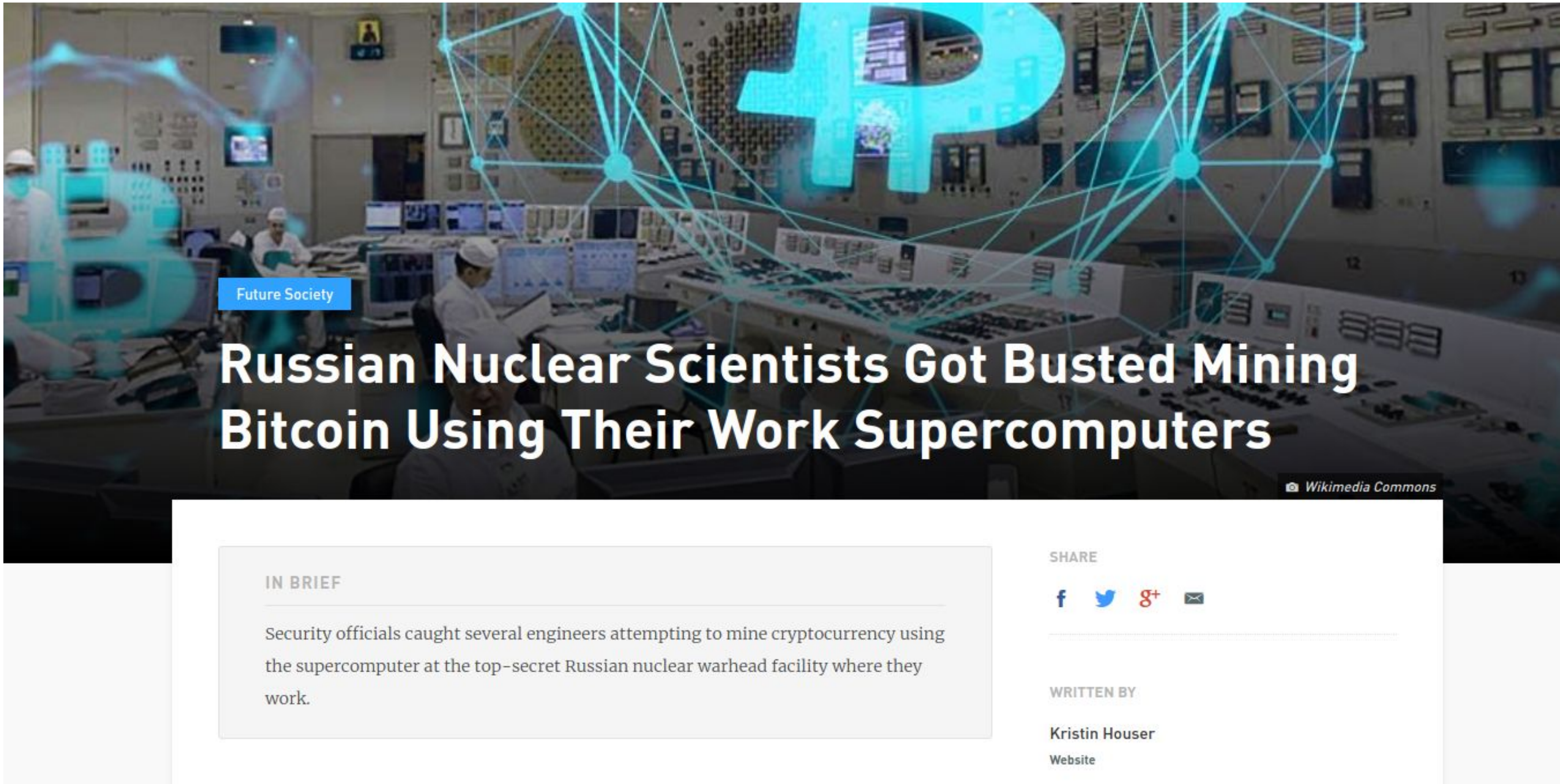
Inspira Crea Transforma

UNIVERSIDAD EAFIT®

Programación Paralela con Python

Juan David Pineda-Cárdenas

Centro de Computación Científica APOLO


Future Society

Russian Nuclear Scientists Got Busted Mining Bitcoin Using Their Work Supercomputers

Wikimedia Commons

IN BRIEF

Security officials caught several engineers attempting to mine cryptocurrency using the supercomputer at the top-secret Russian nuclear warhead facility where they work.

