

# ICMAT

INSTITUTO DE CIENCIAS MATEMÁTICAS

## INTRODUCCIÓN A LA PROGRAMACIÓN PARALELA Y COMPUTACIÓN EN CLUSTER

UAM+CSIC

# PROGRAMA

- Proyectos de código paralelo
- Librerías y herramientas
- Lenguajes, compiladores y entornos de desarrollo
- Comunicación y transmisión de datos entre procesos
- El sistema de archivos Lustre

# Proyectos de código paralelo

1. **Decisiones de diseño**
2. Entornos de desarrollo y ejecución
3. Organización de archivos y directorios
4. Copias de seguridad
5. Sistemas de construcción, compilación y control de versiones

## Proyectos de código paralelo/Diseño

- Diseño de código en paralelo => descomponer los problemas para su ejecución simultánea en bloques:
  - Lo más **independientes** entre sí
  - Que necesiten **transmitir la mínima información** entre ellos
  - Que no dependan de los mismos **recursos**
    - Núcleos de CPU
    - Posiciones de memoria
    - Mismas zonas físicas en los ficheros

## Proyectos de código paralelo/Diseño

- **Tipos de paralelismo**

- **Multihilo:** un único proceso a nivel lógico, con una región de memoria que se comparte entre los varios *hilos* de ejecución. Los hilos pueden acceder directamente a la región de memoria. Todos los hilos deben pertenecer a la misma CPU.
- **Multiproceso:** varios procesos, con regiones de memoria independiente. Deben disponer de mecanismos de comunicación para transmitir información. Pueden residir en distintas CPUs.

## Proyectos de código paralelo/Diseño

- Se suelen clasificar en **tres** tipos de diseños:
  - **“Embarazosamente paralelos”** => Apenas hay dependencias o transmisión de información entre los distintos subprocesos. No es necesaria coordinación entre ellos o es muy ligera.
  - **Paralelismo de “grano grueso”** => Hay poca transferencia de información entre los procesos, pero existe algún tipo de coordinación, establecimiento de condiciones iniciales, control de concurrencia, etc.
  - **Paralelismo de “grano fino”** => Hay una necesidad de coordinación grande, se transmiten cantidades significativas de información entre los procesos y/o necesitan un alto grado de coordinación entre ellos.

## EJEMPLOS

- Embarazosamente paralelos:
  - Simulaciones, fórmulas que actúan de forma independiente sobre una matriz de puntos que se pueden dividir en regiones.
  - Cálculo de trayectorias independientes

