Curso 2022-2023

Práctica 0: Entorno de Trabajo - Configuración Cuentas

13/09/2022
Prácticas de Programación Concurrente y Paralela
Pablo Revuelta & José Ranilla

Tipo de Práctica: Cerrada[1], no cuantificable numéricamente
Entrega Documentación: No
Duración: 1ª Sesión de Prácticas (aproximadamente 2 horas)

[1] **Prácticas Cerradas** son las que no requieren diseñar y/o implementar algoritmos, solamente hay que seguir las instrucciones dadas. El alumno deberá responder a las preguntas formuladas o subir determinadas respuestas al Campus Virtual.

ÍNDICE

Objetivo

Presentar la arquitectura hardware/software de la sala de prácticas, las herramientas básicas, los modos de trabajo y la "*summary card*" básica de ***C*** y ***Linux***. Configurar las cuentas para trabajar con OGE/SEG (*Open Grid Scheduler* / *Sun Grid Engine*), tanto en secuencial como en paralelo. Ejecutar ejemplos de casos de uso con el gestor y explicar el convenio adoptado para la interfaz de *drivers* en lenguaje C para usar desde Python.

Virtual Private Networks

A mediados de 2019 la Universidad de Oviedo implementó nuevas políticas de acceso a sus recursos computacionales (y otros) desde fuera de la Red Corporativa de la Universidad de Oviedo (desde el exterior; p. ej. desde casa), con el objetivo de mejorar la seguridad y estabilidad de los sistemas. Actualmente, el acceso a dichos recursos desde fuera solo es posible usando **VPNs** (*Virtual Private Networks*).

En https://sic.uniovi.es/redes, sección "Acceso Remoto", se explica cómo instalar y configurar los programas que facilitan el acceso desde el exterior (la WEB cambia periódicamente. Si la URL anterior no funciona se recomienda acceder a www.uniovi.es y en el campo de búsqueda escribir VPN). De forma resumida:

- Ir a https://sic.uniovi.es/redes
- En el *Menú* seleccionar "Acceso Remoto"
- Dentro de "Acceso Remoto", en "Enlaces relacionados", se explica cómo configurar la VPN, así como el enlace al "Portal de Software Corporativo de la Universidad de Oviedo".
- Ir al "Portal de Software Corporativo de la Universidad de Oviedo" y entrar en "Software para Terceros". Seguir las instrucciones.

Los recursos informáticos usados en las prácticas de PCP SOLO son accesibles desde fuera de la universidad usando la VPN de la Universidad de Oviedo. Es responsabilidad del alumno configurar sus equipos personales.

Arquitectura hardware de la Sala

El hardware de la sala se ha renovado parcialmente este curso. La Figura 1 muestra la distribución de los distintos elementos computacionales de la sala de prácticas. Estos son:

- **Equipos Interactivos** (cajas azules en Figura 1). Conjunto de 18 equipos iguales con 6/8 GB de RAM, 480 GB de disco SSD y procesador Intel I3 (*dual core*, sin *hyperthreading*, 2 cores) para la realización de tareas interactivas (*login*, navegación, edición, ejecuciones de prueba, etc.).

  Cada uno de ellos tiene una etiqueta en su parte frontal con su nombre lógico (p. ej. las etiquetas de los 2 ordenadores de la primera mesa a la derecha –visto desde la puerta– son di120.edv.uniovi.es y di121.edv.uniovi.es, respectivamente. La enumeración es consecutiva por lo que los ordenadores de la última mesa a la izquierda son di136.edv.uniovi.es y di137.edv.uniovi.es, respectivamente.)

La asignación de direcciones *IP* es estática para ayudar a detectar problemas (p. ej. cuando se ejecuten programas en memoria distribuida). El sufijo de la dirección IP se obtiene con la parte numérica de la raíz del nombre lógico (p. ej. 156.35.160.120 es la IP de di**120**.edv.uniovi.es)

Estos equipos tienen acceso pleno al exterior. Desde el exterior se puede acceder a ellos usando la VPN de la universidad. Salvo excepcionalidades derivadas de las exigencias del curso **solo estarán encendidos durante el horario de prácticas**.

