

## IMT2112 - Algoritmos Paralelos en Computación Científica

Threading

Elwin van 't Wout

3 de septiembre de 2019



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

Facultad de Matemáticas • Escuela de Ingeniería

imc.uc.cl

# Ayudantía previa

- Programar en C++
- Buenos tutoriales: [www.cplusplus.com](http://www.cplusplus.com)
- La universidad tiene licencias de Microsoft Visual Studio IDE (<https://informatica.uc.cl/licencias>)

# Agenda

- ¿Como programar la ejecución de múltiples hilos en paralelo?

# La computación paralela

Sección 2.6 del libro de Eijkhout

# La computación paralela

- Los compiladores son inteligentes pero no pueden paralelizar su código
- La parallelización sigue siendo principalmente la responsabilidad del programador

# Bibliotecas de computación científica

- Hay bibliotecas que proporcionan rutinas que han sido paralelizadas
  - Lapack, Blas, MKL, Petsc, Trilinos, NumPy, etc.
  - multiplicación de matriz-vector, factorización LU, transformada rápida de Fourier, etc.
- La mayoría de los lenguajes de programación interpretados tienen capacidades de paralelización limitadas
  - bloqueo global de intérpretes (GIL)
  - puedes abrir automáticamente varias instancias con el paradigma de *multiprocessing*

# Bibliotecas de computación paralela

- OpenMP (Open Multi-Processing) o Pthreads
  - paralelización de memoria compartida
  - directivas del compilador
  - relativamente fácil de programar
- MPI (Message Passing Interface)
  - paralelización de memoria distribuida
  - instrucciones explícitas para la transferencia de datos
  - relativamente difícil de programar
- Disponible para Fortran / C / C ++ / (Python)

# Paralelismo de hilos

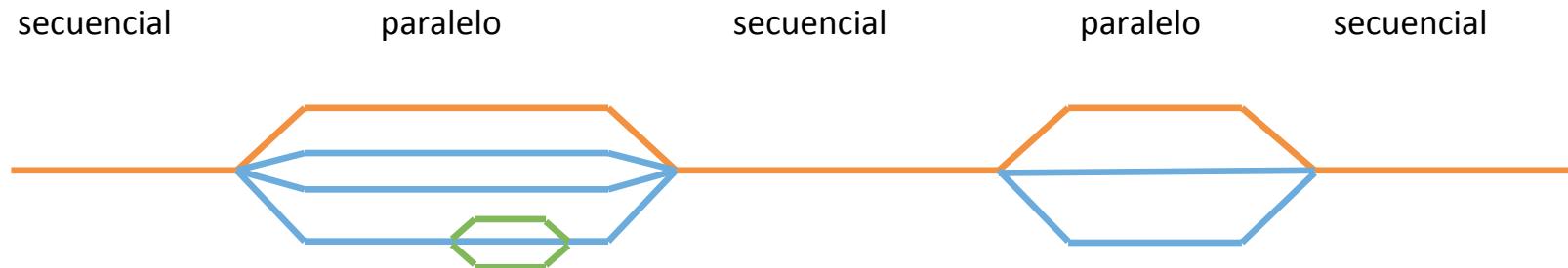
Sección 2.6.1 del libro de Eijkhout

# Terminología

- Un ‘proceso’ corresponde a la ejecución de un solo programa y contiene:
  - el código del programa (compilado)
  - el ‘*heap*’ con los datos asignados en la memoria
  - el ‘*stack*’ (‘pila’) con la información de cambio rápido:
    - contador de programa
    - elementos de datos locales
    - resultados intermedios
- Un ‘hilo’ (*thread*) es un subprocesso con su pila privada

# Hilos

- Un proceso puede crear hilos dinámicamente
- Recuerde que los subprocessos tienen una pila privada y pueden acceder a los datos en la memoria compartida
- El mecanismo de bifurcación ('*fork-join*') tiene un hilo maestro y crea otros hilos al generar ('*spawning*'')



# Los datos en los hilos

- Cuando se usan múltiples hilos, los datos pueden ser ‘privados’ o ‘compartidos’ (*private - shared*)
  - cuando se crea dentro de un hilo, los datos son privados
    - los datos privados se eliminarán cuando finalice el hilo
  - cuando se crea fuera de un hilo, los datos se comparten
    - la memoria compartida puede ser utilizada por todos los hilos
- El programador puede indicar el estado de un dato explícitamente también

# Consistencia de los hilos

- Un código se llama ‘seguro para hilos’ (‘*thread safe*’) si siempre produce el mismo resultado cuando se ejecuta secuencialmente y en paralelo
  - también se llama ‘consistencia secuencial’
- Una ‘condición de carrera’ (‘*race condition*’):
  - el resultado depende de qué hilo termine primero
- El principal peligro es escribir en datos compartidos
- Control del programador:
  - evitar múltiples escrituras en el mismo elemento de datos
  - usar secciones críticas
  - usar bloqueos (*locks*) en el acceso de elementos de datos