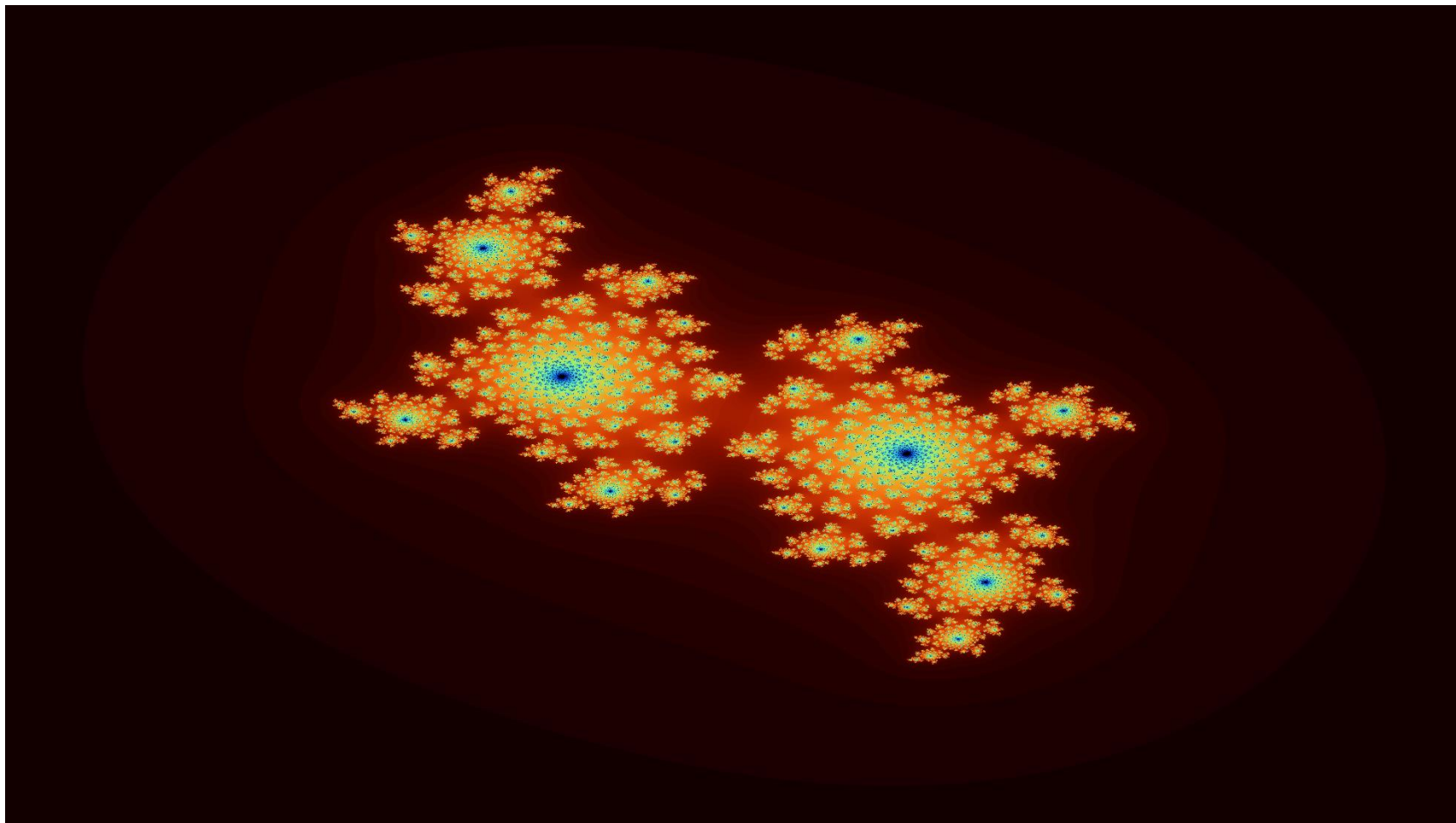
## Proyectos de código paralelo/Diseño

- Algunos aspectos de diseño sobre los que pensar:
  - **¿Cómo se ejecutarán los procesos?**
    - En la **misma CPU**, diferentes núcleos (compartirán CPU y caché)
    - En la **misma máquina**, diferentes CPUs (compartirán memoria)
    - En **diferentes máquinas**, diferentes CPUs (compartirán red y acceso a disco)
  - **¿Cómo se comunicarán los procesos entre sí?**
    - Condiciones iniciales (parámetros de inicio)
    - Estados intermedios (estructuras de datos comunes)
    - Resultados del procesamiento (lo que producen realmente)

## Proyectos de código paralelo/Diseño (cont.)

- **¿Cómo se coordinarán los procesos?**
  - Mediante uno de ellos que actúa como **coordinador**
  - De forma totalmente **distribuida**
- **¿Cómo se comunicarán los procesos entre sí?**
  - Estructuras compartidas de datos (OpenMP)
  - Paso de mensajes (MPI)
  - A través de ficheros (testigos, buzones, etc.)
  - A través de mecanismos de red (sockets, etc.)
  - A través de otros sistemas proporcionados por el sistema operativo: semáforos, tuberías, colas de mensajes, segmentos de memoria compartida, etc.

## EJEMPLO: Fractal de Julia



## Proyectos de código paralelo/Diseño (cont.)

### Algoritmo:

- Para cada coordenada  $(x,y)$  de la imagen asociamos un número imaginario  $z$  cuya parte real corresponde con el eje de abscisas y la parte imaginaria corresponde con el eje de ordenadas  $p(x,y) \rightarrow z(\text{re},\text{im})$ .
- Calculamos  $z \leftarrow z*z + 0.65i$  hasta que:
  - $|z| < 10$  ó
  - El número de iteraciones sucesivas sea mayor que 255
- Asociamos a esa coordenada el número de iteraciones necesarias para salir del bucle.
- Ese número de iteraciones indicará el color del punto de la imagen.

