Práctica. Paralelización con OpenMP

Computación de Altas Prestaciones

Carlos García Sánchez

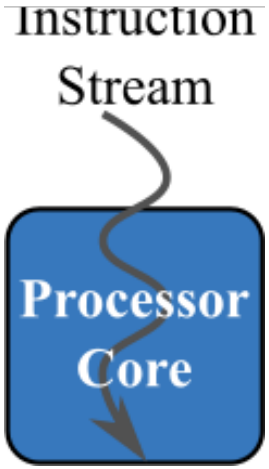
13 de octubre de 2021

- "Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition", James Jeffers, James Reinders, Avinash Sodani
- Youtube `https://www.youtube.com/watch?v=sELW6-3roAc&ab_channel=Danysoft`

Outline

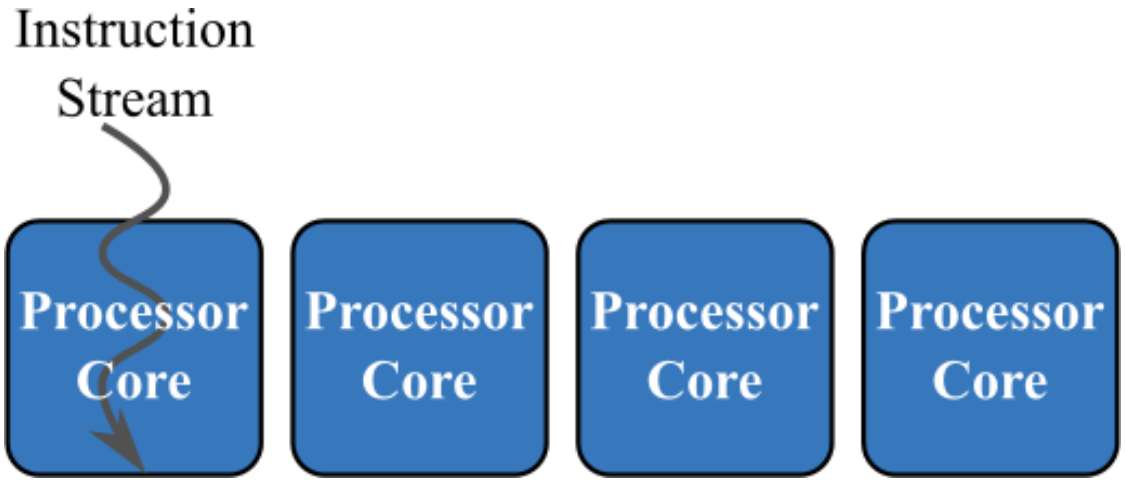
Secuencial vs Paralelo

- Aplicación secuencial en sistema con un único core



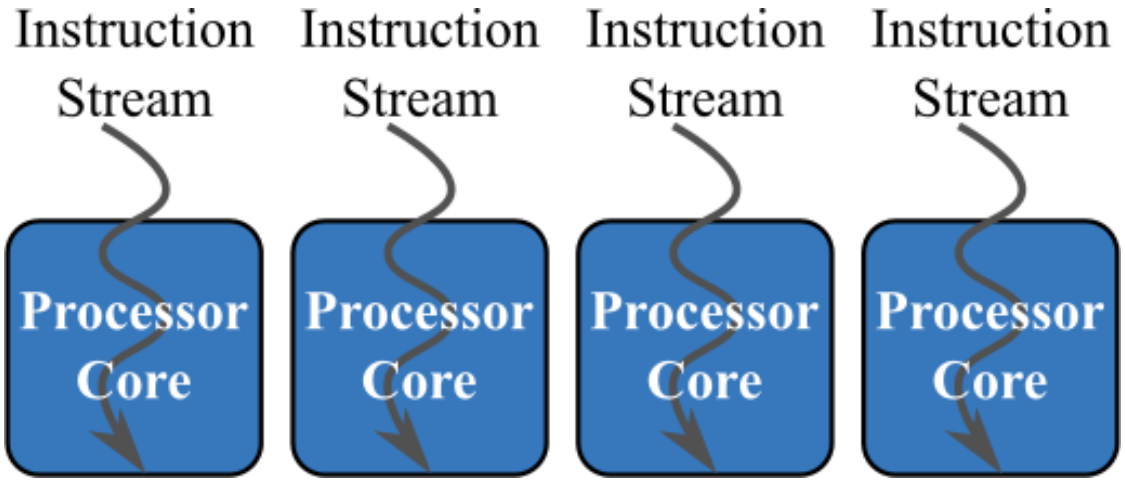
Secuencial vs Paralelo

- Aplicación secuencial en sistema con varios cores



Secuencial vs Paralelo

- Aplicación paralela en sistema con varios cores

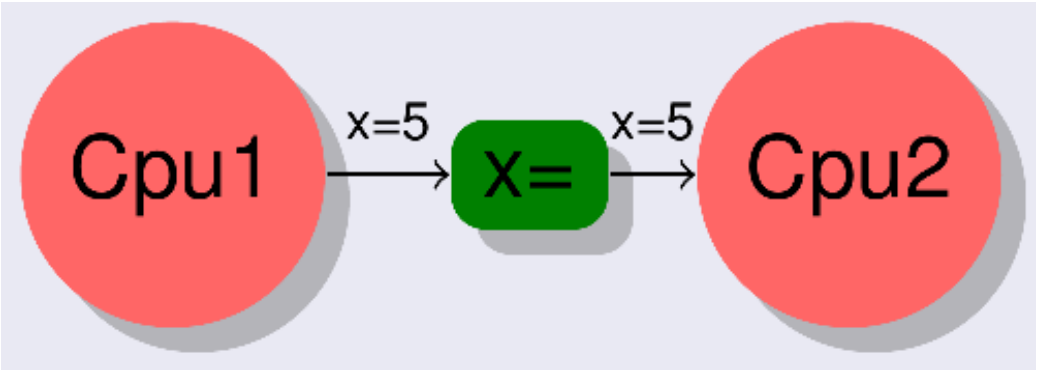


Modelos de memoria Sist. Paralelos

- Memoria compartida vs Memoria Distribuida

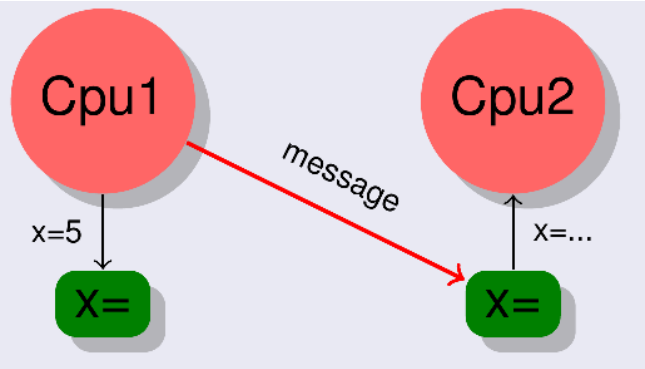
Modelos de memoria compartida

- Memoria es compartida por todos los procesadores: se permite el uso de **hilos**
- Existen mecanismos de comunicación y sincronización a través de la memoria compartida



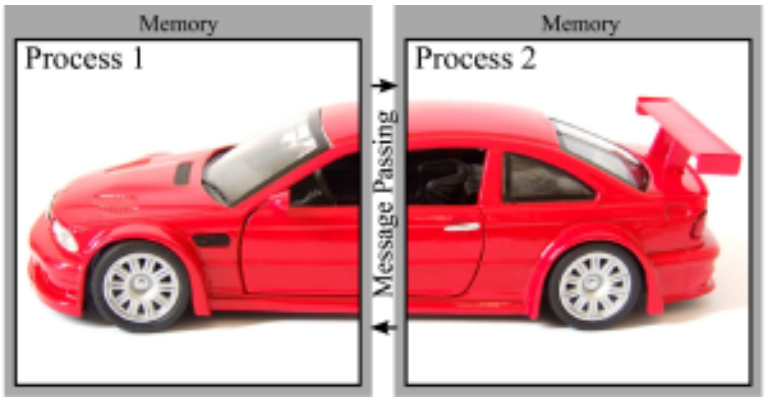
Modelos de memoria distribuida

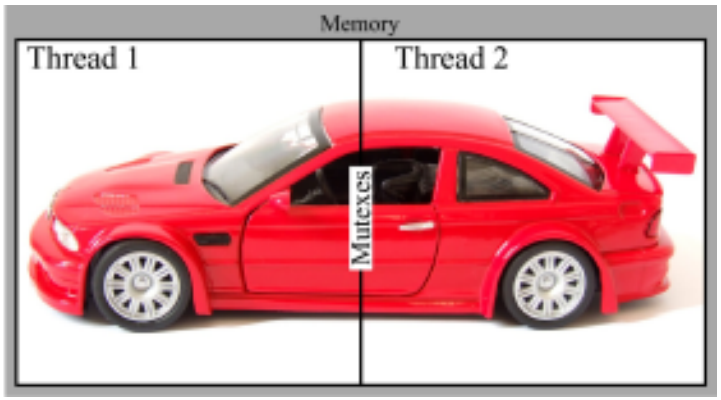
- Cada proceso tiene su propio espacio de direcciones de memoria
  - Ese espacio de memoria no es accesible por otros **procesos**
- La comunicación y sincronización se lleva a cabo explicitamente mediante mensajes



Modelos de memoria compartida vs distribuida

- Hilos (**OpenMP**) vs procesos (**MPI**)





