ordenamos

queremos ordenar unos simples números dentro de un rango

- ¿Cómo procedemos?
- ¿Cuáles son los aspectos/problemas a tratar?
- ¿Cómo lo trasladamos a un entorno programable?

desafios

¿Con qué desafios nos enfrentamos?

- selección del algoritmo
  (RAE: Conjunto (?!) ordenado (?!) y finito (?!) de operaciones que permite hallar la solución de un problema.)
  ¡menos mal que en tercero ya sabemos que es un algoritmo!

ahora para tal algoritmo de forma concurrente:

- división del trabajo
- distribución de los datos
- sincronización necesaria
- comunicación de los datos entre participantes
- comunicación de los resultados

desafios

Y también con:

- medición de características
- depuración del programa
- (fiabilidad de los componentes)
- (fiabilidad de la comunicación)
- (detección de la terminación)

Programar aplicaciones concurrentes puede ser divertido y frustante a la vez :-)

medición de tiempo: la carrera de *San Martiño*

explicado en pizarra en clase

medición de tiempo de ejecución (ejemplo en Java)

```java
class Timer {
  private long[] startTime;
  private long[] stopTime;

  Timer(int n) {
    startTime=new long[n];
    stopTime=new long[n];
  }
  public void Start(int i) {
    startTime[i]=System.nanoTime();
  }
  public void Stop(int i) {
    stopTime[i]=System.nanoTime();
  }
  public double Elapsed() {
    long minTime=startTime[0];
    for(int i=1; i<startTime.length; ++i)
      if(startTime[i] < minTime) minTime=startTime[i];
    long maxTime=stopTime[0];
    for(int i=1; i<stopTime.length; ++i)
      if(stopTime[i] > maxTime) maxTime=stopTime[i];
    return (maxTime-minTime)/1000000.0;
  }
}
```

resumen del uso de OpenMP

- OpenMP es una API abierta para la programación paralela bastante cómoda para usar (`www.openmp.org`) en C++ y Fortran.
- La versión actual es la 5.1 (noviembre 2020), usamos la que viene con el compilador (no hacemos nada ni sofisticado ni específico).
- OpenMP usa en C++ la técnica de las `#pragma` para introducir instrucciones OpenMP (que se ignoran cuando compilamos sin OpenMP).
- Se incluye el fichero `omp.h` y se compila (en g++) con la opción `-fopenmp`.

Bucle de cálculo (compara con prácticas)

```
#pragma omp parallel for
    for(unsigned i=0; i<nIter; ++i) {
      double d(std::tan(std::atan(
        std::tan(std::atan(
          std::tan(std::atan(
            std::tan(std::atan(
              std::tan(std::atan(123456789.123456789))
            ))
          ))
        ))
      )));
      d=std::cbrt(d);
    }
```

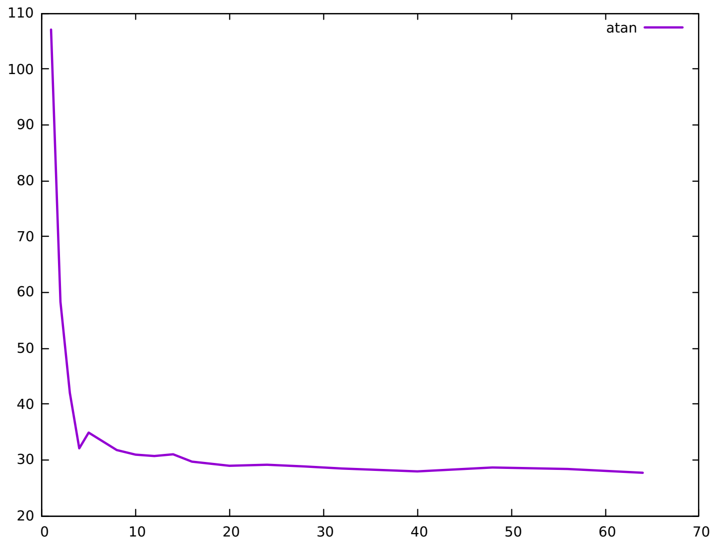
Optimizaciones del compilador

- si compilamos el código con optimización activado
- medimos un tiempo de ejecución casi 0 en caso secuencial
- dado que el compilador es capaz de detectar que no se realiza ninguna operación con la variable `d` fuera del bucle
- y elimina básicamente todo el bucle
  (*dead code elimination*, eliminación de código innecesario)

Bucle de cálculo con variable NO eliminable

```
#pragma omp parallel for \
    reduction(+:result)
    for(unsigned i=0; i<nIter; ++i) {
      double d(std::tan(std::atan(
        std::tan(std::atan(
          std::tan(std::atan(
            std::tan(std::atan(
              std::tan(std::atan(123456789.123456789))
            ))
          ))
        ))
      )));
      result+=std::cbrt(d);
    }
```

tiempo de ejecución observado en i7



tiempo de ejecución observado en i9