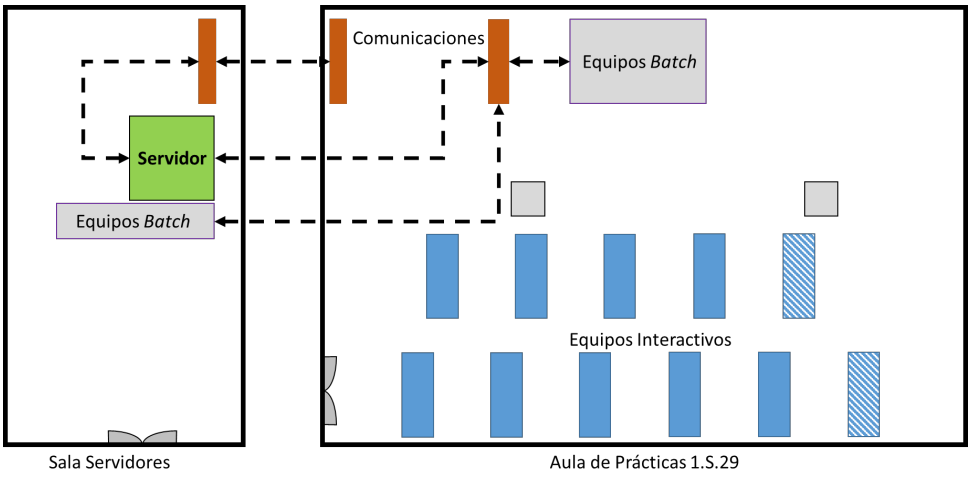

Figura 1. Planos

- **Equipos Batch** (caja de color gris en Figura 1). Conjunto de equipos para la ejecución de trabajos en producción (*jobs* o tareas *batch*). **Solo son accesibles desde el Gestor de Colas OGE/SGE**.

  Parte de los equipos de este mini-clúster estará siempre disponible (24 horas al día, todos los días del curso) salvo causas de fuerza mayor (avería en el servidor, fallo de red eléctrica o de datos, etc.) La herramienta *EECluster* gestiona automáticamente su encendido / apagado según la carga de OGE/SGE, es decir, según las necesidades de los alumnos. Los equipos no gestionables de forma automática (los que tienen las GPUs) se dejarán encendidos cuando sea necesario.

- **Servidor** (caja de color verde en Figura 1). Nombre lógico di119.edv.uniovi.es. Máquina virtual alojada en los servidores corporativos del Dpto. de Informática. Soporta todo el software *corporativo* y de gestión, además del *home* de los usuarios. Siempre encendido (durante el curso académico) y accesible desde todos los nodos de la sala y desde el exterior vía VPN. **No se hace copia de seguridad del *home*** de los usuarios (del resto sí).

  Se puede usar para desarrollo y pruebas puntuales de escasa duración (p. ej. cuando el alumno se conecta desde el exterior) pero nunca para la ejecución en producción por varios motivos como, por ejemplo, porque las mediciones serán erróneas o porque se saturará y afectará al resto de la sala. Si se detecta la ejecución de prácticas en el servidor serán abortadas y la cuenta posiblemente cancelada.

Arquitectura software de la Sala

Todos los equipos tienen SO CentOS 7.9.x, compiladores GCC 4.8.x y 7.3.x, compilador Intel 2021.5.x, Blas, Lapack, MKL, CUDA 11.7.x, Python 3.8.x, etc. Los **Equipos Interactivos** arrancan en modo gráfico (*gnome*) y los **Batch** en modo texto.

Además del software anterior se usa:

- **EECluster** Para el control dinámico y activo de encendido / apagado de los equipos (herramienta propia, por si falla).
- **NFS** Para la exportación desde el servidor del *home* de los alumnos.
- **NIS** Para el control de las cuentas desde el servidor.
- **OGE/SGE** Sistema de colas (planificador de trabajos o gestor de recursos): la forma más eficiente de maximizar la utilización y minimizar las esperas. Garantiza condiciones de exclusividad: un solo proceso de usuario en ejecución por nodo en cada instante. La versión instalada es la versión 8.1.9 (fecha 2016-03-02) de proyecto "*Son of Grid Engine*".

  Los usuarios crean trabajos *batch* (*jobs*) y los envían al gestor. Un *job* es una ejecución concreta de un contexto con una serie de atributos (nodos, tiempo de ejecución, etc.). El gestor lanza la ejecución de los *jobs* cuando los recursos solicitados estén disponibles. Cuando la ejecución concluye el gestor recupera la salida en el *home* de usuario. **Una vez enviado el *job* el usuario puede, si lo desea, cerrar la sesión**.