OpenMP

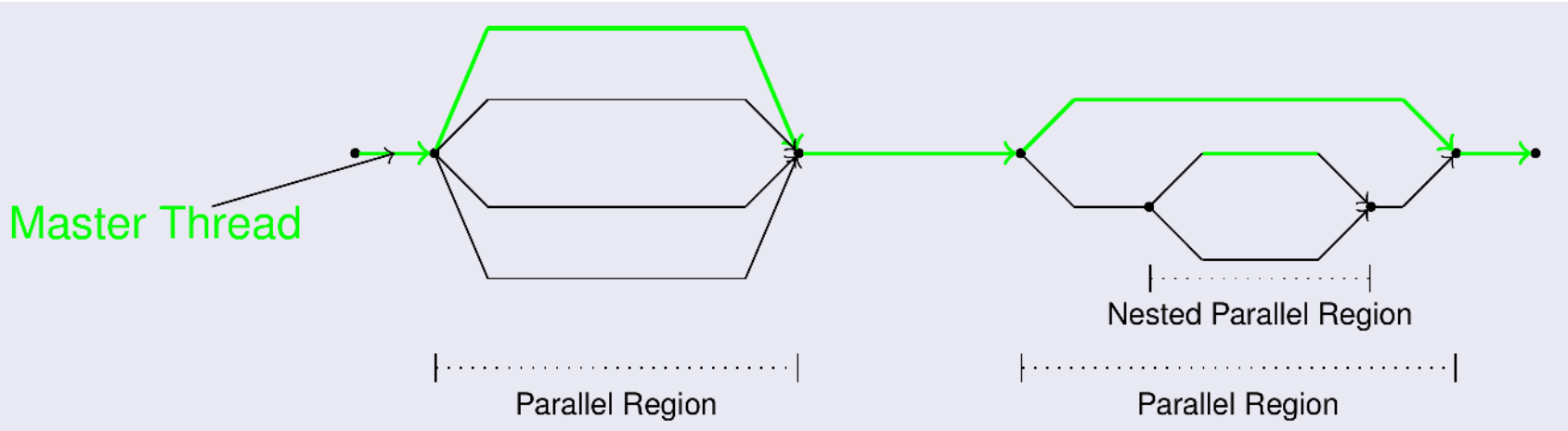
- OpenMP explota el **paralelismo mediante hilos/threads**
- Un hilo es entidad más pequeña de procesamiento
  - Más liviano que un proceso
- Habitualmente, un número de hilos se pueden mapear sobre una máquina multiprocesador/core

Paralelismo explícito

- OpenMP es un paradigma de programación explícito (no automático)
- Expresado mediante **directivas**
- Utiliza modelo **fork-join**

Modelo de ejecución

- Modelo **fork-join**
  - El hilo *master* crea un conjunto de hilos que acaban al finalizar la región paralela
  - Los hilos pueden colaborar



Variables de entorno

Compilador de Intel

- Compilador de Intel (ICC)
- Herramienta de Monitorización (ADVISOR y VTUNE)

Terminal #1

```
user@lab:~ $ export INTEL_STUDIO_PATH=/usr/local/intel_parallel_studio_xe_cluster/
user@lab:~ $ source $INTEL_STUDIO_PATH/bin/iccvars.sh intel64
user@lab:~ $ source $INTEL_STUDIO_PATH/advisor/advixe-vars.sh
user@lab:~ $ source $INTEL_STUDIO_PATH/compilers_and_libraries_2019/linux/mpi/intel64/bin/mpivars.
```

Ejemplo 0

- *Hello World*

hello.c

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

int main(){

   // This code is executed by 1 thread
   printf("OpenMP with %d max threads\n", omp_get_max_threads () );
   printf("OpenMP with %d procs available\n", omp_get_num_procs () );

   #pragma omp parallel
   {
   // This code is executed in parallel
   // by multiple threads
   printf("Hello World from thread %d of %d threads\n",
       omp_get_thread_num(), omp_get_num_threads() );
   }
}
```

Ejemplo 0

Terminal #1

```
carlos@posets:$ icc -o hello hello.c -qopenmp -qopt-report
carlos@posets:$ more hello.optrpt

...
Begin optimization report for: main()

    Report from: OpenMP optimizations [openmp]

OpenMP Construct at hello.c(12,2)
remark #16201: OpenMP DEFINED REGION WAS PARALLELIZED

...
```

Ejemplo 0

Terminal #1

```
carlos@posets:$ export OMP_NUM_THREADS=3
carlos@posets:$ ./hello
OpenMP with 3 max threads
OpenMP with 4 procs available
Hello World from thread 0 of 3 threads
Hello World from thread 1 of 3 threads
Hello World from thread 2 of 3 threads
```

Ejemplo 1: números primos

- El ejemplo determina la lista de números primos desde 1-*número entrada*
  - **i** es primo si no tiene divisores $(2, i/2)$
  - `#define DEBUG` visualiza la lista de primos

prime.c

```
//#pragma omp parallel...
for (i=2; i<n; i++)
{
    not_flag = 0;
    j=2;
    while(j<=i/2 && !not_flag)
    {
        if(i%j==0) // not prime
            not_flag=1;
        j++;
    }
    if (j>=i/2 && !not_flag)
        primes[++k] = i;
}
```

Ejemplo 1: números primos

- Iteraciones del bucle **i** potencialmente paralelas
  - Iteraciones independientes

A tener en cuenta

- Uso de variables: privadas, compartidas.... ect
- Variable `++k` es el índice de la lista de números primos
  - Posible carrera: problema **read&update&write**
  - Solución: `#omp critical` vs `#omp atomic`
  - Lista desordenada: implementar un *sort(primes)*

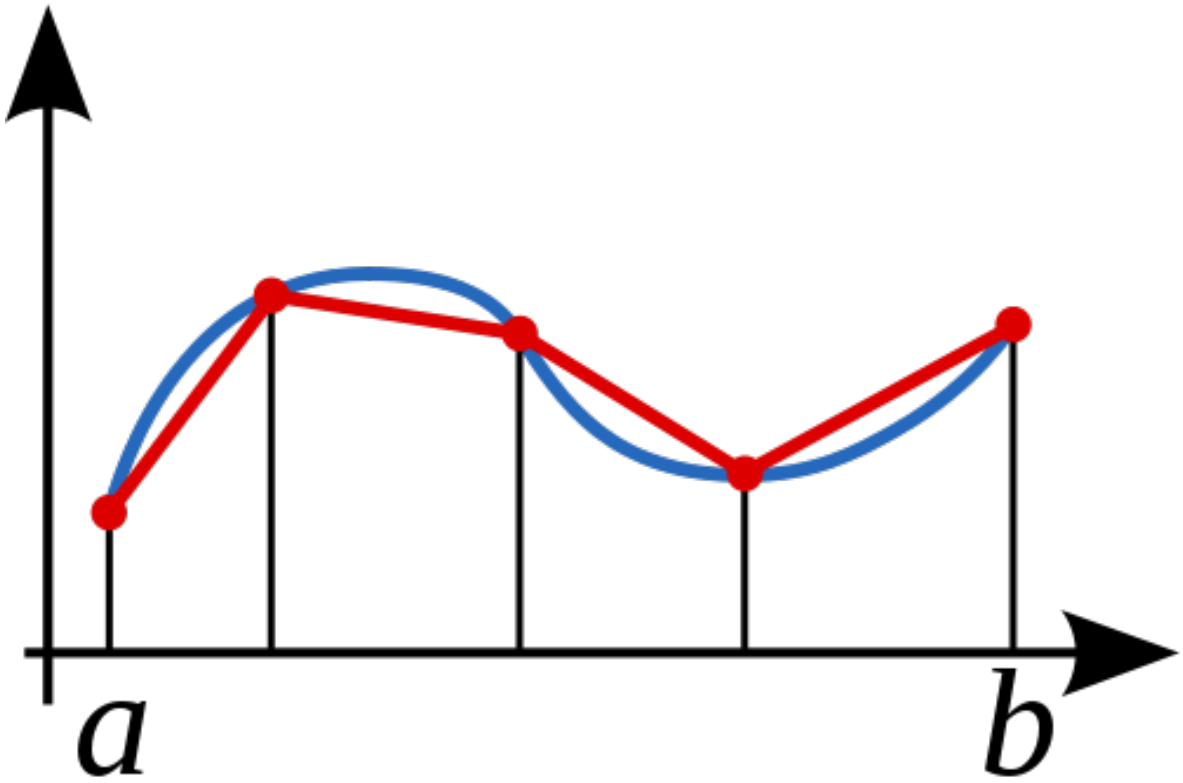
Ejemplo 1: números primos

Clausula `schedule`

- *STATIC*
- *STATIC, chunk*
- *DYNAMIC[, chunk]*
- *GUIDED[, chunk]*
- *AUTO*

Ejemplo 2: Método del trapecio

- Area del trapecio $S = \frac{1}{2}(f(a') + f(b')) * h$
- Integral como suma de trapecios: $\int_a^b f(x)\partial x = \sum \frac{f(a')+f(b')}{2} h$



Ejemplo 2: Método del trapecio

- Podemos destacar dos tipos tareas:
  - Cálculo de areas de cada trapecio individual
  - Acumulación de trapecios (*integral*)

trap.c

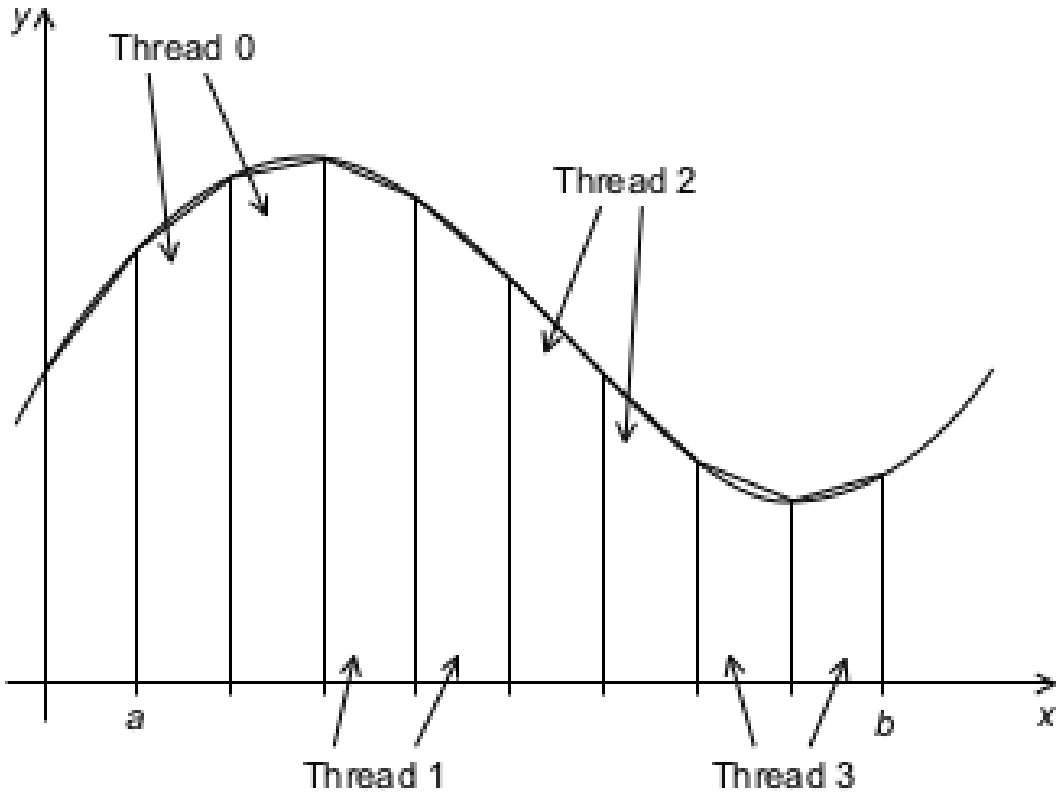
```
double Trap(double a, double b, int n, double h) {
    double integral, area;
    int k;