SHARE

WRITTEN BY

Kristin Houser

Website

Tomado de: https://futurism.com/russian-nuclear-scientists-busted-mining-bitcoin-work-supercomputers/

Figura: Casa Slytherin - Harry Potter

Taxonomía de Flynn

SISD Single Instruction stream Single Data stream	**SIMD** Single Instruction stream Multiple Data stream
MISD Multiple Instruction stream Single Data stream	**MIMD** Multiple Instruction stream Multiple Data stream

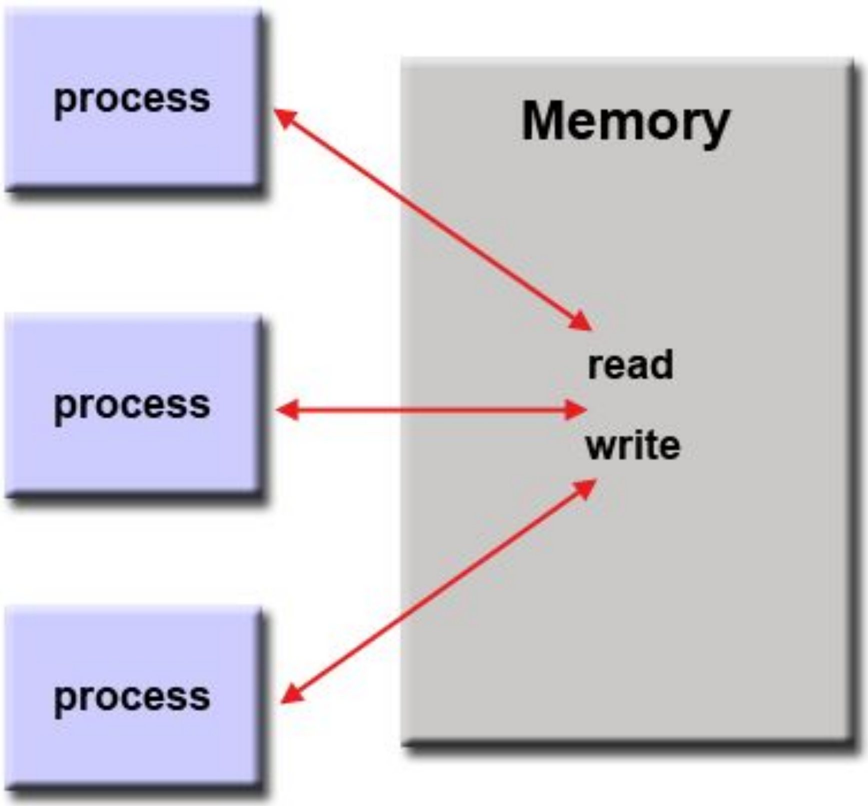
Tomado de: Introduction to Parallel Computing
https://computing.llnl.gov/tutorials/parallel_comp/

Modelos de Programación Paralela

- Memoria compartida
  - Procesos
  - Hilos
- Memoria distribuida
  - Paso de mensajes

Modelos de Programación Paralela: Memoria compartida (Sin hilos)

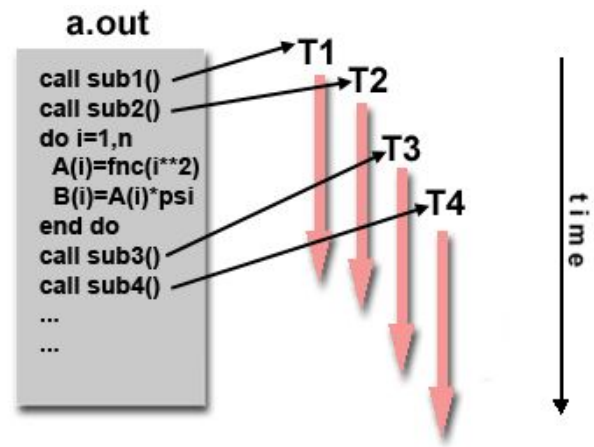
- Los procesos o tareas tienen un espacio de direcciones común, éstos escriben leen y/o escriben asíncronamente.
- Mecanismos como locks y semáforos se usan para controlar el acceso a la memoria compartida, previniendo *deadlocks* y condiciones de carrera
- Es probablemente el modelo más simple

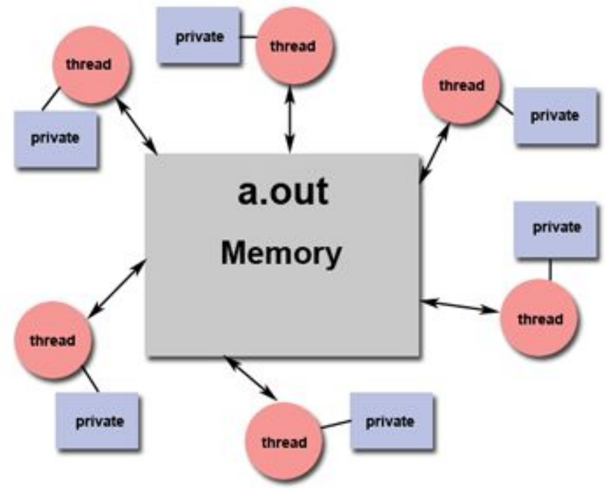


Tomado de: Introduction to Parallel Computing
https://computing.llnl.gov/tutorials/parallel_comp/