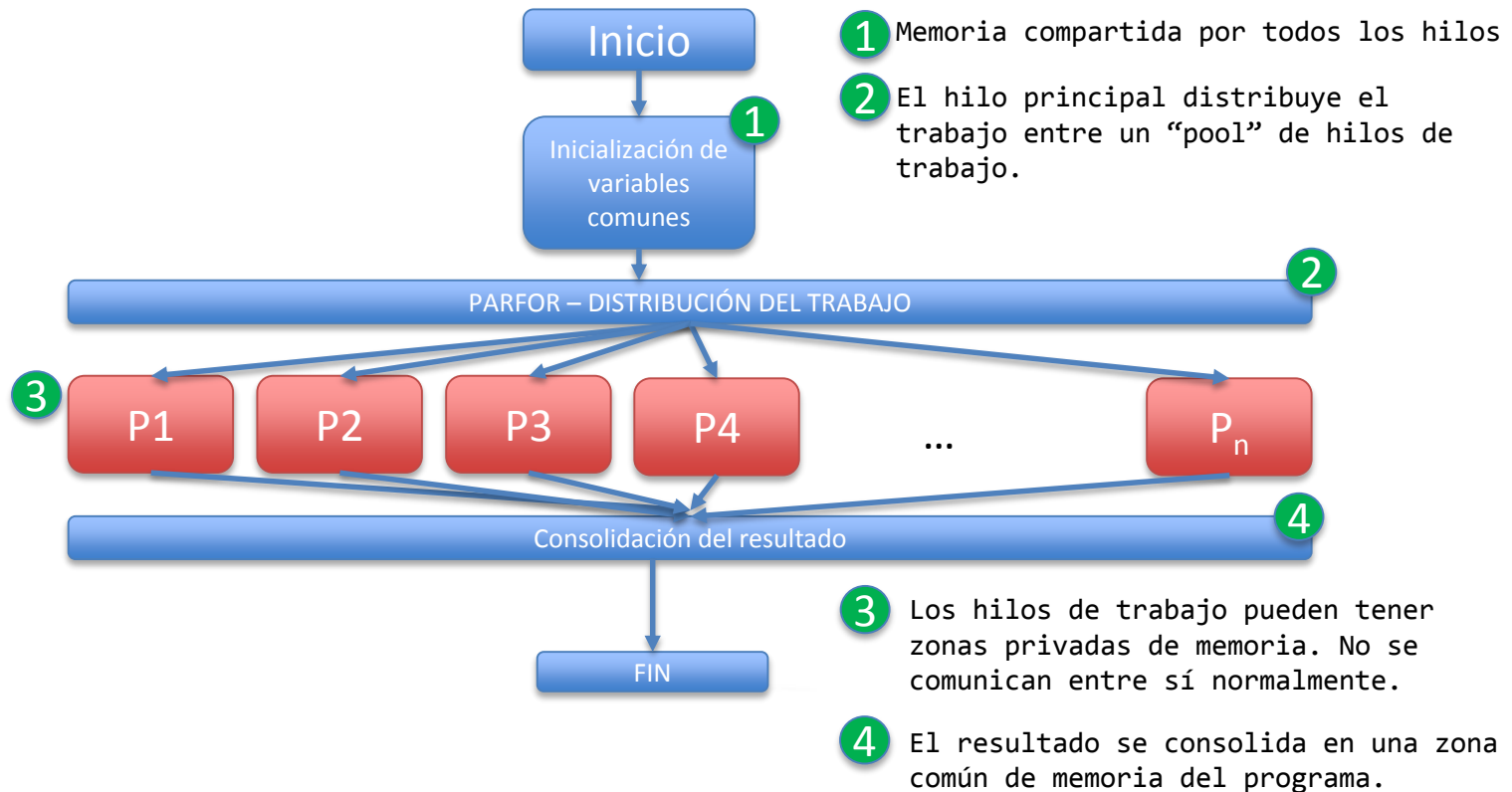
## Proyectos de código paralelo/Diseño (cont.)

### Opción 1: paralelismo de grano grueso (parfor)

- Todos los subprocesos trabajan sobre la misma región de memoria (donde escriben los resultados)
- Son dependientes entre sí, comparten un recurso (una región de memoria común).
- No requieren programación especial para la coordinación, pero deben hacerlo para escribir en la misma región de la memoria.
- Se ejecutan dentro de la misma máquina (hasta 16 hilos o el máximo de cores que establezcamos)
- Corresponde al equivalente en MATLAB a “**parallel for**”.
- Más sencillo de programar, pero menos efectivo (la escritura sobre las zonas de memoria comunes hace de cuello de botella).
- Normalmente limitado al número de núcleos con acceso a la misma región de memoria, o que puedan comunicarse de forma eficiente => el coste de transmisión de datos entre procesos + el coste de generación de procesos < beneficio ganado del paralelismo.