    integral = 0.0;
    for (k = 1; k <= n; k++) {
        area = h*(f(a+k*h)+f(a+(k-1)*h))/2.0;
        integral+=area;
    }

    return integral;
} /* Trap */
```

Ejemplo 2: Método del trapecio

Versiones

- `atomic`
- `critical`
- `paralel for`
  - `reduction`

Ejemplo 2: Método del trapecio

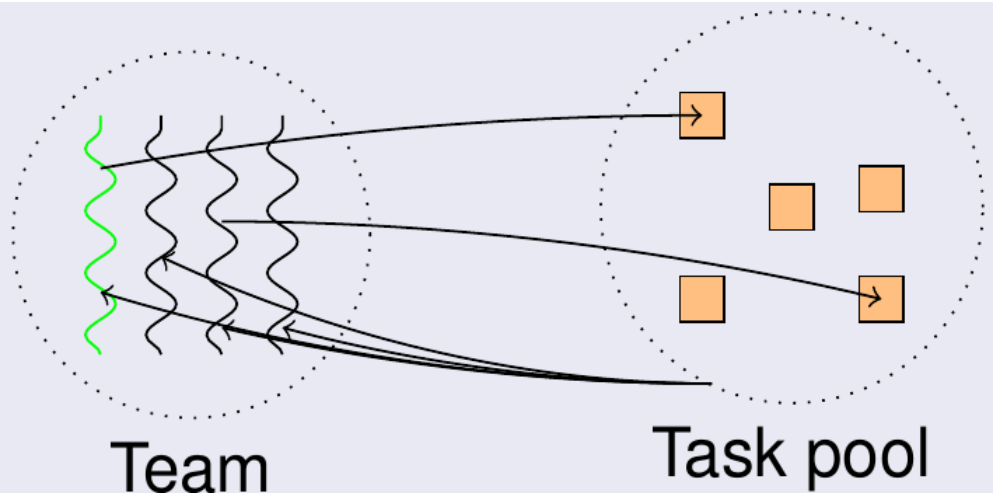


Ejemplo 2: Método del trapecio

trap_atomic.c

```
double Trap(double a, double b, int n, double h) {
   double integral, area, integral_thread;
   int k;

   int id, nths, n_per_thread, k_init_thread, k_end_thread;

   #pragma omp parallel private(...) firstprivate(...) shared(integral)
   {
   id  = ...
   nths = ...
   n_per_thread = n/nths;

   k_init_thread = ...
   k_end_thread = ...

   integral = 0.0;
   for (k = k_init_thread; k <= k_end_thread; k++) {
      area = h*(f(a+k*h)+f(a+(k-1)*h))/2.0;
      integral_thread+=area;
   }

   #pragma omp critical
   {integral += integral_thread;}

   }

   return integral;
}
```

Ejemplo 3: Paralelismo con tareas



¿Que es un tarea en OpenMP?

- Tareas = unidades de trabajo (ejecución puede diferirse)
- Las tareas se componen de:
  - código para ejecutar y datos
- Hilos pueden **cooperar** para ejecutarlas

Ejemplo 3: Paralelismo de tareas

- Sucesión de Fibonacci

Fibonacci

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \end{aligned} \tag{1}$$

Ejemplo 3: Paralelismo de tareas

fibo.c

```
long comp_fib_numbers(int n)
{
    long fnm1, fnm2, fn;

    fnm1 = comp_fib_numbers(n-1);
    fnm2 = comp_fib_numbers(n-2);
    fn = fnm1 + fnm2;

    return(fn);
}


int main(int argc, char* argv[]) {
    ....
    fibo = comp_fib_numbers(n);
    ...
} /* main */
```

Ejemplo 3: Paralelismo de tareas

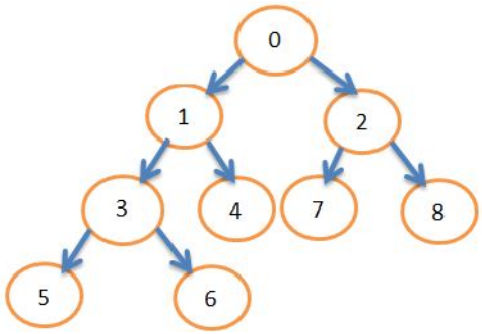
fibo_task.c

```
long comp_fib_numbers(int n)
{
    long fnm1, fnm2, fn;
    if ( n == 0 || n == 1 ) return(1);
    if ( n<20 ) return(comp_fib_numbers(n-1) +comp_fib_numbers(n-2));

    #pragma omp task...
    {fnm1 = comp_fib_numbers(n-1);}
    #pragma omp task...
    {fnm2 = comp_fib_numbers(n-2);}
    #pragma omp...
    fn = fnm1 + fnm2;

    return(fn);
}


int main(int argc, char* argv[]) {
    ....
#pragma omp parallel
{
    #pragma omp single
    fibo = comp_fib_numbers(n);
}
    ...
} /* main */
```

Ejemplo 3: Paralelismo de tareas

fibo_task.c

```
long comp_fib_numbers(int n)
{
    long fnm1, fnm2, fn;
    if ( n == 0 || n == 1 ) return(1);
    if ( n<20 ) return(comp_fib_numbers(n-1) +comp_fib_numbers(n-2));

    #pragma omp task...
    {fnm1 = comp_fib_numbers(n-1);}
    #pragma omp task...
    {fnm2 = comp_fib_numbers(n-2);}
    #pragma omp...
    fn = fnm1 + fnm2;

    return(fn);
}
```



Ejemplo 4: gemm

- $C_{NM} = A_{NK} * B_{KM}$
  - Sin ningún paralelismo
  - Tamaño definido con `#define NUM 1024` en *multiply.h*
- 5 versiones

multiply0.c

```
for(i=0; i<msize; i++) {
    for(j=0; j<msize; j++) {
        for(k=0; k<msize; k++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
    }
}
```

Ejemplo 4: gemm

- Versión 1: OpenMP
  - `#define MULTIPLY multiply1` en *multiply.h*

multiply1.c

```
// Basic parallel implementation
#pragma omp parallel for
for(i=0; i<msize; i++) {
    for(j=0; j<msize; j++) {
        for(k=0; k<msize; k++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
    }
}
```

Ejemplo 4: gemm

- Versión 2: OpenMP multi-bucle
  - `#define MULTIPLY multiply2` en *multiply.h*

multiply2.c

```
#pragma omp parallel for collapse (2)
for(i=0; i<msize; i++) {
    for(k=0; k<msize; k++) {
        for(j=0; j<msize; j++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
    }
}
```

Ejemplo 4: gemm

- Versión 3: OpenMP multi-bucle y vectorización
  - `#define MULTIPLY multiply3` en *multiply.h*

multiply3.c

```
#pragma omp parallel for collapse (2)
for(i=0; i<msize; i++) {
    for(k=0; k<msize; k++) {
        #pragma ivdep
        for(j=0; j<msize; j++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
    }
}
```

Ejemplo 4: gemm

- Versión 4: OpenMP multi-bucle, vectorización y desenrrollado
  - `#define MULTIPLY multiply4` en *multiply.h*

multiply4.c

```
#pragma omp parallel for collapse (2)
for(i=0; i<msize; i++) {
    for(k=0; k<msize; k++) {
        #pragma unroll(8)
        #pragma ivdep
        for(j=0; j<msize; j++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
    }
}
```

Ejemplo 4: gemm

- Versión 5: uso de librerías
  - BLAS (Basic Linear Algebra Subroutine)
  - Operación GEMM:
    - C = α op (A) op (B) + β C
    - α y β son escalares
    - $A_{m\times k}$, $B_{k\times n}$ y $C_{n\times n}$ en (row-major)
  - `#define MULTIPLY multiply5` en *multiply.h*

multiply5.c