Modelos de Programación Paralela: Memoria compartida (Sin hilos)

- Es un tipo de programación de memoria compartida.
- Un simple “proceso pesado” puede tener múltiples “procesos ligeros”, con caminos de ejecución paralelos.
- Cada hilo tiene sus propios datos locales, pero también comparte todos sus recursos con su proceso.
- Los hilos se comunican entre sí usando la memoria global. Esto requiere construcciones de sincronización que manejen la concurrencia



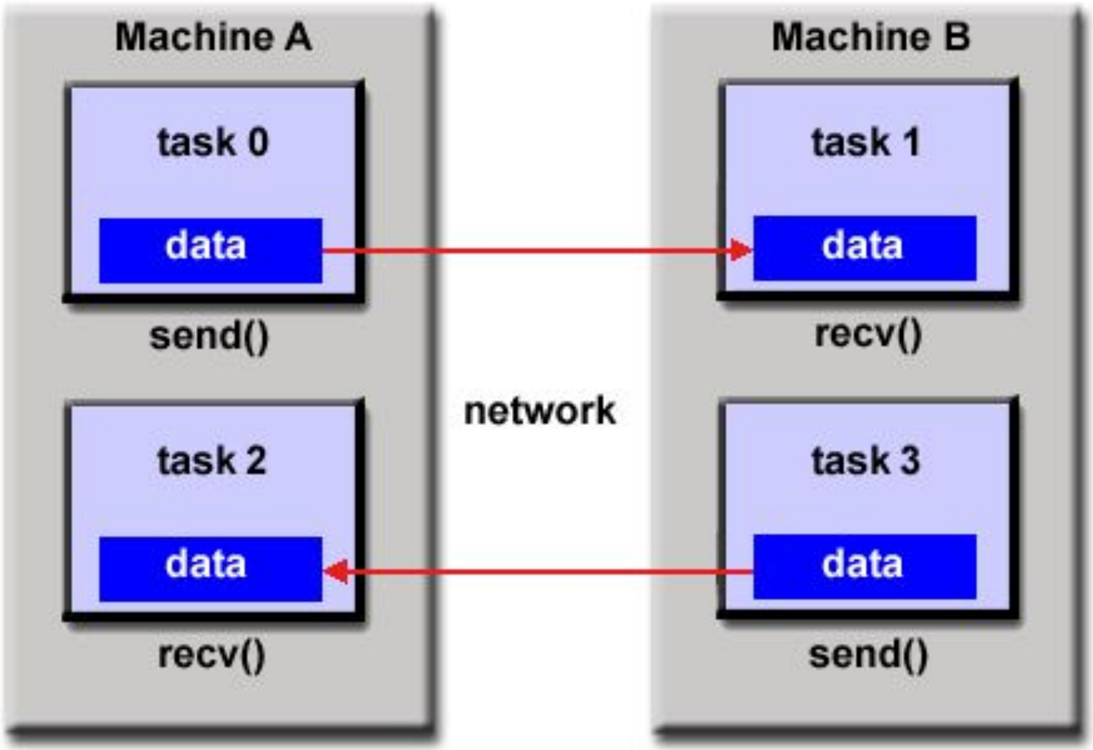


Modelos de Programación Paralela: Memoria compartida (Sin hilos) Implemetaciones

- POSIX Threads
  - Especificado por IEEE 1003.1c (1995). Solo lenguaje C
  - Parte de las bibliotecas de los sistemas UNIX/Linux
  - Basado en bibliotecas
  - Comúnmente conocido como PThreads
  - Paralelismo MUY explícito, requiere una atención al detalle significativa por parte del programador
- OpenMP
  - Estándar de la industria, soportado y creado por un grupo de fabricantes y de software, organizaciones e individuos bastante amplio.
  - Basado en directivas del compilador
  - Portable/Multiplataforma, incluyendo Unix y Windows
  - Disponible en C/C++ y Fortran
  - Puede ser bastante simple de usar, provee “paralelismo incremental” y puede empezar con código serial.

Modelos de Programación Paralela: Memoria distribuida: Paso de mensajes

- Un conjunto de tareas que usan su propia memoria local durante el cómputo. Múltiples tareas pueden residir en la misma máquina física y/o entre entre un número arbitrario de máquinas.
- Las tareas intercambian datos a través de las comunicaciones, enviando y recibiendo mensajes.
- La transferencia de datos usualmente requiere operaciones cooperativos realizadas por cada uno de los procesos. Por ejemplo, una operación de envío tiene que tener una operación de recepción como contraparte.