# Proyectos de código paralelo/Diseño (cont.)

## Opción 1: paralelismo de grano grueso (parfor - threads o hilos)



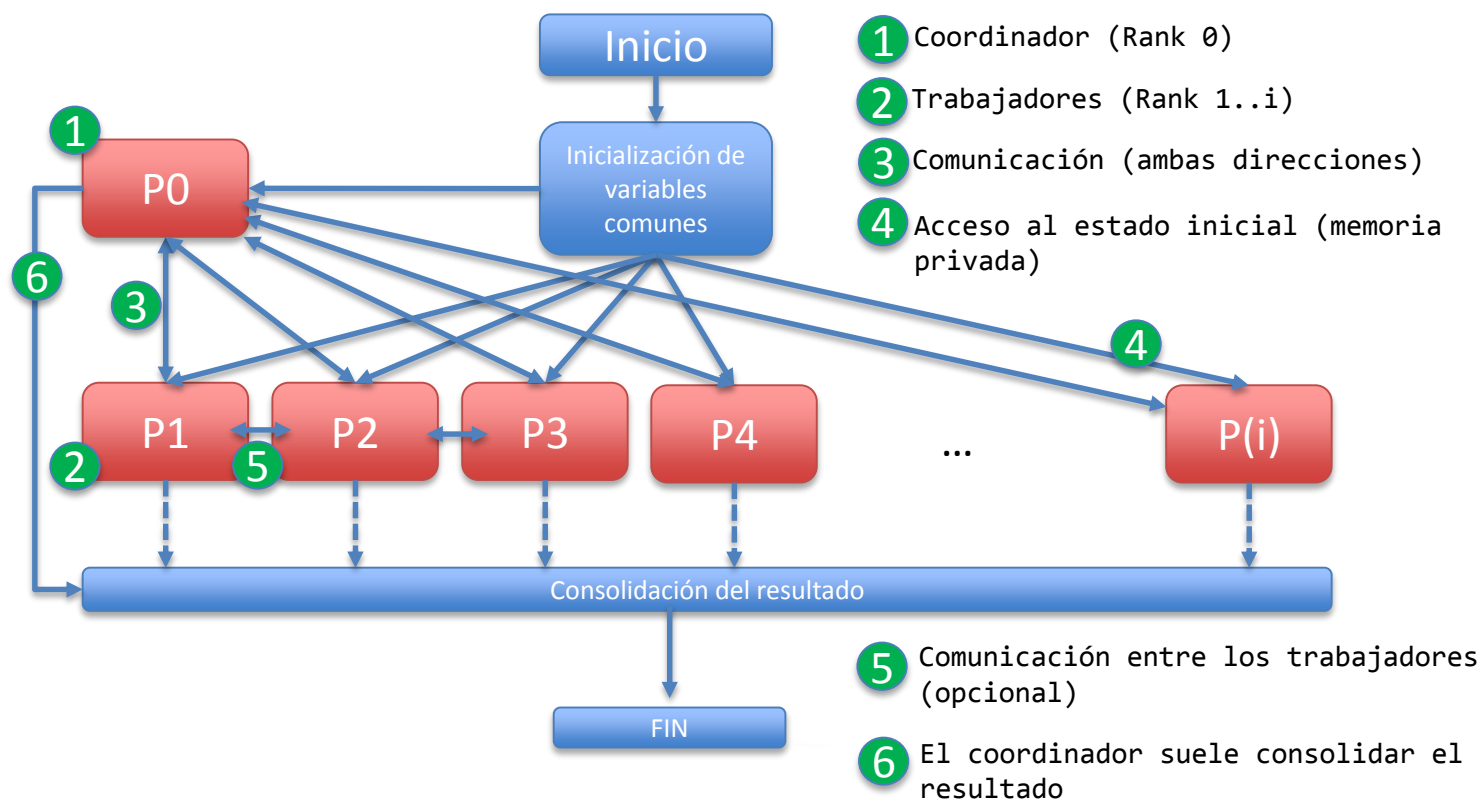
## Proyectos de código paralelo/Diseño (cont.)

### Opción 2: paralelismo de grano fino (MPI)

- Todos los subprocesos trabajan sobre su propia región de memoria. Para compartir información, deben de enviarse mensajes entre sí.
- Son independientes entre sí, pueden ejecutarse en máquinas distintas o en la misma máquina, con la misma o diferente arquitectura, sistema operativo, etc.
- Requieren programación especial para la coordinación, bien mediante uno de ellos que actúa como coordinador o en algoritmos totalmente distribuidos.
- Se ejecutan dentro de la misma máquina u otras.
- Menos sencillo de programar, pero más efectivo al ser procesos independientes. Más flexible en cuanto al número de procesos y su ubicación en ejecución.
- El paralelismo normalmente está limitado a que puedan comunicarse de forma eficiente => el coste de transmisión de datos entre procesos + el coste de generación de procesos < beneficio ganado del paralelismo.
- El planificador del cluster tiene opciones para indicar la afinidad de los supprocesos (a la CPU, al socket, al servidor, ...)