```
double alpha = 1.0, beta = 0.;
cblas_dgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,
    NUM,NUM,NUM,alpha,
    (const double *)b,NUM,(const double *)a,NUM,beta,
    (double*)c,NUM);
```

Ejemplo 5 (N-Body)

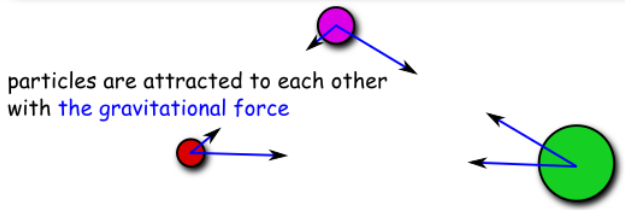
Gravitational N-body dynamics:

Newton's law of universal gravitation:

$$M_i \vec{R}_i''(t) = G \sum_j \frac{M_i M_j}{\left|\vec{R}_i - \vec{R}_j\right|^3} \left(\vec{R}_j - \vec{R}_i\right)$$

where:

$$\left|\vec{R}_i - \vec{R}_j\right| = \sqrt{(R_{i,x} - R_{j,x})^2 + (R_{i,y} - R_{j,y})^2 + (R_{i,z} - R_{j,z})^2}$$



Cálculo de fuerzas gravitatorias

$$F_{i,j} = m_i \frac{\partial^2 R_i}{\partial^2 t}$$

$$F_{i,j} = \frac{G m_i m_j \left(R_j - R_i\right)}{\left\|R_j - R_i\right\|^3}$$

$$m_i \frac{\partial^2 R_i}{\partial^2 t} = \sum_{j=0, i \neq j}^{N} \frac{G m_i m_j \left(R_j - R_i\right)}{\left\|R_j - R_i\right\|^3} = \frac{\partial U}{\partial R_i}$$

Ejemplo 5 (N-Body)

nbody.c

```
void bodyForce(body *p, float dt, int n) {
   for (int i = 0; i < n; i++) {
      float Fx = 0.0f; float Fy = 0.0f; float Fz = 0.0f;

      for (int j = 0; j < n; j++) {
         if (i!=j) {
            float dx = p[j].x - p[i].x;
            float dy = p[j].y - p[i].y;
            float dz = p[j].z - p[i].z;
            ....
```

Optimizaciones a estudiar

- Código vectorizado (Soa vs AoS)
- Doble bucle (`for i` y `for j`): **posible paralelización**

Ejemplo 6: ecuación del calor

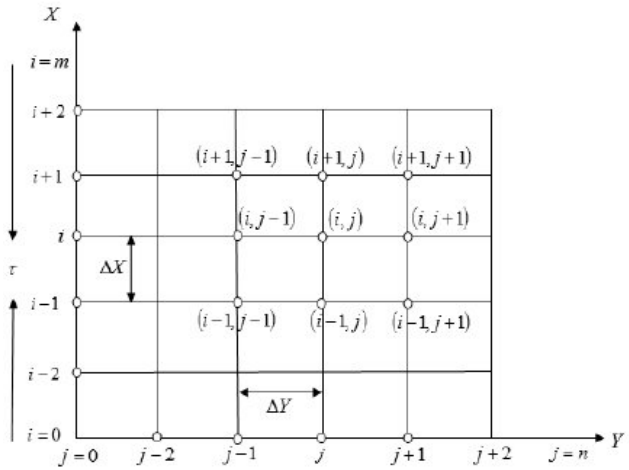
- Resolución Numérica de Problemas de Transmisión de Calor

Diferencias finitas

1. División del espacio considerado en una serie de elementos cuyas propiedades vienen representadas por un punto central (nodo)
2. Aplicación de balances de energía a cada elemento, obteniendo la ecuación característica para cada nodo.
3. Resolución simultánea de todos los balances, para obtener el perfil de temperaturas.
4. Si el caso lo requiere cálculo del flujo de calor con la ley de Fourier y el perfil de temperaturas.

Ejemplo 6: ecuación del calor

- Considerando la ecuación del calor en 2 dimensiones:
  $Q = -kA\frac{\partial T}{\partial x} \cong -k\frac{\Delta x T}{\Delta x} = -(W\Delta y)\frac{T_{i+1,j} - T_{i,j}}{\Delta x}$
  - donde Q se ha agregado como un término de generación de calor (positivo para la generación)



Ejemplo 6: ecuación del calor

- Para una celda (i, j) la aportación de calor $Q = Q_1 + Q_2 + Q_3 + Q_4$ (celdas vecinas) donde:
  - $Q_1 = \frac{T_{i,j+1} - T_{i,j}}{\Delta y / k(W \Delta x)}$
  - $Q_2 = \frac{T_{i+1,j} - T_{i,j}}{\Delta x / k(W \Delta y)}$
  - $Q_3 = \frac{T_{i-1,j} - T_{i,j}}{\Delta x / k(W \Delta y)}$
  - $Q_4 = \frac{T_{i,j-1} - T_{i,j}}{\Delta y / k(W \Delta x)}$
- En el estado estacionario $Q_1 + Q_2 + Q_3 + Q_4 = 0$
  - ... luego haciendo las sumas anteriormente descritas $T_{i,j} = \frac{T_{i,j+1} + T_{i,j-1} + T_{i+1,j} + T_{i-1,j}}{4}$

Ejemplo 6: ecuación del calor

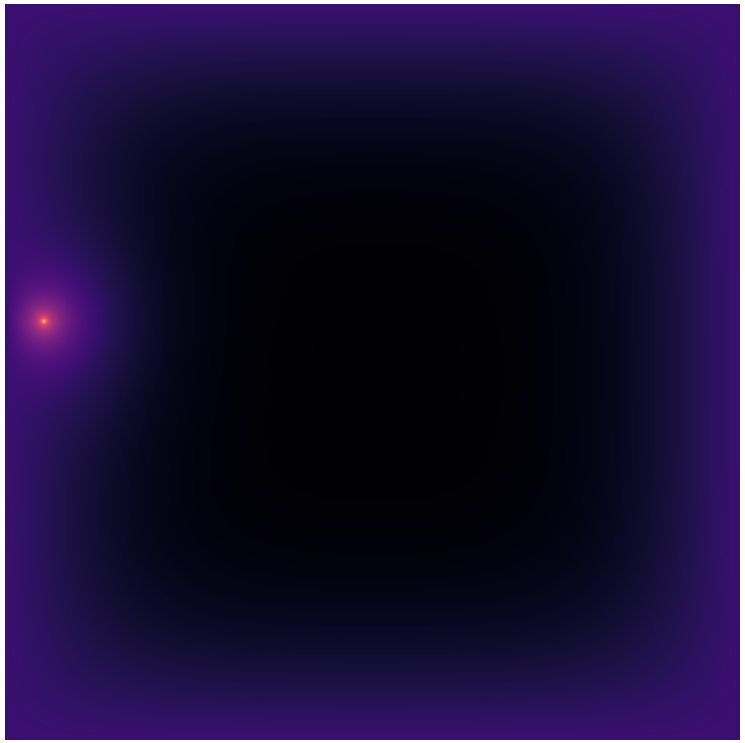
- Inicializa con un valor rand las coordenadas de la fuente de calor: `source_x, source_y`
- Inicializa las condiciones de contorno
- Ejecución del bucle principal `(it < MAX_ITERATIONS) && (t_diff > MIN_DELTA)`
- Salida en fichero .png que muestra el calor en una placa 2D

heat.c

```
for (unsigned int it = 0; (it < MAX_ITERATIONS) && (t_diff >
     MIN_DELTA); ++it) {
    step(source_x, source_y, current, next);
    t_diff = diff(current, next);
    if(it%(MAX_ITERATIONS/10)==0){
       printf("%u: %f\n", it, t_diff);
    }