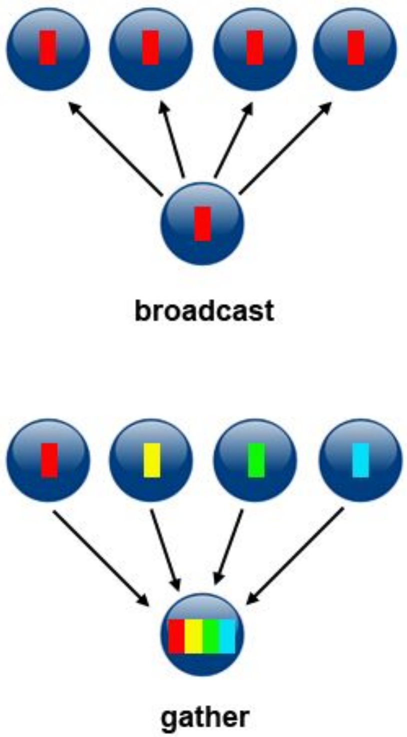


Tomado del Lawrence Livermore National Laboratory: Introduction to Parallel Computing

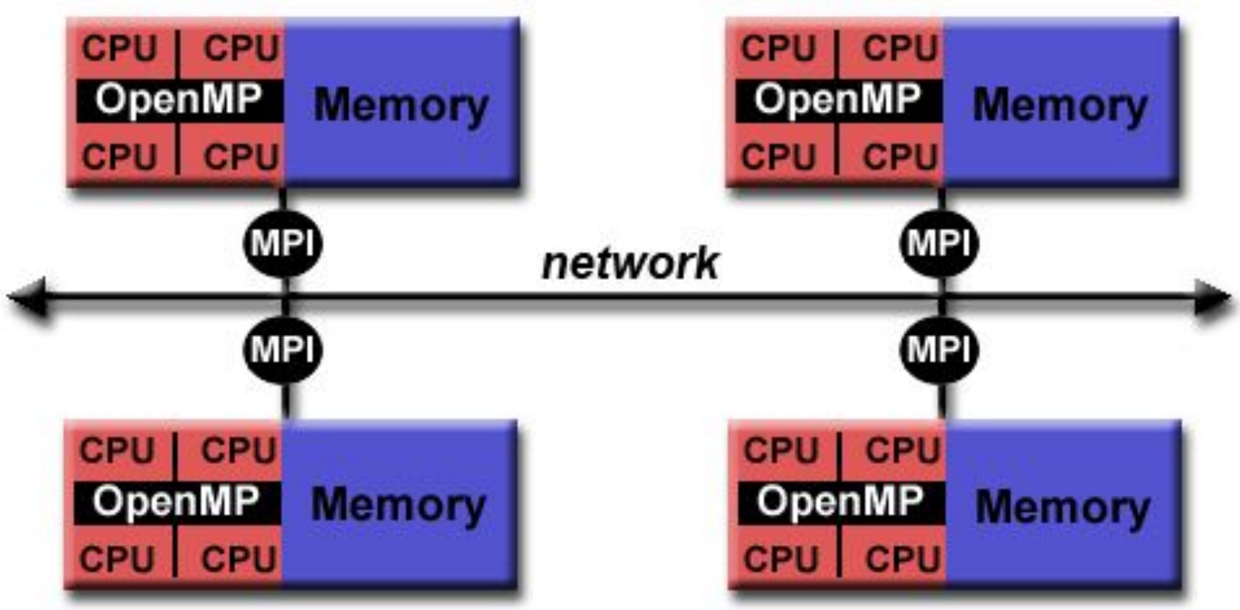
Memoria Distribuida / Paso de Mensajes Implementaciones

- Desde una perspectiva de programación, suele ser un conjunto de bibliotecas de subrutinas. El programador es responsable de determinar todo el paralelismo.
- Desde los 80's se han realizado esfuerzos al respecto, hasta 1992 que se crea el MPI Forum
- MPI-1 1994
- MPI-2 1996
- MPI-3 2012