# Proyectos de código paralelo/Diseño (cont.)

## Opción 2: paralelismo de grano fino (paso de mensajes - MPI)



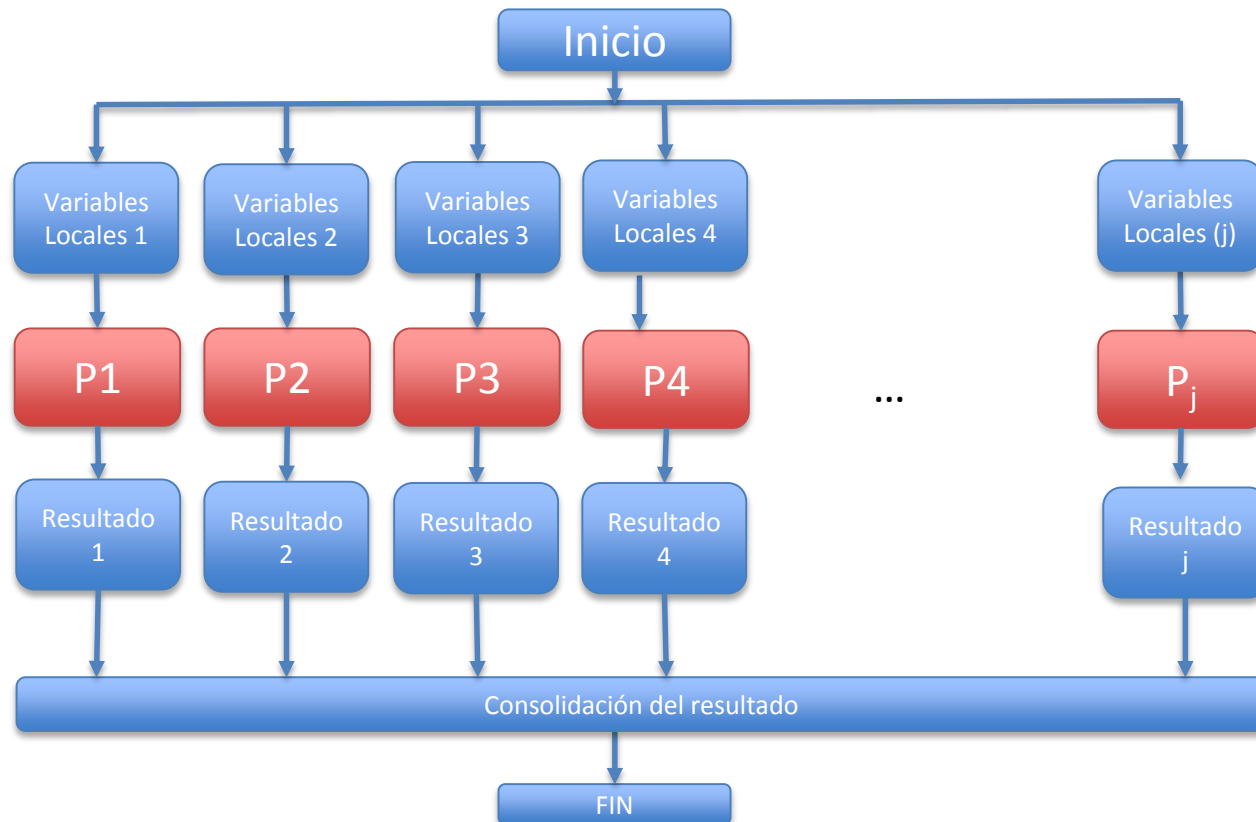
## Proyectos de código paralelo/Diseño (cont.)

### Opción 3: embarazosamente paralelos (“embarrassingly parallel”)

- Todos los subprocesos son totalmente independientes, no se comunican entre sí.
- No requieren programación especial para la coordinación, únicamente en algún caso para consolidar los resultados.
- Es el más sencillo de programar, y el más efectivo al ser procesos independientes.
- El paralelismo normalmente está limitado únicamente a la disponibilidad de recursos.
- No todos los problemas son susceptibles de este esquema.

## Proyectos de código paralelo/Diseño (cont.)

### Opción 3: embarazosamente paralelos



## Proyectos de código paralelo

1. Decisiones de diseño
2. **Entornos de desarrollo y ejecución**
3. Organización de archivos y directorios
4. Copias de seguridad
5. Sistemas de construcción, compilación y control de versiones

# Proyectos de código paralelo/entornos

## Recomendaciones

- Utilizar un [entorno de desarrollo](#) en local (PC), con una configuración similar (arquitectura, software, librerías, sistema operativo, etc.) o al menos compatible.
- Si no disponéis de uno, solicitadlo, es posible utilizar algunos entornos disponibles ([Everest.icmat.es](#))
- Disponibilidad de entorno gráfico, herramientas de desarrollo (editores, compiladores, depuradores)
- Herramientas de gestión de versiones, de sincronización de código o de acceso remoto a ficheros (GIT, CVS, rsync, SSHFS, SFTP, etc.)