    float * swap = current;
    current = next;
    next = swap;
}
```

Ejemplo 6: ecuación del calor

- Tras 20000 iteraciones



Ejemplo 7: Navier-Stokes

- Implementación de dinámica de fluidos como resolutor de ecuaciones para motores de juegos [1]

Ecuaciones

1. $\frac{\delta u}{\delta t} = -(u \cdot \nabla)u + \nu \nabla^2 u + f$
2. $\frac{\delta \rho}{\delta t} = -(u \cdot \nabla)\rho + \kappa \nabla^2 \rho + S$

- Donde u corresponde a la velocidad y ρ al movimiento de la densidad repecto a la velocidad

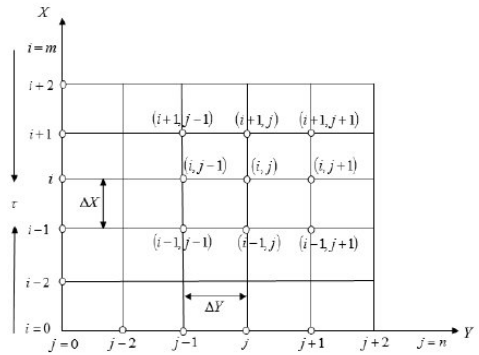
[1] Real-Time Fluid Dynamics for Games.
https://www.youtube.com/watch?v=UM3VFnHBi0U

Ejercicio 7: Navier-Stokes

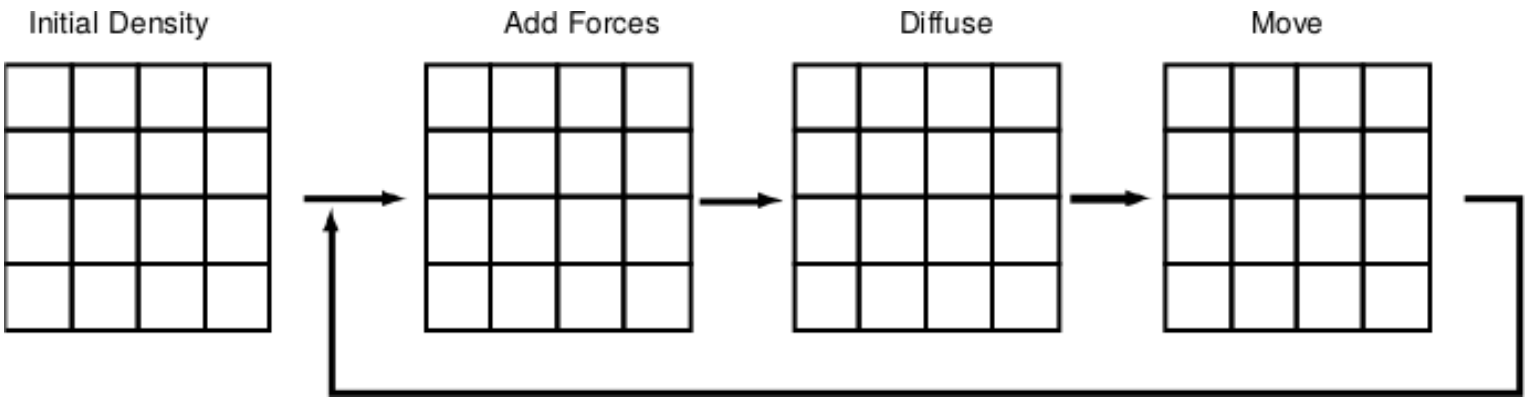
- Matemáticamente, el estado de un fluido en un instante de tiempo determinado se modela como un vector de velocidad: una función que asigna un vector de velocidad a cada punto del espacio
  - Ej: aire de radiador en una habitación, ciculará ascendentemente debido al aumento de calor
- El campo velocidad no es visualmente interesante hasta que se produce movimiento de objetos: como particulas de humo, polvo o las hojas

Ejercicio 7: Navier-Stokes

- El modelo se basa en el fluido que recorre una caja, por lo que se modelará como un espacio mediante diferencias finitas
  - `u[size]`, `v[size]`, `u_prev[size]`, `v_prev[size]` representan las velocidades en una malla de tamaño `size=(N+2)*(N+2)`
  - `dens[size]`, `dens_prev[size]` corresponde a la densidad del fluido
  - Acceso a las cordenadas se realiza con macro `#define IX(i,j) ((i)+(N+2)*j)`

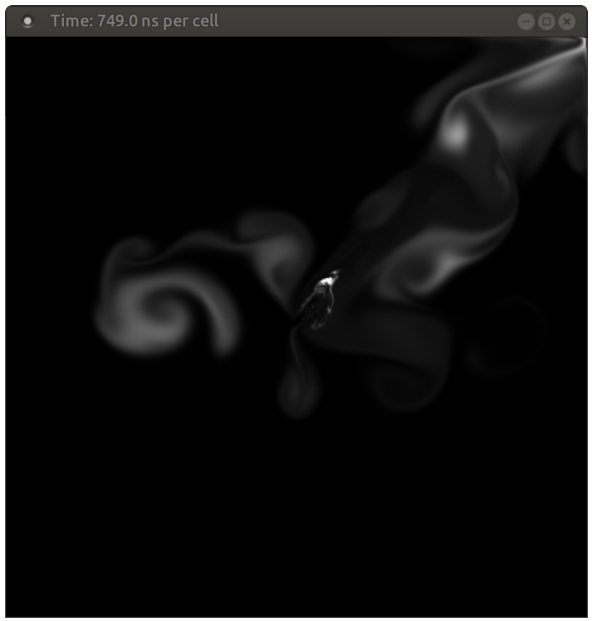


Ejercicio 7: Navier-Stokes



Ejercicio 7: Navier-Stokes

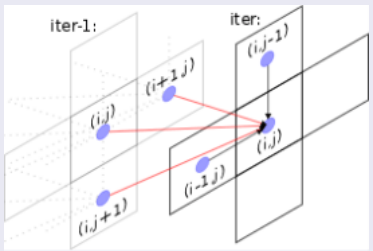
- Exiten dos ejecutables: **demo** y **headless**
  - **demo** es simulación gráfica (*botón derecho del ratón* añade densidad, *izquierdo* velocidad al fluido, *v* muestra velocidades y *c* inicializa simulación)