  Dado que los equipos donde finalmente se ejecutan los *jobs* lo decide el gestor, dentro del abanico indicado por el usuario, un trabajo *batch* **no suele** estar formando por un ejecutable; está formado por los ficheros con el código fuente, un fichero *makefile* y un *script* con órdenes de ejecución y para OGE/SGE.

  Hay un entorno gráfico para el envío/control de los *jobs*, pero es relativamente complejo. Es recomendable hacerlo desde una terminal usando los comandos básicos (**qdel**, **qsub**, **qstat** y **qhost**[2]). Para ayudar al alumno en el directorio /opt/Ejemplos_SGE/ hay ejemplos de uso para cada modalidad. Durante esta práctica se usarán todos los ejemplos.

Arranque / Apagado de los equipos

Como se ha comentado, el servidor está, debe estar, siempre encendido y los nodos del mini-clúster se encienden / apagan automáticamente bajo demanda (a excepción de los nodos con GPU). Por su parte los nodos interactivos estarán apagados, salvo excepciones, fuera del horario de prácticas, como ya se ha comentado.

El arranque de los equipos (interactivos y batch) es relativamente lento debido a que:

- **Boot** Proceso clásico de inicio (BIOS, verificar memoria, etc.)
- **Retardo eth0** Arrancando en Linux hay un retardo en la detección y configuración del dispositivo *eth0* (la red) con los nodos interactivos.

[2] Todos los comandos de SGE/OGE empiezan por la letra *q* y son escasos los comandos Linux que empiezan por la citada letra. Por tanto, no es necesario recordarlos, en un terminal después de escribir la letra *q* se presiona la tecla "tabulador" un par de veces y aparece la lista.

Modos de trabajo

El alumno puede interactuar con los recursos de la sala de 2 modos:

- **Presencial** Como es habitual en las prácticas de la titulación, usando cualquiera de los equipos interactivos.
- **Remoto** Accediendo al servidor usando el protocolo *ssh* (**ssh di119.edv.uniovi.es**) usando el puerto 2222. Si el equipo remoto no está dentro de la Red Corporativa de la Universidad de Oviedo se debe iniciar, previamente, la VPN como se ha comentado en la sección "*Virtual Private Networks*" de este documento. El funcionamiento es similar al modo presencial.

¿Qué debe hacer el alumno? (Fase 1ª)

1. Cambiar la *password*, usando los comandos apropiados de NIS/YP. Se explica en el aula el cómo y el porqué.

Pruebas del correcto funcionamiento de los compiladores. En una terminal teclear:

2. Para el compilador Intel
   source /opt/intel/oneapi/setvars.sh
   icc –v
   (la salida debe contener "… Version 2021.5.0 …")
3. Para el compilador Gcc 7.3.x.
   bash
   source /opt/rh/devtoolset-7/enable
   gcc -v
   exit
   (la salida debe contener "… gcc version 7.3.1 …")
4. Prueba el compilador de CUDA 11.7.x.
   nvcc –V
   (la salida debe contener "… cuda_11.7 …")
5. Para el intérprete de Python
   python –V
   (la salida debe contener "… Python 3.8. …")

Importante. Cuando se activa el entorno de desarrollo de Intel OneAPI (ejecución del comando "source /opt/intel/oneapi/setvars.sh") también se cambia el intérprete de Python, que deja de ser "RedHat 3.8.x" para convertirse en "Python 3.9.7: Intel Corporation".
El alumno debe recordar este cambio que puede tener efectos colaterales.

En /opt/Ejemplos_SGE/ del servidor di119 hay un conjunto de ejemplos que ilustran el uso del gestor de trabajos. El alumno debe copiar los ejemplos a su cuenta y:

6. Ejecutar el ejemplo secuencial (/opt/Ejemplos_SGE/Secuencial). El profesor irá guiando y explicando cada ejecución.
7. Ejecutar el ejemplo para Memoria Compartida (/opt/Ejemplos_SGE/OpenMP). El profesor irá guiando y explicando cada ejecución.
8. Ejecutar el ejemplo para GPUs (/opt/Ejemplos_SGE/GPUs). El profesor irá guiando y explicando cada ejecución.