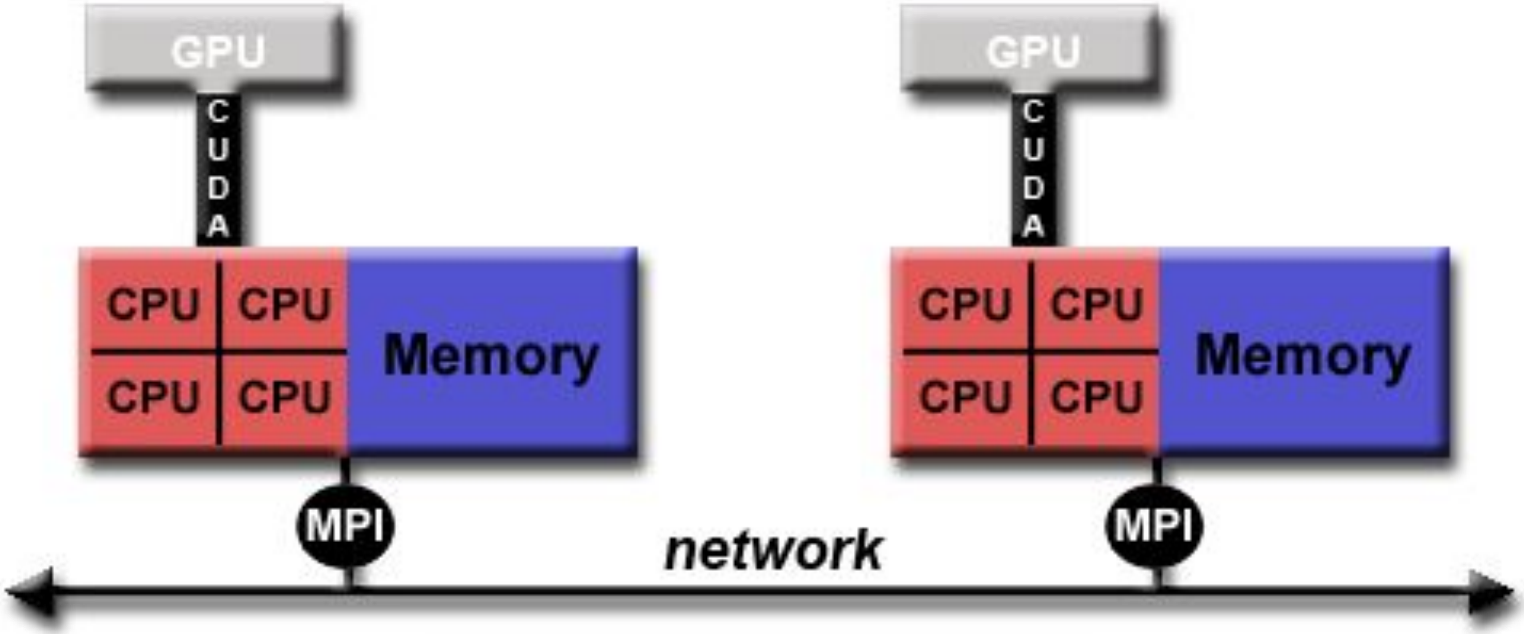
Directivas MPI



Tomado del Lawrence Livermore National Laboratory: Introduction to Parallel Computing



Tomado del Lawrence Livermore National Laboratory: Introduction to Parallel Computing



Tomado del Lawrence Livermore National Laboratory: Introduction to Parallel Computing

inside HPC

News | HPC Hardware | HPC Software | Industry Segments | White Papers | Resources | Special Reports | Parallel Programming

Sign up for our newsletter and get the latest HPC news and analysis.

Email Address

Daily

Weekly

Subscribe

Home » HPC Software » Intel Releases Optimized Python for HPC

Intel Releases Optimized Python for HPC

February 9, 2017 by Richard Friedman

Sponsored Post

Is Python overtaking Java as the most popular programming language? Python has successfully found its way into nearly all the major application areas, including data science and machine learning, financial transactions, academic research, and, in particular, numerical computing for science and engineering.

Why is this surprising? Because Python, which was introduced in 1991 as a good way to learn programming, has been long thought to be a slow interpretive language that's just not the right choice for large industrial-strength applications.

However, the myths about Python seem no longer to be true. Easy to learn and understand, Python comes with a very active user community. As a result, today Python is a significant contender, especially with younger or beginner programmers. What attracts people to Python is its clean style, and the vast number of open source library packages that seem to have solutions for everything worth doing. If your day job is data analysis or computational physics and not computer science, and programming is just one of the tools you use to get work done, Python is now a reasonable choice.

