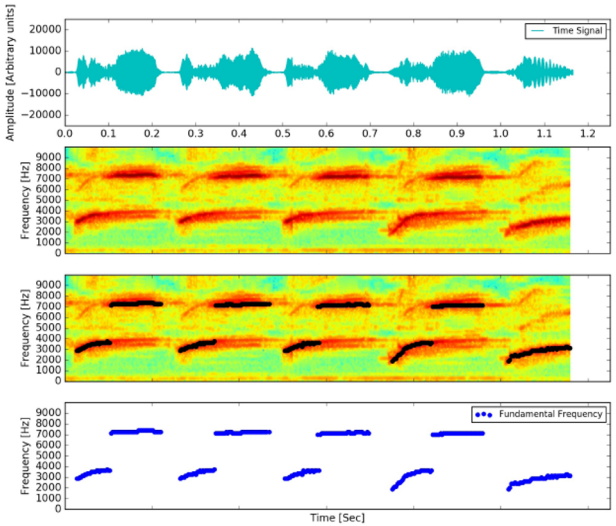
3 [-0.00068609 0.00641246 0.00257799]]

Also possible to apply for signal analysis and time series

DOI:https://doi.org/10.31527/analesafa.2018.29.2.51

https://doi.org/10.1016/j.mex.2018.12.011

DOI:https://doi.org/10.31527/analesafa.2019.30.3.68