Convenio para la interfaz de lenguaje C desde Python

En este curso académico se usará Python para simplificar ciertas tareas colaterales que directamente desde C pueden ser más complejas. Es decir, el alumno seguirá desarrollando los núcleos computaciones (secuenciales, paralelos, SIMT, etc.) en C usando librerías de altas prestaciones, pero se apoyará, cuando sea preciso, en el intérprete de Python para tareas colaterales (lectura de ficheros, importar/exportar datos de/a ficheros tipo csv/xls, visualizar imágenes, comprobar errores, etc.)

Para simplificar el trabajo del alumno se ha adoptado el siguiente convenio para las funciones (*drivers*) en C a ser "usadas" desde Python:

- Las funciones C serán de tipo *void* (no retornan nada) o retornan un único valor escalar, que siempre será de tipo *double*.
- Los parámetros que reciben las funciones C (sus argumentos) pueden ser escales (enteros, reales, etc.) o estructuras de datos homogéneos tipo double almacenados consecutivamente en memoria.
- Por simplicidad, las estructuras serán siempre objetos tipo *array numpy*.
- Si hay que modificar una estructura esta será pasada como argumento, por lo tiene que estar creada previamente en Python (p. ej. si se va a resolver el problema C = A * B, siendo A, B y C matrices, se llamará al driver en C pasando como argumentos tanto A/B como C).

Importante. Cuando se pasa a C desde Python la referencia a un *array numpy N* dimensional (con *N* mayor que 1) Python lo transforma en un puntero a "vector", donde los elementos serán almacenados linealmente siguiendo el método *Row-Major Order*, que se explicará en clase junto con otros modelos más complejos.

¿Qué debe hacer el alumno? (Fase 2ª)

En /opt/Ejemplos_SGE/Py2C del servidor di119 hay dos ejemplos de uso de funciones C desde Python. El alumno debe copiar los ejemplos a su cuenta y analizar la estructura, así como el contenido de los ficheros. Seguidamente ejecutar:

1. Ejecutar el ejemplo para CPU (/opt/Ejemplos_SGE/CPU).
2. Ejecutar el ejemplo para GPU (/opt/Ejemplos_SGE/GPU) en un ordenador sin GPU (nodos I3, por ejemplo).
3. Ejecutar el ejemplo para GPU (/opt/Ejemplos_SGE/GPU) en un ordenador con GPU.

Summary card básica Linux

- **Ficheros y directories** cd, pwd, ls, cp, mkdir, rm, mv, scp, less, cat, chmod, find, etc.
- **Gestión procesos** w, kill, ps, top, etc.
- **Editores y otros** vim, emacs, joe, nano, man, etc.

Summary card básica OEG/SGE

- **qsub** para enviar trabajos. Para simplificar su uso se han definido varios envoltorios (*wrappers*) para cada tipo de trabajo. El nombre de todos los envoltorios empieza por el

prefijo "*Cola*" (en una terminal de texto escribir "*Cola*" y presionar la tecla "*Tab*" para que el sistema muestre los disponibles en cada momento). Los envoltorios actuales son:

1. **ColaI3** Para trabajos secuenciales / OpenMP usando un único nodo. Cada nodo dispone de 1 procesador Intel® Core™ I3-2100 @ 3.10GHz con 2 núcleos (*cores*).
2. **ColaXeon** Para trabajos secuenciales / OpenMP usando un único nodo. Cada nodo dispone de 2 procesadores Intel® Xeon® E5620 @ 2.40GHz, con 4 núcleos por procesador.
3. **ColaGPU** Para trabajos secuenciales / OpenMP / SIMT usando un único nodo. Cada nodo dispone de 1 procesador AMD Ryzen 7 3700X @ 3.6GHz con 8 núcleos y GPU NVIDIA GeForce GTX 1660 Ti.

Por ejemplo, para lanzar un trabajo secuencial en un nodo con procesador I3 ejecutar, en la consola, la orden:

ColaI3 Lanza_Secuencial.sh

Lanza_Secuencial.sh es el *script* de órdenes para OEG/SGE. Cuando un trabajo finaliza el gestor crea en el directorio de trabajo (donde se ejecutó la orden "Colaxxxx …") un fichero con sufijo .o**id** (**id** es el identificador del trabajo dentro de OGE/SEG -4 dígitos numéricos-) con la salida estándar (y de error si los hubiera).