FEATURED JOB

High Performance Computing System Administrator

EMBRY-RIDDLE Aeronautical University

Embry-Riddle Aeronautical University

Daytona Beach

Learn More »

Other Jobs

- High Performance Computing Systems Administrator
- Software Developer (Exascale Computing)
- HPC System Administrator

See all Jobs | Post a Job

LATEST VIDEO

Recent Videos

INDUSTRY PERSPECTIVES

Exploiting Modern Microarchitectures: Meltdown, Spectre, and other Attacks

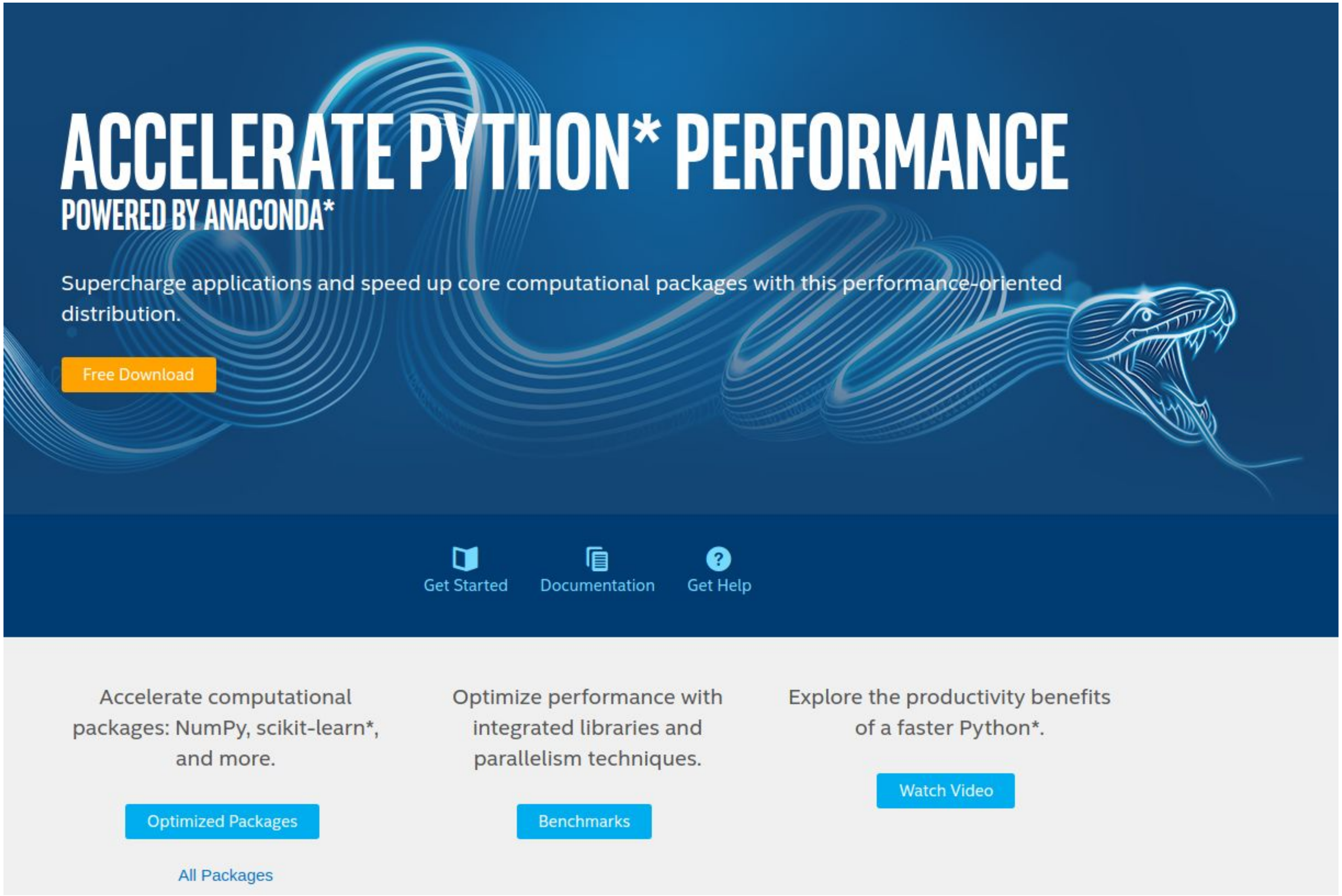
Jon Masters from Red Hat gave this Keynote at FOSDEM 2018. "Recently disclosed vulnerabilities against modern high performance computer microarchitectures known as 'Meltdown' and 'Spectre' are among an emerging wave of hardware-focused attacks. This talk will describe several of these attacks, how they can be mitigated, and generally what we can do as an industry to bring performance without trading security." [Read More...]