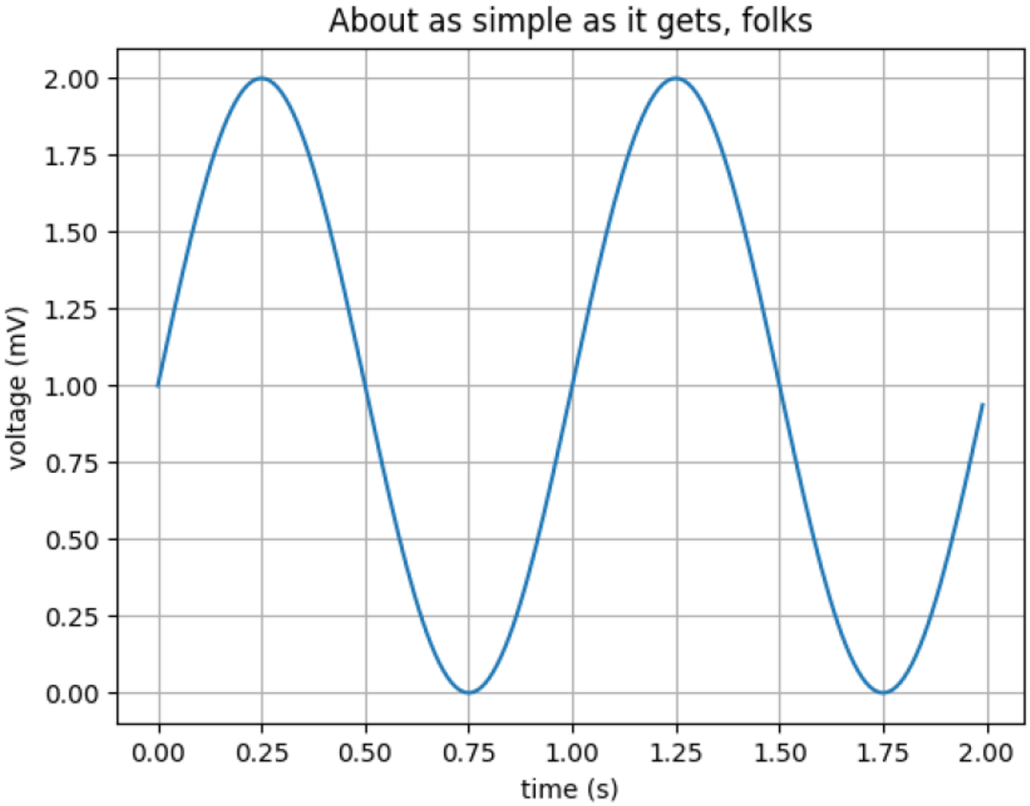


MatplotLib

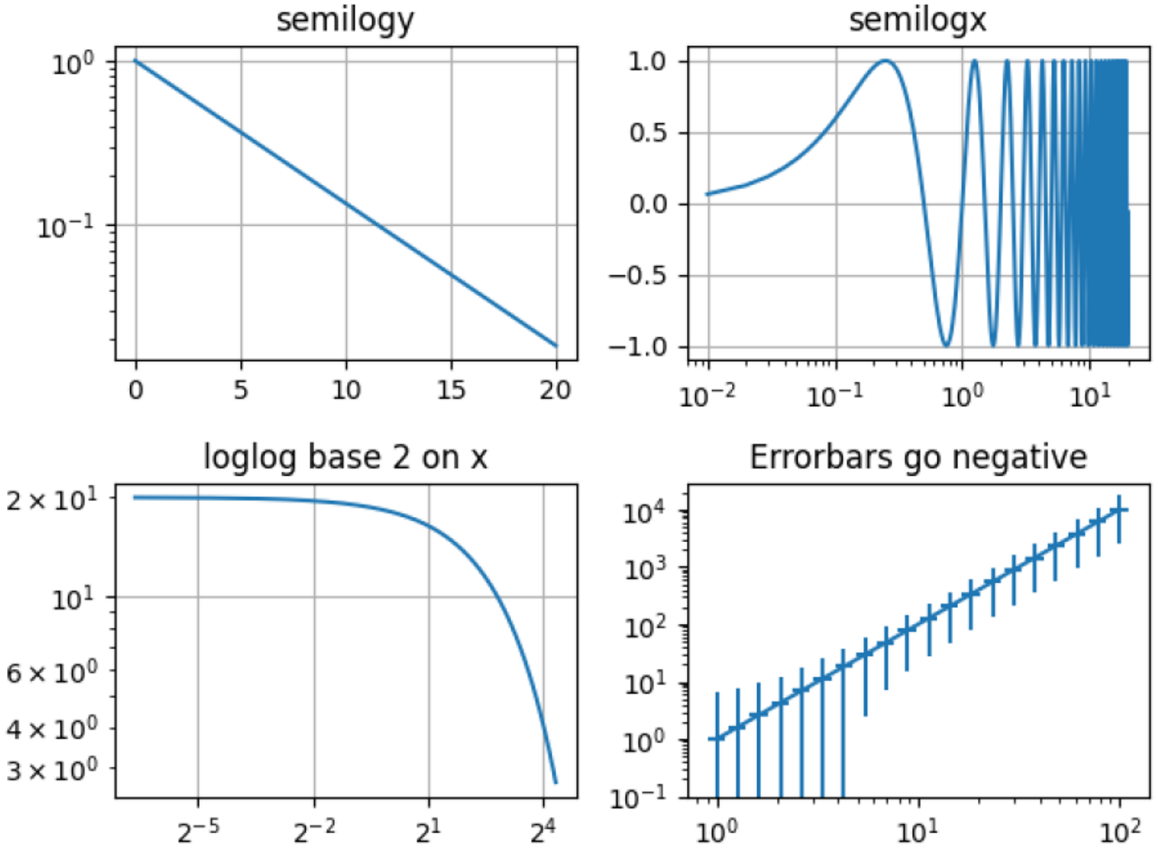
Is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Can be used in:

- Python scripts.
- The Python and IPython shell.
- The jupyter notebook.
- Web application servers.
- Graphical user interface toolkits.