- **qstat** Para ver los trabajos en ejecución, en cola, suspendidos, etc. **qstatus –a** es similar a qstat.
- **qdel** Para borrar un trabajo de la cola. Se recomienda el uso del *flag* "-f" que, aunque es más lento, es más efectivo.
- **qhost** Para ver el estado de los nodos de cálculo del gestor.

Summary card básica C

El diseño de las prácticas está basado en el uso del lenguaje *C* por ser el mejor integrado/soportado por los estándares, librerías y entornos de desarrollo (p. ej. MPI, CUDA) que se usarán durante las prácticas. Además, la coordinación vertical del Grado garantiza que los alumnos lo conocen.

No obstante, el alumno puede elegir cualquier otro lenguaje de programación, corriendo por su cuenta la integración, depuración, licencias, etc. Si el lenguaje elegido presenta alguna carencia no será causa justificativa para no realizar el trabajo programado.

- Variables y Tipos Básicos. Todas las variables han de ser declaradas.

▪ Enteros:	char, int, long, modificador unsigned
▪ Enumerados:	enum (equivale a un entero)
▪ Coma flotante:	float, double
▪ Tipo vacío:	void (uso especial)
▪ Tipos derivados:	struct, arrays, punteros
▪ typedef	Definición de nuevos tipos

```
char c;
int a, b;
enum {N, S, E, O}  cardinal;
unsigned int k;
const float pi=3.141592;
c = 'a';
a= 1;
b=32*a;
cardinal=S;
k=(unsigned int)cardinal;
typedef enum {rojo, verde, amarillo, azul, negro} color;
color c1,c2;
```

- Tipos de Variables

▪ Globales
  ▪ Se declaran fuera de cualquier función
  ▪ Acceso desde cualquier punto del programa
  ▪ Se crean en el segmento de datos

▪ Locales
  ▪ Declaradas dentro de una función
  ▪ Visibles dentro del bloque
  ▪ Se crean en la pila (stack), se destruyen al salir

▪ Estáticas
  ▪ Modificador static
  ▪ Ámbito local pero persistentes de una llamada a otra

▪ Dinámicas
  ▪ Memoria reservada con malloc, persisten hasta el free
  ▪ Se crean en el heap

```
#include <stdio.h>
#include <stdlib.h>
void leedatos( char *filename) {
  FILE *fd;
  int i, n,*ia,*ja;
  double *va;

  fd = fopen(filename,"r");
  If (!fd)
    exit(1);

  fscanf(fd,"%i",&n);

  ia = (int*) malloc(n*sizeof(int));
  ja = (int*) malloc(n*sizeof(int));
  va = (double*) malloc(n*sizeof(double));

  for (i=0;i<n;i++)
    fscanf(fd,"%i%i%lf",ia+i,ja+i,va+i);
  fclose(fd);
  procesa(ia,ja,va);

  free(ia); free(ja); free(va);
}
```

- Sentencias
  - Declaración de variables y tipos (dentro/fuera de función)
  - Expresión, típicamente una asignación **var=expr**
  - Sentencia compuesta (**bloque {...}**)
  - Condicionales (**if**, **switch**), bucles (**for**, **while**, **do**)
  - Sentencia vacía (;)
  - etc.

- Expresiones
  - Asignaciones =, +=, -=, *=, /=
  - Incrementales ++, --
  - Aritméticas +, -, *, /, %
  - A nivel de bit ~, &, |, ^, <<, >>
  - Lógicas ==, !=, <, >, <=, >=, ||, &&, !
  - El cero se asimila a “falso” y cualquier otro a “verdadero”
  - Operador ternario a? b: c

```
if (j>0)
  valor = 1.0;
else
  valor = -1.0;
```

```
if (i>1 && (q[i]-1.0)<1e-7) {
  zm1[i] *= 1.0+sk1[i-1];
  zm2[i] *= 1.0+sk1[i-1];
}
else {
  zm1[i] *= 1.0+sk0[i-1];
  zm2[i] *= 1.0+sk0[i-1];
}
```

```
switch (dir)
{
  case NORTE:
    y += 1;
    break;
  case SUR:
    y -= 1;
    break;
  case ESTE:
    x += 1;
    break;
  case OESTE:
    x -= 1;
    break;
  default:
    break;
}
```

```
for (i=0;i<n;i++)
  x[i] = 0.0;
```

```
k = 0;
while (k<n)
{
  if (a[k]<0.0)
    break;
  z[k] = 2.0*sqrt(a[k]);
}

for (i=0;i<n;i++)
{
  y[i] = b[i];
  for (j=0;j<i;j++)
  {
    y[i] -= L[i][j]*y[j];
  }
  y[i] /= L[i][i];
}
```