  - **headless** realiza 2048 iteraciones



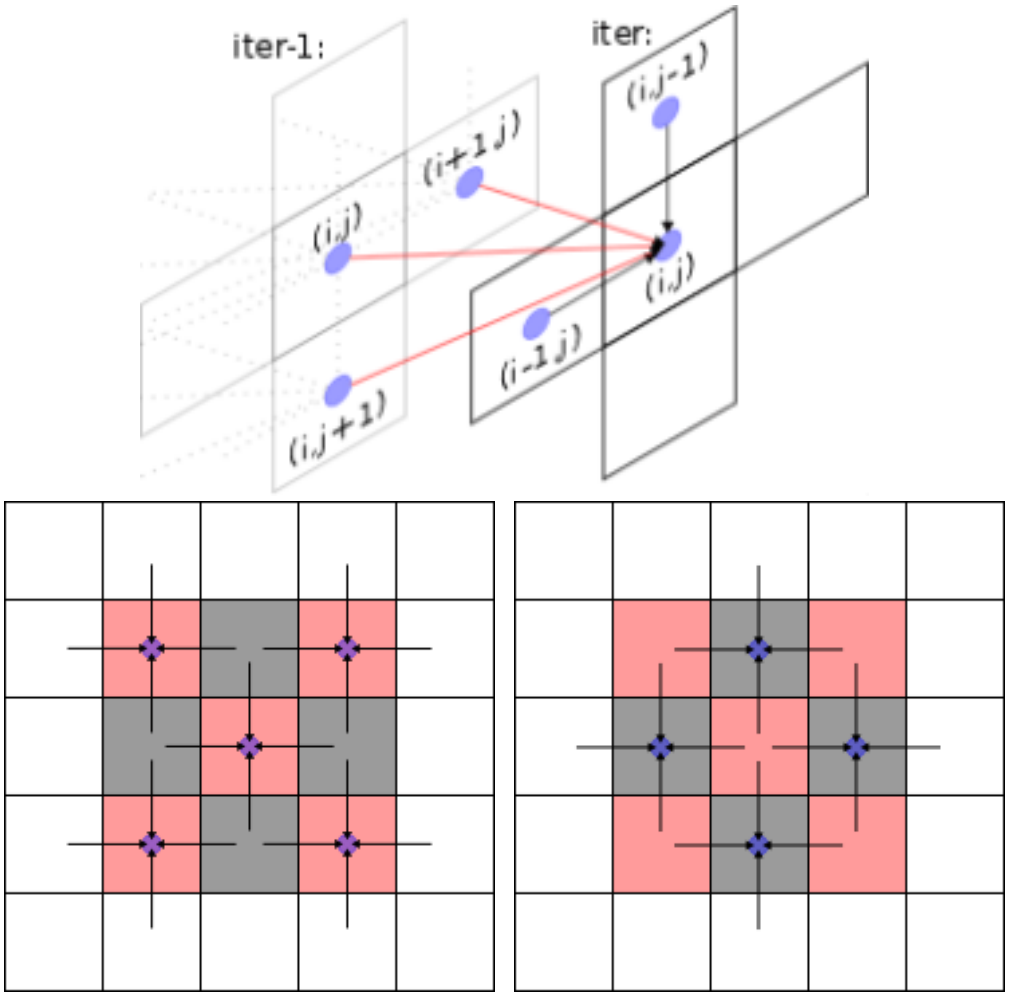
Ejercicio 7: Navier-Stokes

Optimizaciones

- Vectorización (recordad bucles independientes, accesos alineados, accesos consecutivos...)
- Paralelización: conveniente en bucles externos
- Función `lin_solve` del fichero *solver.c* resuelve las ecuaciones aplicando el método numérico Gauss-Seidel (más complicado su paralelización), conviene resolverlo aplicando el método Jacobi (paralelización evidente) o en su defecto un red-black

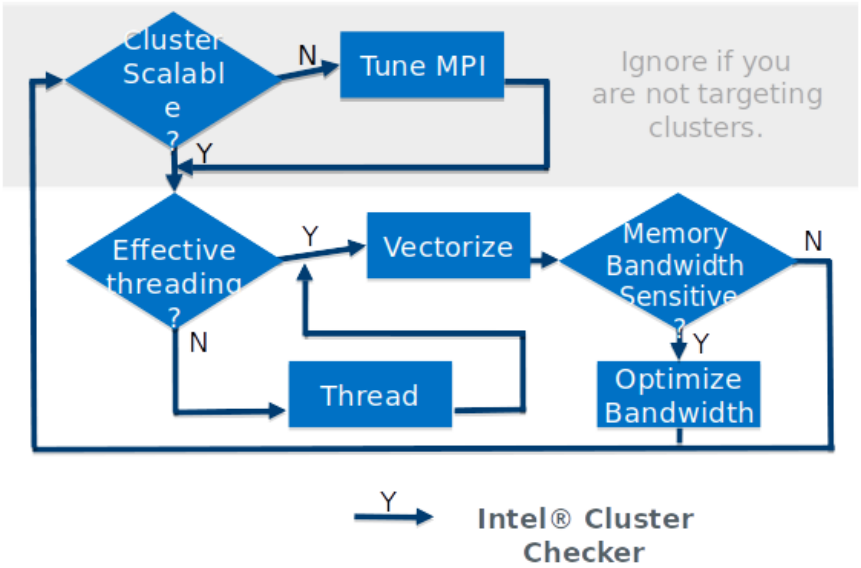


Ejercicio 7: Navier-Stokes



Herramientas de profiler

- El proceso de paralelización se puede considerar un proceso iterativo



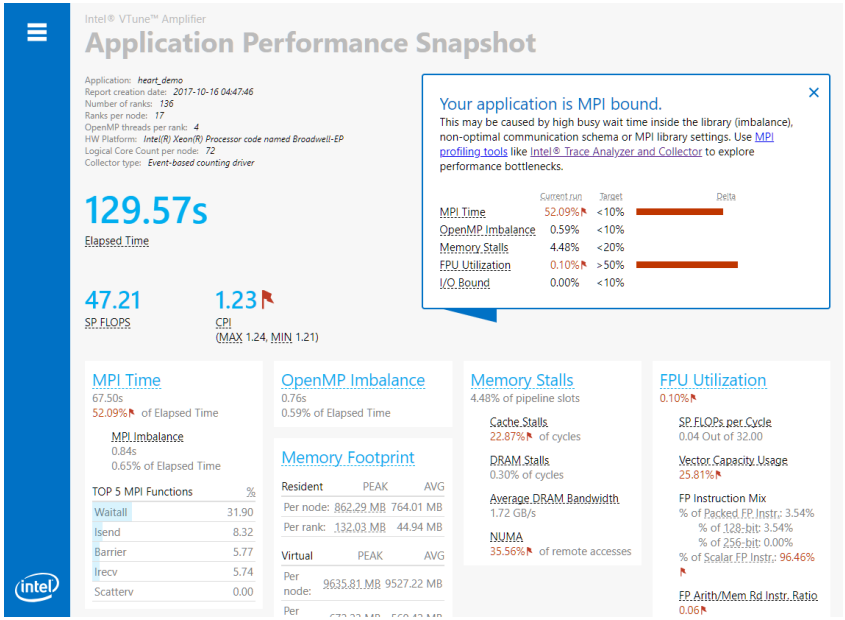
Intel APS-Application Performance Snapshot

- Vista rápida de algunos aspectos importantes en las aplicaciones de cómputo intensivo
  - Uso de MPI o OpenMP
  - Utilización de CPU
  - Acceso de memoria eficientes
  - Vectorización
  - E/S
  - Huella de la memoria

Intel APS-Application Performance Snapshot

- Fácil y rápido (vista rápida)
  - Haz un test en lo que tardas en preparer un café
  - Toda la información de un vistazo
- MPI + OpenMP + Memory + Floating Point
  - Soporta implementaciones MPI comunes
  - Intel® MPI, MPICH, OpenMPI y Cray MPI
- Novedades de version 2020
  - Diagnostico de comunicaciones
  - Tiempos en altos anchos de banda, no solamente édia

Intel APS-Application Performance Snapshot

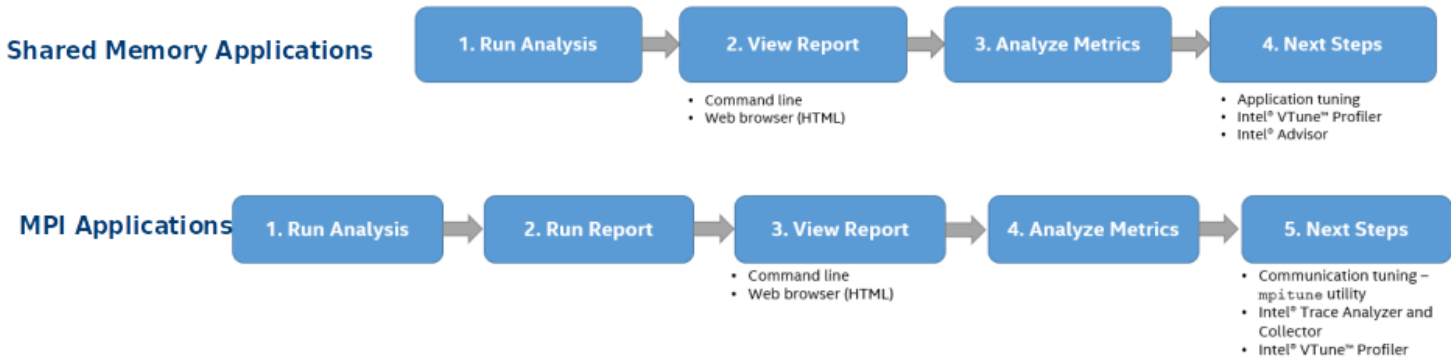


* Free download:

`intel.com/performance-snapshot`

Intel APS-Application Performance Snapshot

- Para ejecutar:
  - **aps my_app app_parameters**
  - Genera report HTML



Intel VTune

- Intel® VTune™ Profiler
- Como mejorar el rendimiento con varios análisis
  - Hotspots
  - Threading Efficiency
  - Microarchitecture
  - Memory Access

Intel VTune

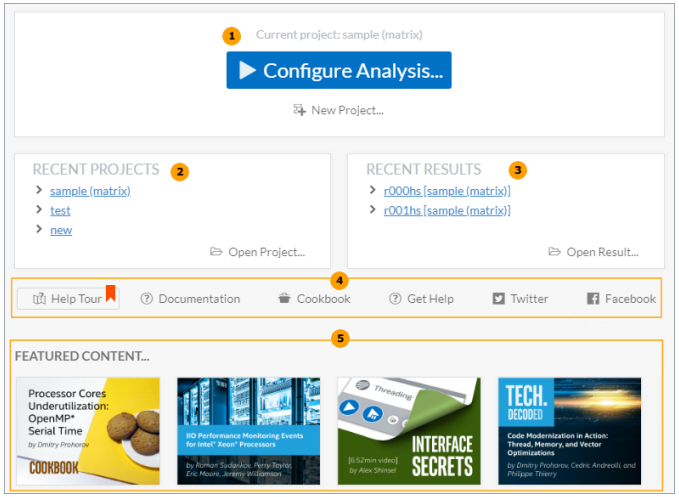
- Variables de entorno

Terminal #1