## EJEMPLO

|                            | C/C++/FORTRAN                                                                                                            | PYTHON                                             | R                        | MATLAB          |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|--------------------------|-----------------|
| EDITOR/IDE                 | <a href="#">Visual Studio Code</a><br><a href="#">Atom</a><br><a href="#">Sublime Text</a><br><a href="#">Qt Creator</a> | <a href="#">Ipython</a><br><a href="#">Jupyter</a> | <a href="#">R studio</a> | Matlab          |
| Compiladores / intérpretes | <a href="#">GCC (6.3.0)</a><br><a href="#">Intel compiler</a>                                                            | <a href="#">Anaconda</a>                           | R                        | Matlab compiler |
| Librerías esenciales       | OMP, OpenMPI, Eigen Boost                                                                                                | Pillow, Numpy, ScyPy, mpi4py                       | Ggplot2 parallel         | (*)             |

(\*) Matlab tiene un soporte pobre para la ejecución de código paralelo en clusters. Requiere además la instalación de productos adicionales para gestionar los diferentes procesos, con un coste adicional. Veremos alguna técnica para superar alguna de estas limitaciones.

## EJEMPLO – CÓDIGO PARALELO

- 1) Paralelismo multihilo Julia set (OMP) – C++
- 2) Multiprogramación: Mandelbrot set (MPI) – Python
- 3) Multiprogramación: MATLAB y procesos en cluster

## Proyectos de código paralelo

1. Decisiones de diseño
2. Entornos de desarrollo y ejecución
3. **Organización de archivos y directorios**
4. Copias de seguridad
5. Sistemas de construcción, compilación y control de versiones

## Proyectos de código paralelo/organización

### Recomendaciones

- Decidir por un método de organización y mantenerlo en la medida de lo posible.
- Ahorra tiempo, evita problemas (ej. Borrado accidental)
- Algunas ideas:
  - Un directorio por proyecto/programa.
  - Dentro de éste:
    - `/bin`: para los binarios y scripts directamente ejecutables
    - `/src`: para los ficheros fuente y directorios intermedios de construcción.
    - `/data`: para los ficheros de datos, tanto de entrada como de salida
    - `/etc`: para los ficheros de configuración
- Éste modelo se integra bien con sistemas de gestión de versiones (`GIT`, SVN, CVS)
- Permite la sincronización remota mediante `rsync`

## Proyectos de código paralelo

1. Decisiones de diseño
2. Entornos de desarrollo y ejecución
3. Organización de archivos y directorios
4. **Copias de seguridad**
5. Sistemas de construcción, compilación y control de versiones

# Proyectos de código paralelo/copias de seguridad

## Recomendaciones

- Disponer del código en más de una ubicación
- Realizar copias o sincronizaciones automáticas del código (`rsync`, `tar`, `sftp`, `scp`)
- Programar procesos mediante `crontab`:
  - Generar un script que haga las copias
  - Programarlo mediante `crontab -e`
  - Revisar el log y las copias de vez en cuando

## EJEMPLO – RSYNC programado mediante CRONTAB

## Proyectos de código paralelo

1. Decisiones de diseño
2. Entornos de desarrollo y ejecución
3. Organización de archivos y directorios
4. Copias de seguridad
5. **Sistemas de construcción, compilación y control de versiones**

# Proyectos de código paralelo/control de versiones

## Recomendaciones

- Son más útiles cuando tenemos más de una de estas condiciones:
  - Proyectos complejos, con varios componentes desarrollados de forma independiente (ejecutables, librerías) pero dependientes entre sí.
  - Gran número de ficheros fuente (50+).
  - Desarrollados por más de una persona en simultáneo.
  - Desarrollados en más de una ubicación.
- Herramientas:
  - Git: <https://git-scm.com/>
  - Subversion: <https://subversion.apache.org/>
  - CVS: <http://www.nongnu.org/cvs/>
- Ventajas:
  - Más de una copia de los fuentes
  - Consolidación de cambios entre varios orígenes
  - Posibilidad de revertir cambios y de hacer variantes del código (versiones/ramas).

# Proyectos de código paralelo/herramientas de construcción de código y compilación

## Recomendaciones

- Son más útiles cuando tenemos más de una de estas condiciones:
  - Proyectos complejos, con varios componentes desarrollados de forma independiente (ejecutables, librerías) pero dependientes entre sí.
  - Gran número de ficheros fuente (50+).
  - Necesidad de compilación en varios entornos diferentes.
  - Dependencias con varias librerías y versiones.
- Herramientas:
  - CMAKE: <https://cmake.org/>
  - Unix make
- Ventajas:
  - Ahorro en tiempos de construcción: sólo se compilan los cambios
  - Facilidad de uso: definir una vez y ejecutar varias veces
  - No sólo para la construcción: también para el despliegue, instalación, desinstalación, sincronización, copia remota ...