WHITE PAPERS

Transforming Financial Services with AI Technologies

As the financial industry increasingly realizes the impact of faster analytical insights on

Tomado de: https://insidehpc.com/2017/02/python-for-hpc-2/



https://software.intel.com/en-us/distribution-for-python

Lo bueno y lo malo

- Comunidad
- Fácil de Leer y Aprender
- Orientado a Objetos
- ¡Portable!
- Cantidad de librerías
  a. Numpy
  b. Scipy
  c. Mpi4py

- Interpretado
- Dos versiones
- 3.x no retrocompatible con 2.x
- Dinámicamente tipado

ANACONDA®

https://anaconda.org/

http://jupyter.org/

https://www.jetbrains.com/pycharm-edu/download/

Código en C

```
#include <stdio.h>
#include <stdlib.h>

int comp(const void * a,const void * b)
{
  const int *ia = (const int *)a;
  const int *ib = (const int *)b;
  return *ia  - *ib;
}

int main(int argc, char **argv) {
  int* array;
  int i;
  array = (int*) malloc(3*sizeof(int));
  array[0] = 4;
  array[1] = 2;
  array[2] = 6;

  int* array2;
  array2 = (int*) malloc(4*sizeof(int));
  for ( i=0; i < 3; i++ )
    array2[i] = array[i];
  array2[3] = 1;
  free(array);
  array = array2;

...
```

```
...

  printf("Before sorting\n");
  for ( i=0; i < 4; i++ )
    printf("%d ", array[i]);
  printf("\n");

  qsort(array, 4, sizeof(int),comp) ;
  printf("After sorting\n");
  for ( i=0; i < 4; i++ )
    printf("%d ", array[i]);
  printf("\n");
}
```

Tomado de: Python in High Performance Computing. Jussi Enkovaara.
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf

Código en Python

```
array = [4, 2, 6]
array.append(1)
print "Before sorting", array
array.sort()
print "After sorting", array
```

Tomado de: Python in High Performance Computing. Jussi Enkovaara.
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf

Procesamiento paralelo y multiprocesamiento

SMP

- dispy
- delegate
- forkmap
- forkfun
- ppmap
- POSH
- pp (parallel python)
- pprocess
- processing
- PyCSP
- **PyMP**
- Ray
- remoteD
- VecPy

Cluster Computing

- batchlib
- Celery
- Deap
- disco
- dispy
- DistributedPython
- exec_proxy
- execnet
- IPython
- job_stream
- jug
- **mip4py**
- Networkspaces
- PaPy
- papyros
- dask
- pp (Parallel Python)
- PyCOMPSs
- PyLinda
- pyMPI
- pypar
- pyPasSet
- pynpvm
- pypvm
- Pyro
- Ray
- rthread
- ScientifiPython
- SCOOP
- seppo
- Star-P for Python
- superpy

Tomado de: https://wiki.python.org/moin/ParallelProcessing

¡No use Python Puro!

- **Explore las APIS**
  - Numpy
  - Scipy
- **Escriba en**
  - C-Extensions
  - Cython
  - Numba
- **Guarde en Formatos Apropiados**
  - NetCDF
  - HDF5
- Use algoritmos ya implementados

```python
filenames = glob('2014-*-*.log')                # Collect all filenames

def process(filename):
    with open(filename) as f:                    # Load from file
        lines = f.readlines()

    output = ...                                 # do work

    with open(filename.replace('.log', '.out'), 'w') as f:
        for line in output:
            f.write(line)                        # Write to file

# [process(fn) for fn in filenames]             # Single-core processing

import multiprocessing
pool = multiprocessing.Pool()
pool.map(process, filenames)                     # Multi-core processing
```

Tomado de: http://matthewrocklin.com/slides/sf-python-parallelism.html

¡Procesos!

```python
filenames = glob('2014-*-*.log')                  # Collect all filenames

def process(filename):
    with open(filename) as f:                     # Load from file
        lines = f.readlines()

    output = ...                                  # do work

    with open(filename.replace('.log', '.out'), 'w') as f:
        for line in output:
            f.write(line)                         # Write to file

# [process(fn) for fn in filenames]               # Single-core processing

from concurrent.futures import ProcessPoolExecutor
executor = ProcessPoolExecutor()
executor.map(process, filenames)                  # Multi-core processing
```

Tomado de: http://matthewrocklin.com/slides/sf-python-parallelism.html

C-Extensions: Pasando un array Numpy a C

Python

```python
import myext

a = np.array(...)
myext.myfunc(a)
```

C: myext.c

```c
#include <Python.h>
#define NO_IMPORT_ARRAY
#include <numpy/arrayobject.h>

PyObject * my_C_func(PyObject *self, PyObject *args){
  PyArrayObject * a;
  if (!PyArg=ParseTuple(args, "O", &a))
    return null;
  ...
}
```

Tomado de: Python in High Performance Computing. Jussi Enkovaara.
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf

C-Extensions: Accediendo a los datos del arreglo

C: myext.c

```c
...
PyArrayObject* a;
int size = PyArray_Size(a);
double *data = (double *) a -> data;
for (int i = 0; i < size; i++){
  /* Process data */
}

Py_RETURN_NONE;
}
```

Tomado de: Python in High Performance Computing. Jussi Enkovaara.
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf

C-Extensions: Definiendo la interfaz con Python

C: myext.c

```
static PyMethodDef functions[] = {
  {"myfunc", my_C_func, METH_VARARGS, 0},
  {0, 0, 0, 0}
};

PyMODINIT_FUNC INITMYEX(void){
  (void) PyInitModule("myext", functions);
}
```

Construir como librería compartida

```
jdpinedac@atalaya
[07:29 AM]$|~> gcc -shared -o myext.so -l/usr/include/python2.6 -fPIC myext.c
```

Tomado de: Python in High Performance Computing. Jussi Enkovaara.
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf

C-Extensions: Uso desde Python

Python

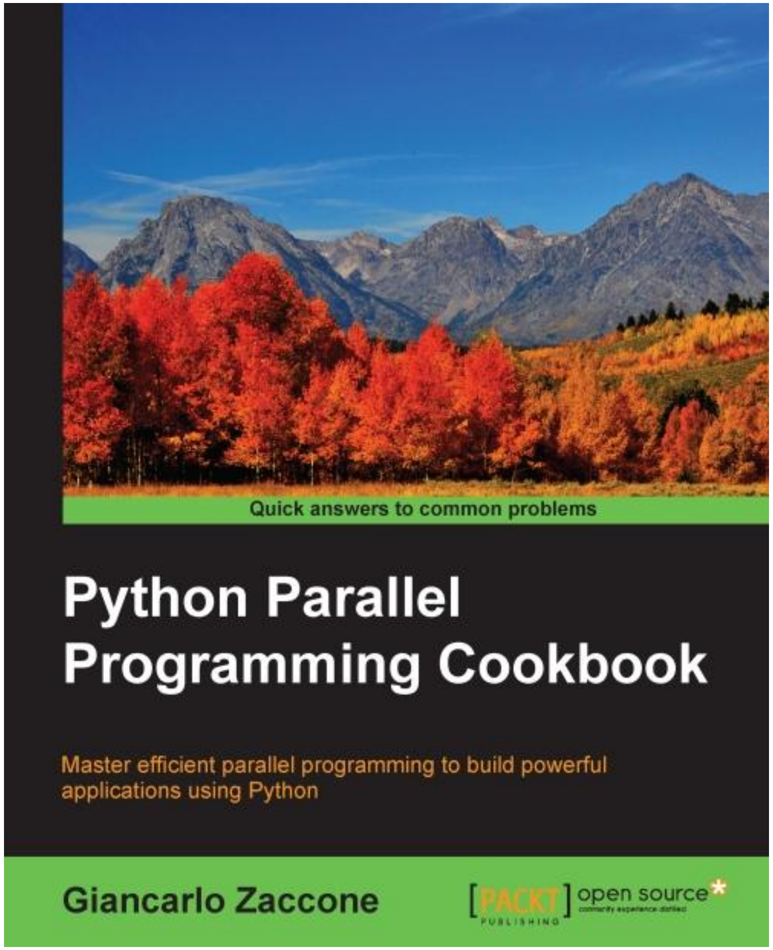
```python
import myext

a = np.array(...)
myext.myfunc(a)
```

Tomado de: Python in High Performance Computing. Jussi Enkovaara.
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf

Recursos

Este conjunto de diapositivas están basadas e inspiradas en los siguientes recursos:

- Python in High Performance Computing. Jussi Enkovaara. CSC - Tieteen tietotekniikan keskus Oy. CSC - IT Center for Science Ltd. http://www.training.prace-ri.eu/uploads/tx_pracetmo/pythonHPC.pdf
- Python Parallel Programming Cookbook. Master Efficient parallel programming to build powerful applications using Python. Giancarlo Zaccone. PACKT Publishing. OpenSource*. 2015.
- TACC - HPC Python - https://portal.tacc.utexas.edu/-/hpc-python
- Python and Parallelism - Matthew Rocklin. Continuum Analytics. http://matthewrocklin.com/slides/sf-python-parallelism.html
- Numba - https://numba.pydata.org/
- Introduction to Parallel Computing - https://computing.llnl.gov/tutorials/parallel_comp/
- MPI4py crash course. Ariel Lozano and David COlignon. CÉCI training. October 25, 2017. http://www.ceci-hpc.be/assets/training/mpi4py.pdf
- MPI for Python http://mpi4py.readthedocs.io/en/stable/
- Extending Python with C or C++ - https://docs.python.org/2/extending/extending.html



In Memoriam

```
01001010 01101111 01101000 01101110 00100000 01000101 01100100 01100111
01100001 01110010 00100000 01000011 01101111 01101110 01100111 01101111
01110100 01100101 00100000 01000011 01100001 01101100 01101100 01100101
00001010 00101000 00110001 00111001 00111000 00110010 00100000 00101101
00100000 00110010 00110000 00110001 00111000 00101001
```