```
user@lab:~ $ export INTEL_STUDIO_PATH=/usr/local/intel_parallel_studio_xe_cluster/
user@lab:~ $ source $INTEL_STUDIO_PATH/vtune_amplifier/amplxe-vars.sh
```

- La Herramienta Intel Vtune Amplier nos permite realizar un perfilado de la aplicación paralela para detectar posibles mejoras
  - Lanzamiento herramienta gráfica amplxe-gui, más moderno **vtune-gui**
  - Lanzamiento por línea de comandos amplxe-cl,más moderno **vtune-collect**

Intel VTune: comienzo



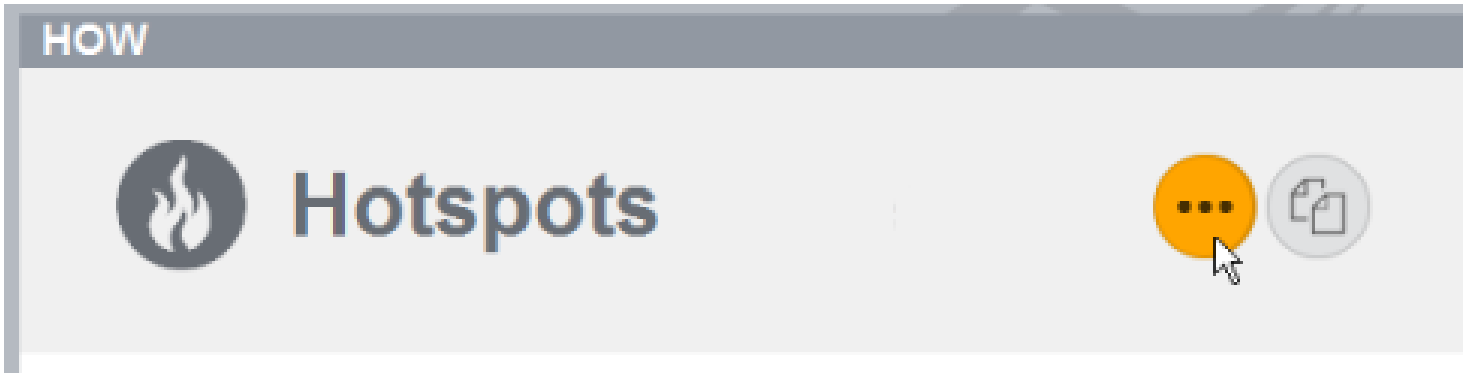
1. Proyecto actual con el análisis de configuración (*New Project*)
2. Proyectos recientes
3. Resultados recientes
4. Recursos y documentación online
5. Artículos de y últimas noticias

Intel VTune: Opciones de análisis

- Hotspots
  - Utilización de CPU
- Memory Consumption:
  - Consumo de memoria (malloc-free) en cada rutina
- Microarchitecure Exploration:
  - Analiza cuellos de botella que afectan al rendimiento
  - Contadores HW
- Memory Access
  - Identifica los accesos a jerarquía memoria (NUMA, DRAM, L2...)

Intel VTune: Análisis

- Más info en Intel-VTune[2] y vídeo [3]
- A la hora de crear el proyecto con *New Project* seleccionar *Local Host* para analizar en el equipo local
  - Introducir el ejecutable (*path*) y primer análisis *Hotspots*



[2] https://software.intel.com/en-us/vtune

[3] https://software.intel.com/en-us/videos/introduction-to-intel-vtune-amplifier

Intel VTune: Análisis Hotspots

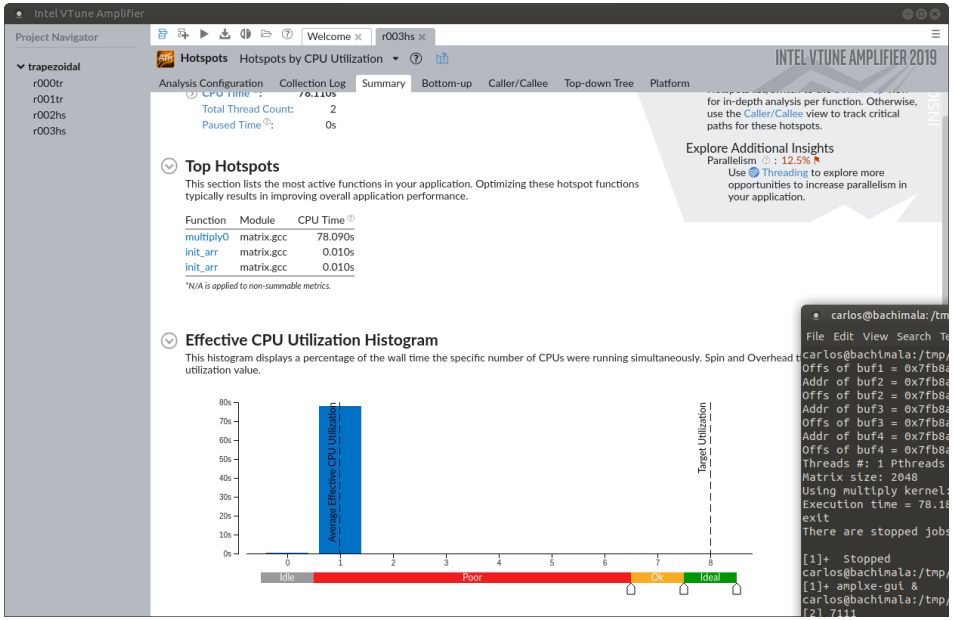
- *Hotspots* dirigido a la optimización de software y conocer dónde pasa tiempo la aplicación: **analizar la eficiencia**
- Incluye los análisis:
  - *Hotspots* para identificar las funciones más costosas y muestra la actividad de cada hilo
  - *Memory consuption* para analizar el consumo de memoria: caches y RAM

Intel VTune: Parallelism Analysis

- Tipos de análisis para aplicaciones sensibles en cómputo: análisis general del rendimiento
- Incluye los análisis:
  - *Threading*: muestra balanceo de trabajo entre hilos y puntos de sincronización