# OpenMP

Sección 2.6.2 del libro de Eijkhout

# Características de OpenMP

- OpenMP no es un lenguaje de programación sino una extensión para C / C ++ / Fortran
  - utiliza paralelismo dinámico
  - requiere una memoria compartida
  - usa directivas para el compilador
- Puede tomar un código secuencial existente y extender al cálculo en paralelo
  - casi imposible con MPI

# Un ejemplo de OpenMP

Un bucle paralelo se parece a lo siguiente

```
#pragma omp for
for (i=0; i<N; i++){
    a[i] = b[i];
}
```

# Un ejemplo de OpenMP

Recuerde que usar datos compartidos es peligroso para la seguridad del hilo

```
#pragma omp for
for (i=0; i<N; i++){
    t = b[i]*b[i];
    a[i] = sin(t) + cos(t)
}
```

# Un ejemplo de OpenMP

Puede y, a menudo, debe usar declaraciones explícitas para datos públicos y privados

```
#pragma omp parallel for shared(a,b), private(t)
for (i=0; i<N; i++){
    t = b[i]*b[i];
    a[i] = sin(t) + cos(t)
}
```

# Programación de tareas en OpenMP

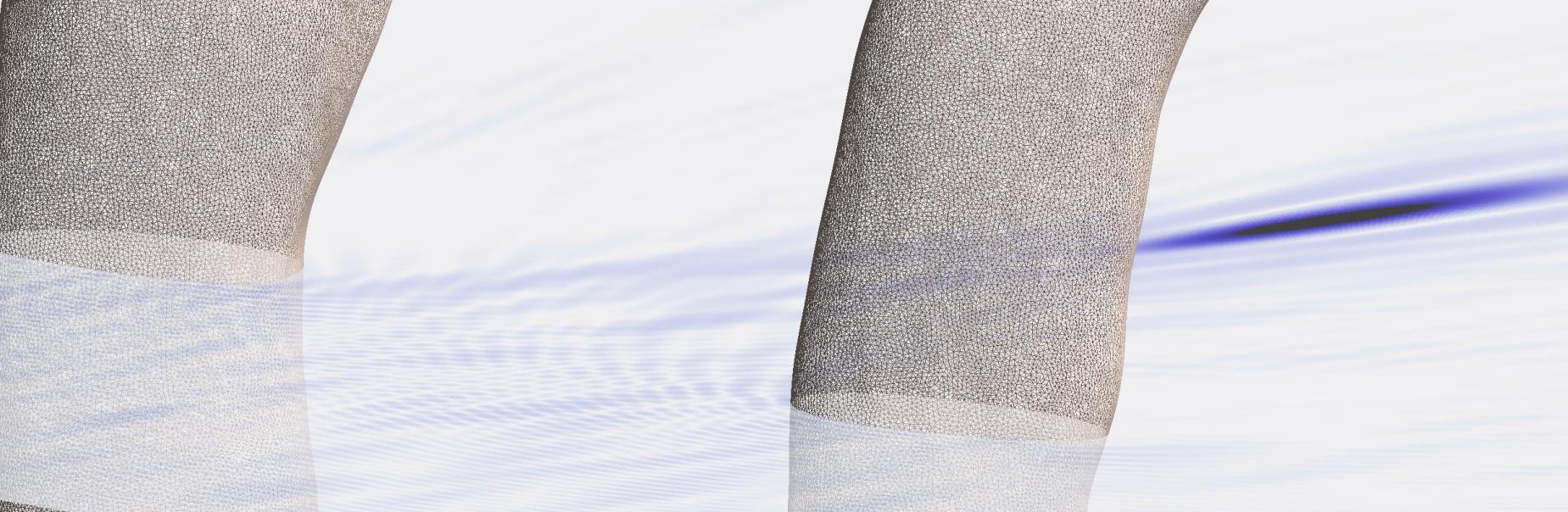
- Considere un bucle de tamaño  $N$  con  $P$  hilos
- Programación estática (*static scheduling*)
  - el hilo  $k$  ejecuta iteraciones  $k(N / P) \dots (k + 1)(N / P)$
  - ventaja: mejor reutilización de datos del caché
- Programación dinámica (*dynamic scheduling*)
  - cuando finaliza su tarea actual, cada subprocesso comienza a funcionar en la primera iteración no procesada
  - ventaja: mejor equilibrio de carga
- Control del programador: tipo programación y tamaño del fragmento (*chunk size*)

# Resumen

- Programación en paralela
- Paralelismo de hilos
- OpenMP

# Clase siguiente

- Las operaciones colectivas



## IMT2112 - Algoritmos Paralelos en Computación Científica

Threading

Elwin van 't Wout

3 de septiembre de 2019



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

Facultad de Matemáticas • Escuela de Ingeniería

imc.uc.cl