# Librerías y herramientas de programación

1. **Compiladores y entornos de ejecución**
2. Depuradores y análisis de rendimiento

# Librerías y herramientas de programación/compiladores

```
[acaso@master ~]$ module avail
----- /usr/share/Modules/modulefiles -----
dot          module-git  module-info modules      null          use.own

----- /LUSTRE/apps/Modules/ -----
Anaconda          matlab/2010b
cdo                 matlab/2012b
cmake/3.7.1        matlab/2014a
compilers/gcc/4.4.7  matlab/2014b
compilers/gcc/6.3.0  matlab/2015a
compilers/gcc/binutils/2.28 matlab/2016a
compilers/intel/15.0.3  matlab/2016b
compilers/nvidia/7.5  mpi/intel
gnuplot             mpi/mpich-x86_64
Julia               mpi/openmpi
libs/Boost          openfoam-4.1
libs/nag            openfoam-v1612+
Mathematica
```

# Librerías y herramientas de programación/compiladores

## PYTHON: Componentes

- **Anaconda**: distribución de Python que incluye las librerías más frecuentes y orientadas al cálculo y uso científico.
  - `module load Anaconda` -> prepara el entorno de Python
- **Ipython** -> Intérprete interactivo en modo consola:
  - Autocompletar
  - Ayuda en línea
  - Ejecución paso a paso
- **ipyparallel** -> librería de ejecución en paralelo y en cluster: permite paralelizar trabajos desde Python que se ejecuten en un número de nodos. (WIP), así como la ejecución en varios hilos.
- **Jupyter notebooks** -> Permite la ejecución visual desde un navegador de los comandos lanzados en Python. También permite la paralelización mediante **ipyparallel**.

# Librerías y herramientas de programación/compiladores

## PYTHON: Configuración del entorno en Linux/Mac/Windows

- Opción A) Instalación desde un “bundle” como Anaconda:  
<https://www.continuum.io/downloads> (Windows, Mac, Linux)
- Opción B) Instalación manual de paquetes y librerías

En Ubuntu:

```
$ sudo apt-get install python[3]
$ sudo apt-get install jupyter-notebook
$ sudo apt-get install python[3]-pip
$ sudo apt-get install python[3]-numpy
$ sudo apt-get install python[3]-scipy
$ sudo apt-get install python[3]-mpi4py
$ sudo apt-get install python[3]-matplotlib
```

# Librerías y herramientas de programación/compiladores

## PYTHON: Ejecución de programas con paralelismo en PYTHON

- Python no soporta de forma sencilla los hilos de ejecución. Algunas librerías como `NumPy` permite la ejecución de operaciones de álgebra lineal utilizando paralelismo de forma transparente.
- `Mpi4py` permite la utilización del estándar de MPI para la comunicación de procesos en la misma o diferentes máquinas.
- También es posible la aproximación de la ejecución de múltiples procesos independientes en Python sin comunicación mediante subtarefas.

# Librerías y herramientas de programación/compiladores

## MATLAB: Componentes

- La versión 2010b es completa y consta del intérprete, con interfaz gráfico opcional, los toolboxes licenciados y el compilador.
- Del resto, se encuentra instalado el “runtime” únicamente, que permite la ejecución de programas compilados en otros entornos.
- “Distributed Computing Toolbox” está licenciado, pero MATLAB no permite la ejecución de procesos en distintos nodos sin una licencia adicional, y está limitado a 8 núcleos por programa.
- Puede superarse esta limitación:
  - Dividiendo el problema en subproblemas
  - Compilando el programa en Matlab para no tener problemas de licencia
  - Utilizar el runtime y el planificador del cluster para lanzar n procesos con un parámetro que los diferencie, u otro mecanismo de coordinación (ficheros de sincronización, variables de entorno, ...)