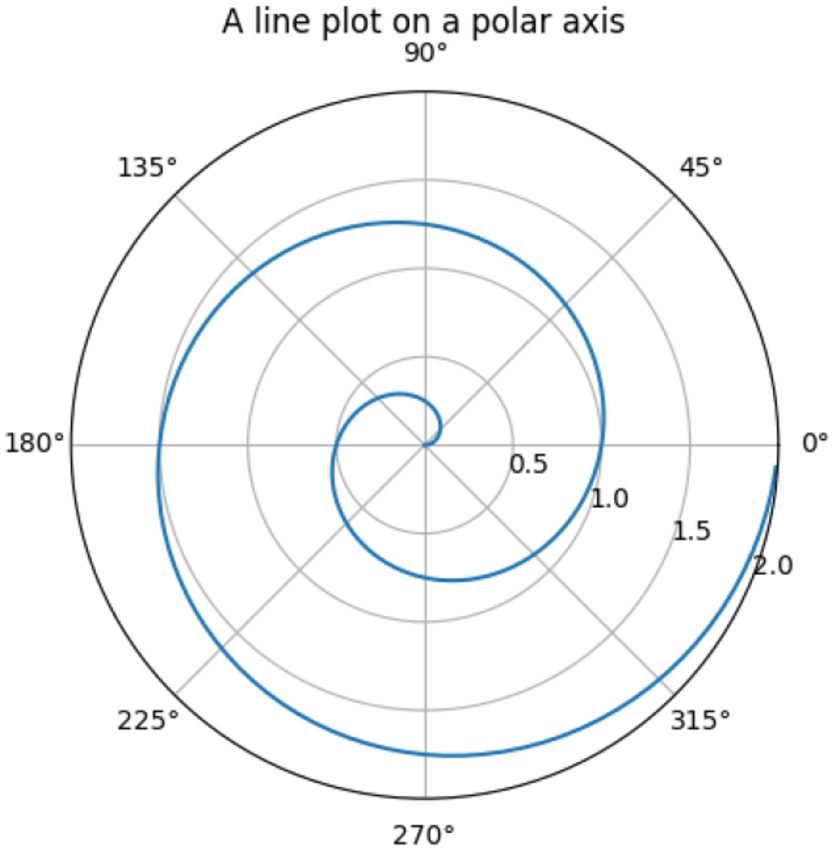
Different kind of plots: y vs. x



Different scales:

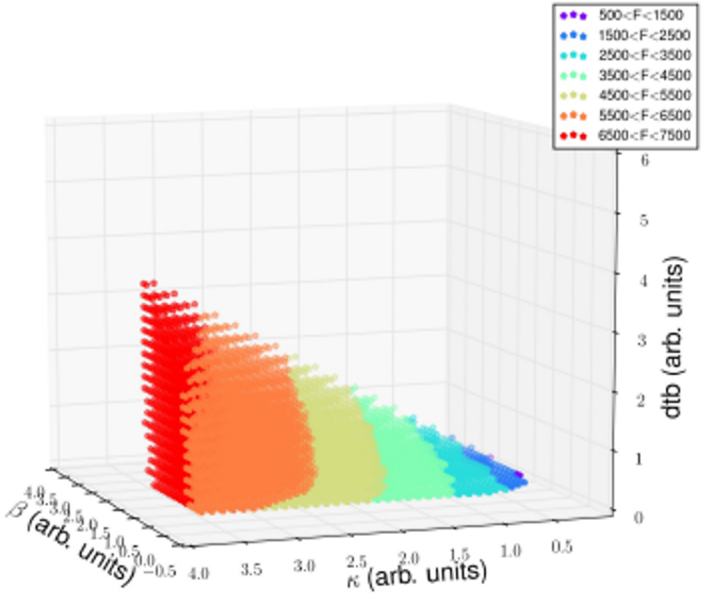


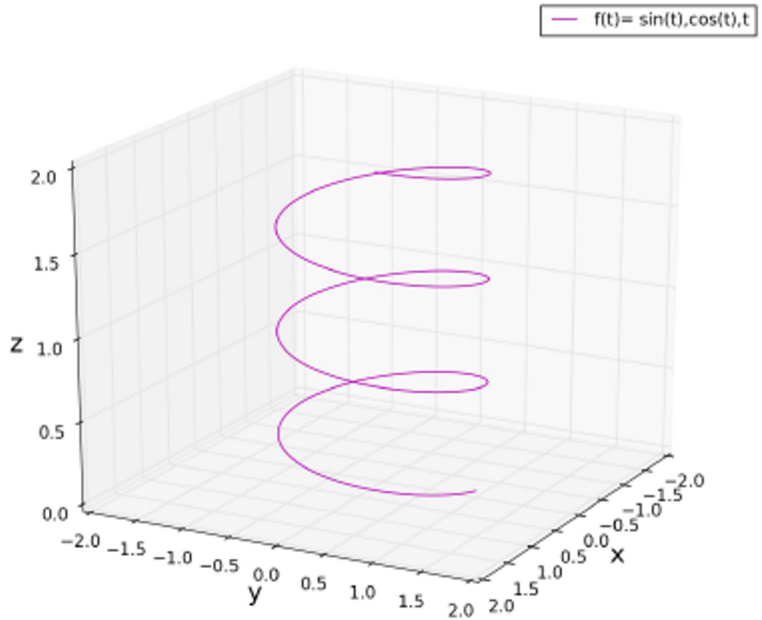
Different coordinates



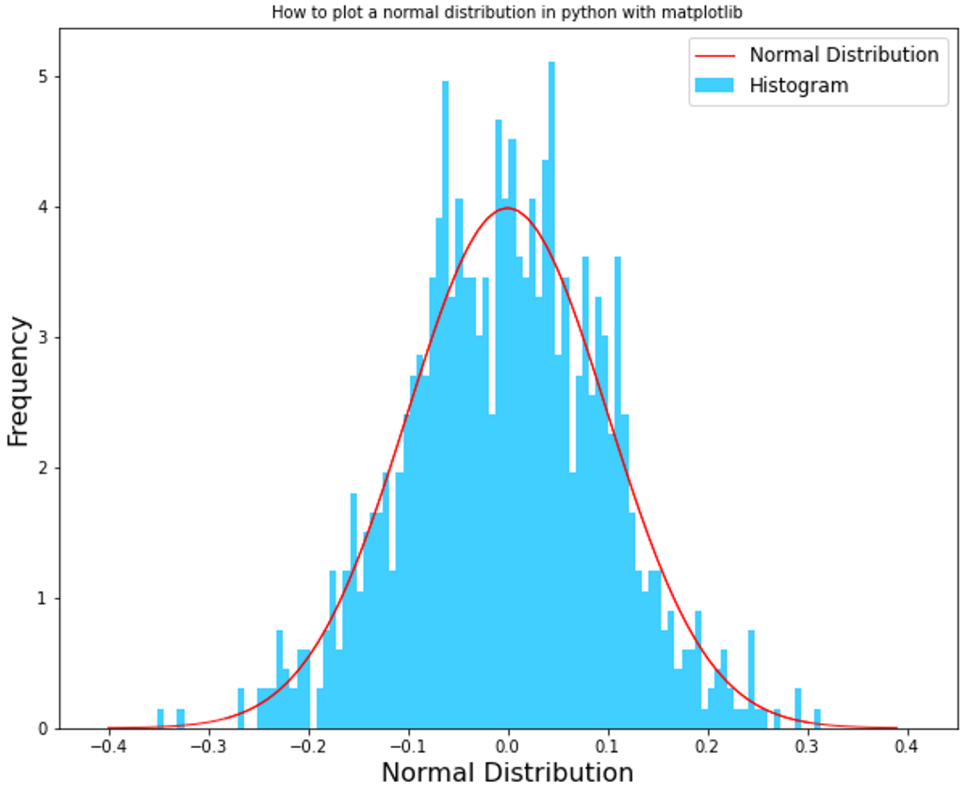
matplotlib

3d plots:





Histograms and distributions



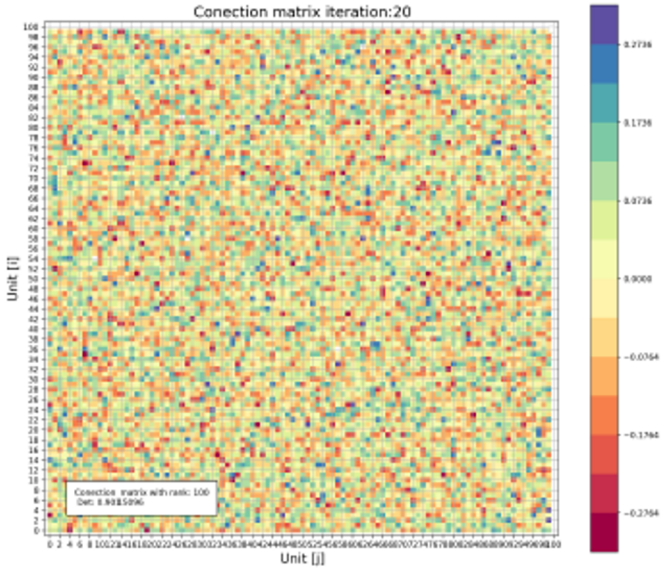
The code for the histogram:

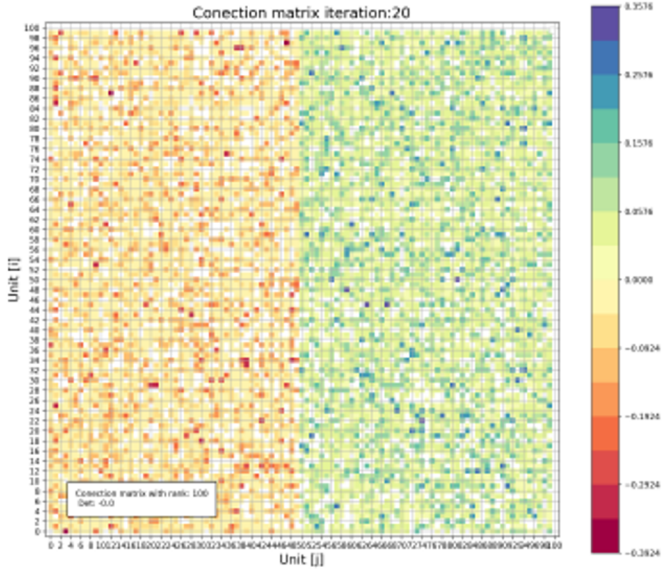

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
mu, sigma = 0, 0.1 # mean and standard deviation
s = np.random.normal(mu, sigma, 1000)
x_n = np.arange(-0.4,0.4,0.01)
y_n = norm.pdf( x_n, mu, sigma)

fig     = plt.figure(figsize=(10,8))
plt.title('Histogram Normal Distribution', fontsize = 18)
plt.hist(s, 100, density=1, facecolor='deepskyblue',label='Histogram', alpha=0.75)
plt.plot(x_n, y_n, 'r-', linewidth=1,label='Normal Distribution')
plt.show()
```

Matrices:

Numpy Scipy Matplotlib





Please follow the examples in Documentation

there are a lot!

https://matplotlib.org/stable/gallery/index.html

More on visualization with python

https://clauswilke.com/dataviz/

https://www.python-graph-gallery.com/

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

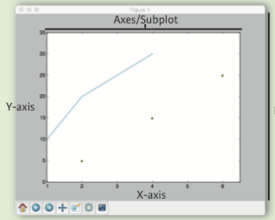
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

http://pandas.pydata.org/

To import library

```
import pandas as pd
```

```
A unidimensional array with labels:
```

```
s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd']
```

```
print(s)
```

```
a    3
b   -5
c    7
d    4
dtype: int64
```

A bi-dimensional array

```
data = {'Country': ['Belgium', 'India', 'Brazil'], 'Capital': ['Brussels',
'New Delhi', 'Brasilia'], 'Population': [11190846, 1303171035, 207847528]}

df = pd.DataFrame(data,columns=['Country', 'Capital', 'Population'])

print(df)
```

```
  Country    Capital  Population
0  Belgium   Brussels    11190846
1    India  New Delhi  1303171035
2   Brazil   Brasilia   207847528
```

pandas

To read a file and write it with pandas

```
pd.read_csv('file.csv', header=None, nrows=5)

df.to_csv('myDataFrame.csv')
```

To read multiple sheets of the same file

```
xlsx = pd.ExcelFile('file.xls')
df = pd.read_excel(xlsx, 'Sheet1')
```

To request help!

```
help(pd.Series.loc)

           mark ii           1       4
    viper      mark ii           7       1
               mark iii         16      36

    Single label. Note this returns a DataFrame with a single index.

    >>> df.loc['cobra']
             max_speed  shield
    mark i          12       2
    mark ii          0       4

    Single index tuple. Note this returns a Series.

    >>> df.loc[('cobra', 'mark ii')]
    max_speed    0
    shield       4
    Name: (cobra, mark ii), dtype: int64

    Single label for row and column. Similar to passing in a tuple, this
    returns a Series.

    >>> df.loc['cobra', 'mark i']....
```

To select one element

```
s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd']
s['b']
-5
```

To select a subset

```
data = {'Country': ['Belgium', 'India', 'Brazil'], 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
'Population': [11190846, 1303171035, 207847528]}

df = pd.DataFrame(data,columns=['Country', 'Capital', 'Population'])


df[1:]
```

	Country	Capital	Population	
1	India	New Delhi	1303171035	
2	Brazil	Brasilia	207847528	

Retrieving Series/DataFrame Information

```
#(rows, columns)
df.shape

#Describe index
df.index

#Describe DataFrame columns

df.columns
```

```
#Info on DataFrame

df.info()

#Number of non-NA values

df.count()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column      Non-Null Count
Dtype
---  ------      --------------  --
---
 0   Country     3 non-null
object
 1   Capital     3 non-null
object
 2   Population  3 non-null
int64
dtypes: int64(1), object(2)
memory usage: 200.0+ bytes
Country       3
Capital       3
Population    3
dtype: int64
```

Applying Functions

```
f = lambda x: x*2
#Apply function
df.apply(f)

#Apply function element-wise
df.applynap(f)
```