- Vectores
  - Colección de variables del mismo tipo

- En la declaración se indica la longitud
- Los elementos se acceden con un índice (empieza en 0)
- Vectores multidimensionales: **double matriz[N][M]**
- Las cadenas son vectores tipo c*har* acabadas con el carácter '\0'

```
#define N 10
int i;
double a[N], s=0.0;
for (i=0;i<N;i++)
  s = s + a[i];
```

- Punteros
  - Variable que contiene la dirección de otra
  - En la declaración se antepone * antes del nombre de variable
  - El operador & devuelve la dirección de una variable
  - El operador * permite acceder al dato apuntado
  - Aritmética de punteros
    - Operaciones básicas +, -, ++
    - El desplazamiento es del tipo al que apunta la variable
  - Puntero nulo
    - Su valor es cero (NULL)
    - Se usa para indicar un fallo
  - Puntero genérico
    - De tipo void*
    - Puede apuntar a variables de cualquier tipo
  - Puntero múltiple: double **p (puntero a puntero). Evitaremos usarlo.

```
double a[4] = {1.1, 2.2, 3.3, 4.4};
double *p, x;
p = &a[2];
x = *p;
*p = 0.0;
p = a;                  /* &a[0]  */
```

- Funciones. Un programa *C* se compone de, al menos, una función (la función *main*). Devuelven un valor (nada si la función es de tipo *void*). Se pueden declarar funciones antes de su definición (prototipo). Paso de parámetros por valor y por referencia.

```
int calcula(double, float, int);

double rad2deg(double x) {
  return x*57.29578;
}

void mensaje(int k) {
  printf("Fin en %d sec\n",k);
}
```

```
float fun1(float a,float b){          /* Por valor */
  float c;
  c = (a+b)/2.0;
  return c;
}
...
w = fun1(6.0, 6.5);
void fun2(float *a,float *b){         /* Por referencia */
  float c;
  c = ((*a)+(*b))/2.0;
  if (fun3(c)*fun3(*a)<0.0)
    *a = c;
  else
    *b = c;
}
...
fun2(&x,&y);
```

- Bibliotecas

▪ Operaciones con cadenas <string.h>
  ▪ Copiar cadena (strcpy), comparar cadena (strcmp)
  ▪ Copiar memoria (memcpy), inicializar memoria (memset)

▪ Entrada-salida <stdio.h>
  ▪ Estándar: printf, scanf
  ▪ Ficheros: fopen, fclose, fprintf, fscanf

▪ Utilidades estándar <stdlib.h>
  ▪ Gestión de memoria dinámica: malloc, free, etc.
  ▪ Conversiones: atof, atoi, etc.

▪ Funciones matemáticas <math.h>
  ▪ Gestión de memoria dinámica: malloc, free, etc.
  ▪ Conversiones: atof, atoi, etc.

El proceso de compilación consta de:

- **Preprocesado** Modifica el código fuente en C según una serie de instrucciones (directivas de preprocesado).
- **Compilación** Genera el código objeto (binario) a partir del código ya preprocesado. Por cada fichero *.c se genera un *.o (no si se hace en un solo paso). Contiene código máquina de las funciones y variables, y una lista de símbolos no resueltos.
- **Enlazado** (*ld*) Une los códigos objeto de los distintos módulos y bibliotecas externas para generar el ejecutable final. Resuelve todas las dependencias pendientes a partir de *.o y bibliotecas (*.a, *.so).

gcc -o ejemplo ejemplo.c funcion1.c -lm

Los compiladores modernos poseen un amplio abanico de opciones (*flags*) de optimización, algunas son muy potentes (para más información ejecutar "*man gcc*" o "*man icc*"). Algunas opciones de optimización (p. ej. -fast) y otras de paralelización / vectorización (p. ej. parallel, SIMD, etc.) deben ser usadas con precaución.