# Librerías y herramientas de programación/compiladores

## MATLAB: Subprocesos

- Desarrollar una función en MATLAB que reciba una serie de parámetros, preferentemente un entero que indique al subproceso la parte del trabajo que le toca realizar.
- Compilamos la función:  
\$ gcc -m func.m -o func
- Creamos un trabajo con subtareas para lanzar con qsub

```
#!/bin/bash
#$ -q all.q
#$ -l h_vmem=1G
#$ -j y
#$ -t 1-20
#$ -tc 5
#$ -cwd
./func $SGE_TASK_ID
```

# Librerías y herramientas de programación

1. Compiladores y entornos de ejecución
2. Depuradores y análisis de rendimiento

## Librerías y herramientas de programación/herramientas de depuración y análisis

- Fundamentalmente, para lenguajes compilados (C, C++, FORTRAN)
- En GCC tenemos GDB (ejecución paso a paso) y GPROF (profiling)
- Permiten, el primero, inspección de variables y ejecución paso a paso y el segundo, análisis de rendimiento.
- En los lenguajes interpretados (MATLAB, R, PYTHON), tenemos herramientas similares para la depuración.
- La depuración y análisis de rendimiento conviene realizarla sobre las versiones serie, y una vez libres de errores y optimizadas, sobre las versiones en paralelo sólo si es necesario.

# Transmisión de datos entre nodos: Altas prestaciones para proceso de información distribuido

1. La red Infiniband e IpoIB
2. Como utilizar la red Ib para la comunicación entre procesos

## Transmisión de datos entre nodos/red IB

- La red **Infiniband** es una red alternativa a Ethernet que ofrece **baja latencia** y **altas tasas de transferencia**.
- Permite velocidades más altas de transmisión con **bajo consumo de CPU** evitando el acceso al kernel.
- Dispone de una API, pero es complicada de programar. Salvo necesidades muy específicas, es mejor utilizar utilidades y APIs que ofrecen mayor facilidad de uso.
- Se utiliza la red Infiniband de forma sencilla y transparente:
- Al acceder al sistema de archivos **LUSTRE** (a través de LNET)
- Al comunicarse con otros procesos locales o remotos a través de **OpenMPI (Send, Recv)**
- Al utilizar cualquiera de los servicios de IP over Infiniband

## Transmisión de datos entre nodos/red IB

.Ejemplo:

.Prueba de transmisión de datos entre nodos

.Servicios de IpoIB: la red IP interna del cluster

# El sistema de archivos Lustre

1. Características del sistema de archivos Lustre
2. Comandos de gestión del sistema de archivos

## El sistema de archivos Lustre/Características

- \* Ofrece una alta disponibilidad y redundancia de los datos (RAID-6)
- \* Ofrece baja latencia en los accesos, paralelización en los accesos y altas tasas de transferencia
- \* Ofrece un sistema de caché para asegurar el máximo uso de los datos en memoria
- \* Permite operaciones avanzadas como *\*STRIPING\** de ficheros para aumentar el rendimiento
- \* Es altamente escalable y puede alcanzar Petabytes o Exabytes de tamaño de ficheros
- \* Se encuentra montado en todos los nodos en modo de lectura/escritura en la ruta **/LUSTRE**
- \* **/LUSTRE/users** contiene los \$HOME de los usuarios.
- \* Cualquier cambio en cualquiera de los nodos es directamente observable desde cualquier otro nodo.
- \* Dispone de 4 servidores OSS, permitiendo escrituras en paralelo de entre 1 y 4 nodos (“stripe” 1 a 4).
- \* La comunicación y transmisión de datos se realiza de forma transparente a través de Infiniband.
- \* Documentación: <http://lustre.org/documentation/>
- \* Cumple es estándar POSIX, todos los comandos y funciones estándar de UNIX aplican de forma transparente a LUSTRE.

# El sistema de archivos Lustre/Comandos

## # LUSTRE en LOVELACE

- \* Se forma por dos servidores de metadatos (MDS) en alta disponibilidad y 4 servidores de datos (OSS) balanceados.
- \* Funciona de forma transparente al sistema operativo y a las aplicaciones, cuando se trabaja en el sistema de ficheros (/LUSTRE)
- \* Dispone de una línea de comandos para operaciones especiales:

```
$ lfs help
```

## El sistema de archivos Lustre/Comandos

```
$ lfs df -> Visualizar la ocupación de los OSTs.
$ lfs osts -> Obtener información de los OSTs.
$ lfs find -> Buscar ficheros de forma más eficiente que GNU find.
$ lfs quota <fs> -> Obtener información del uso de la cuota asignada.
$ lfs getstripe <dir> -> Obtener información del "stripe" de un archivo.
$ lfs rgetfacl <file> -> Obtener información acerca de las listas de acceso.
$ lfs setstripe <dir|fs> -> Establecer el nivel de stripe de un fichero.
$ lfs getstripe <dir|fs> -> Obtener el nivel de stripe de un fichero.
```



C/ Nicolás Cabrera, nº 13-15  
Campus Cantoblanco UAM  
28049 Madrid, Spain

[www.icmat.es](http://www.icmat.es)



**ICMAT**  
INSTITUTO DE CIENCIAS MATEMÁTICAS