  - *Compute-intensive Application*: caracterización rendimiento de HPC (punto flotante y la eficiencia de la memoria)

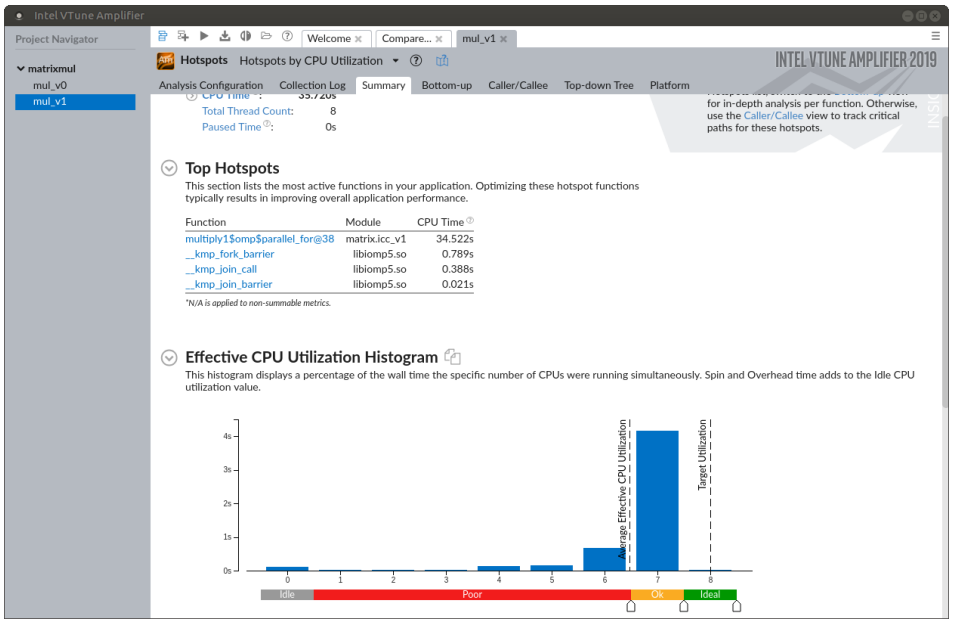
Intel VTune: multiply0

- Sin paralelismo de ningun tipo

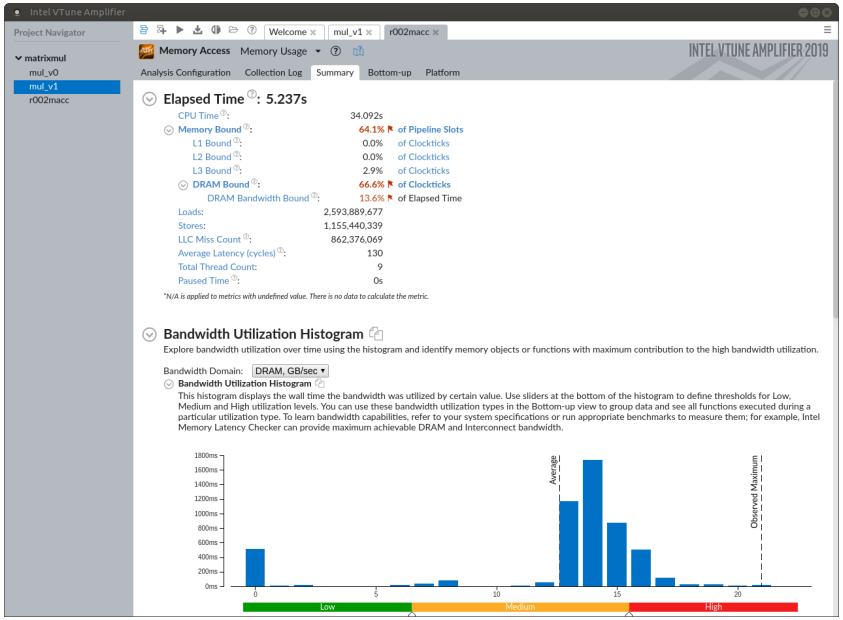


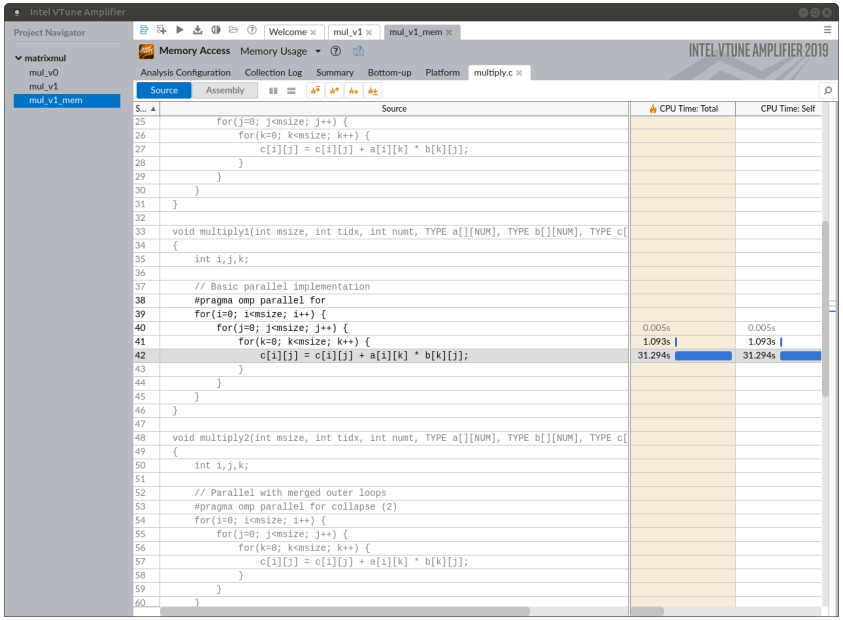
Intel VTune: multiply1

- Más eficiencia paralela (OpenMP), pero gran demanda de memoria



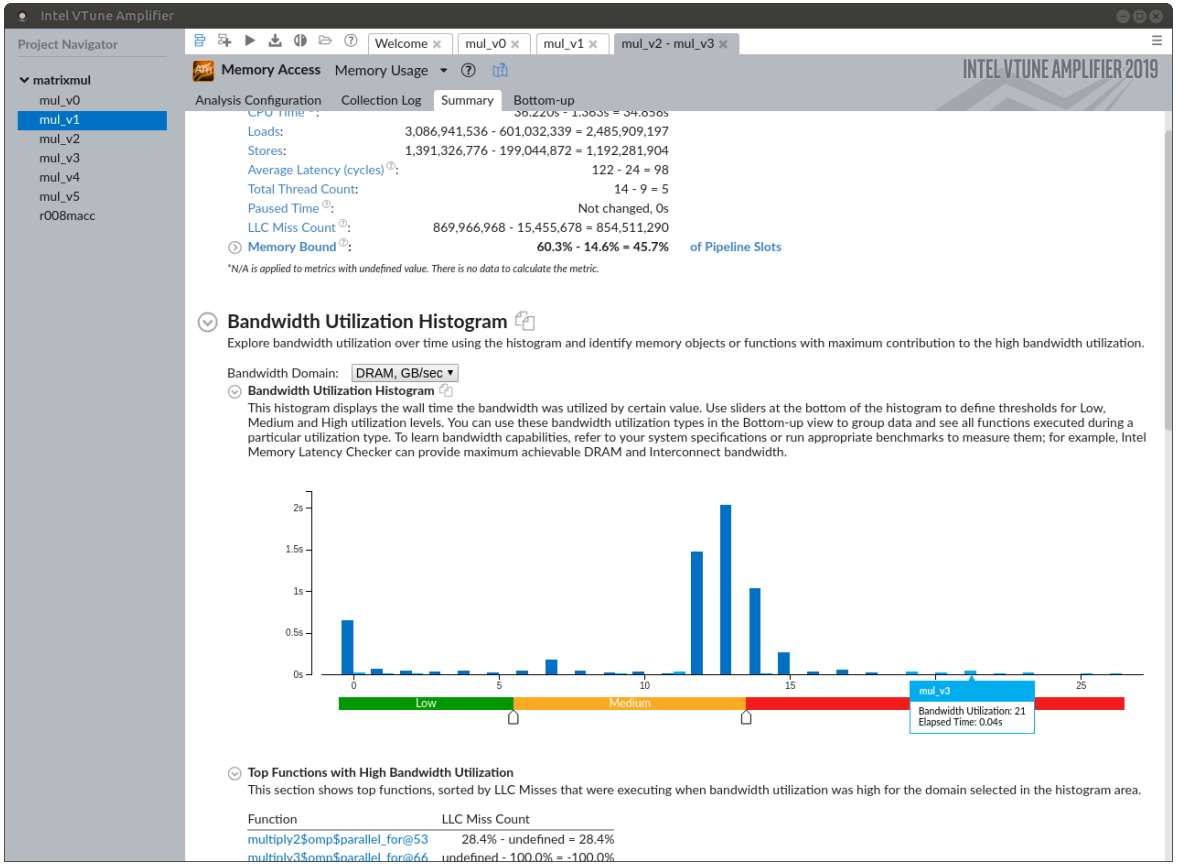
Intel VTune: multiply1





Intel VTune: multiply2 vs multiply3

- **Memory Bound** multiply2: 60 % vs multiply3: 15 %



Intel VTune: Threading Efficiency

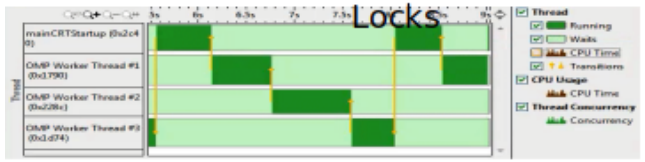
- Permite explorar
  - Wait Time: esperas prologandas por regiones de sincronización
  - Spin y Overhead Time: esperas y sobrecoste asociado al manejo de las regiones paralelas
  - Thread count: tiempo espera debido a sobresubscripción, espera a que los recursos compartidos esten disponibles cuando se ejecutan más hilos lógicos que cores físicos

Intel VTune: Threading Efficiency

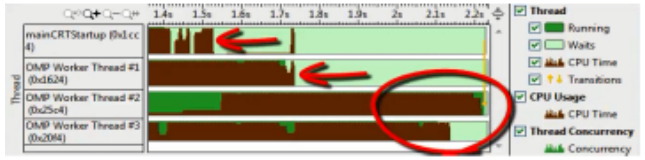
- Problemas asociados al paralelismo
  - Asegurarse que se ejecutan todos los hilos disponibles
  - Problemas comunes a la concurrencia pueden diagnosticarse
  - Análisis para detectar la contención en operaciones tipo Locks/Waits

Intel VTune: Threading Efficiency

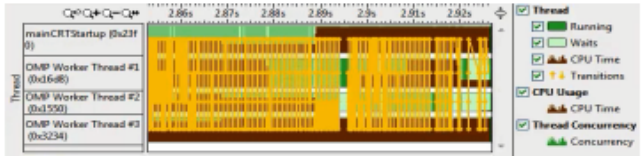
Coarse-Grain Locks



Thread Imbalance

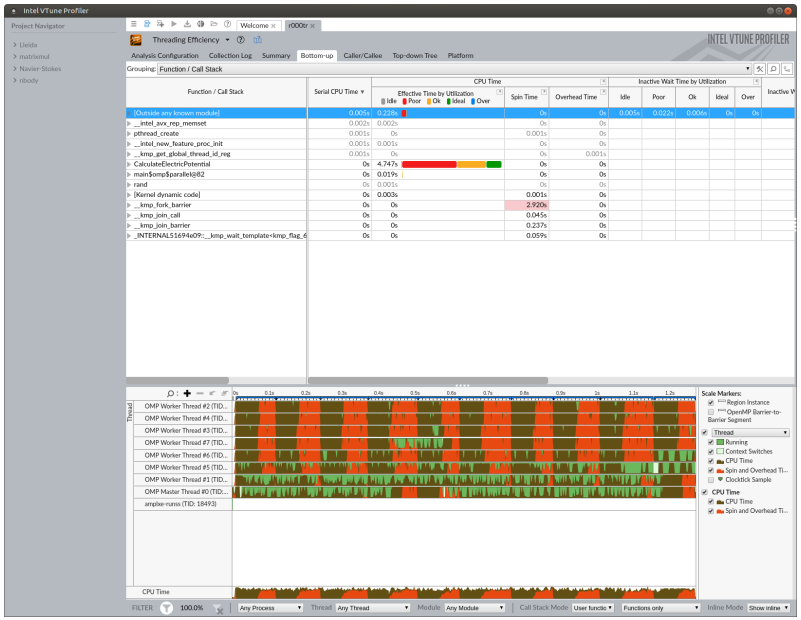


High Lock Contention



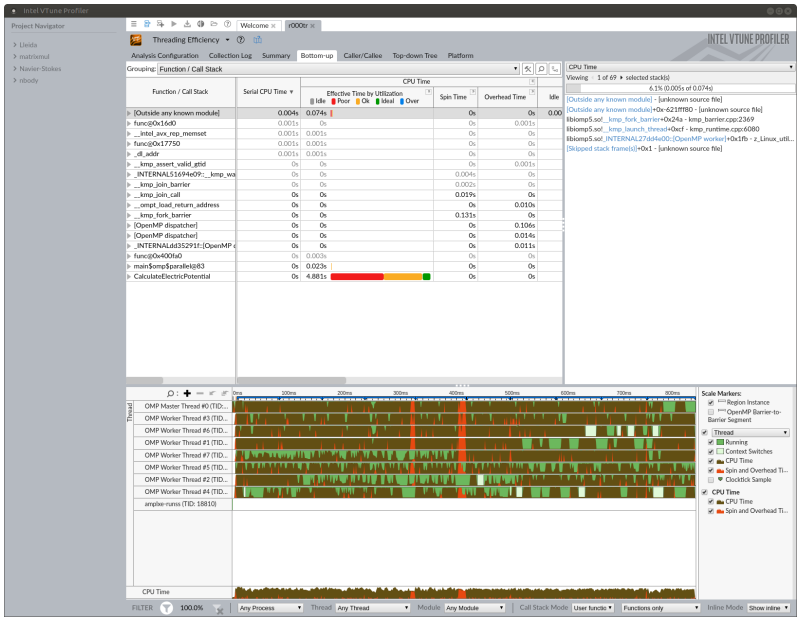
Intel VTune: Threading Efficiency

- Demo:
  - Paralelización bucle **for**
  - schedules: static, dynamic, guided
- Ej: Nbody-coulomb: static



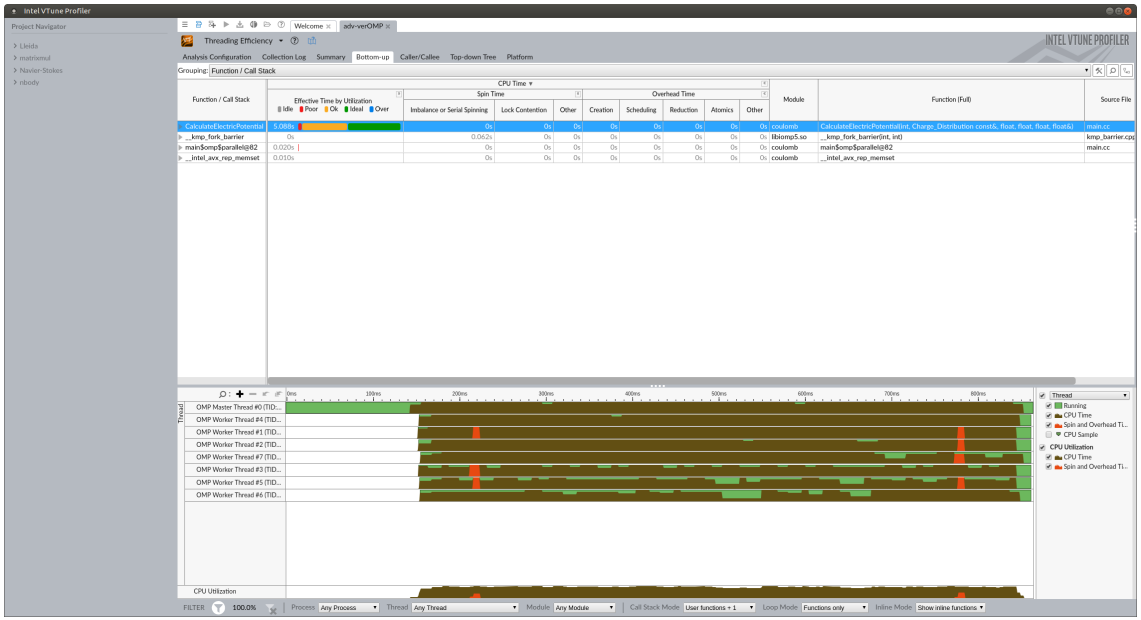
Intel VTune: Threading Efficiency

- Demo:
  - Paralelización bucle **for**
  - schedules: static, dynamic, guided
- Ej: Nbody-coulomb: dynamic



Intel VTune: Threading Efficiency

- Demo:
  - Paralelización bucle **for**
  - schedules: static, dynamic, guided
- Ej: Nbody-coulomb: guided



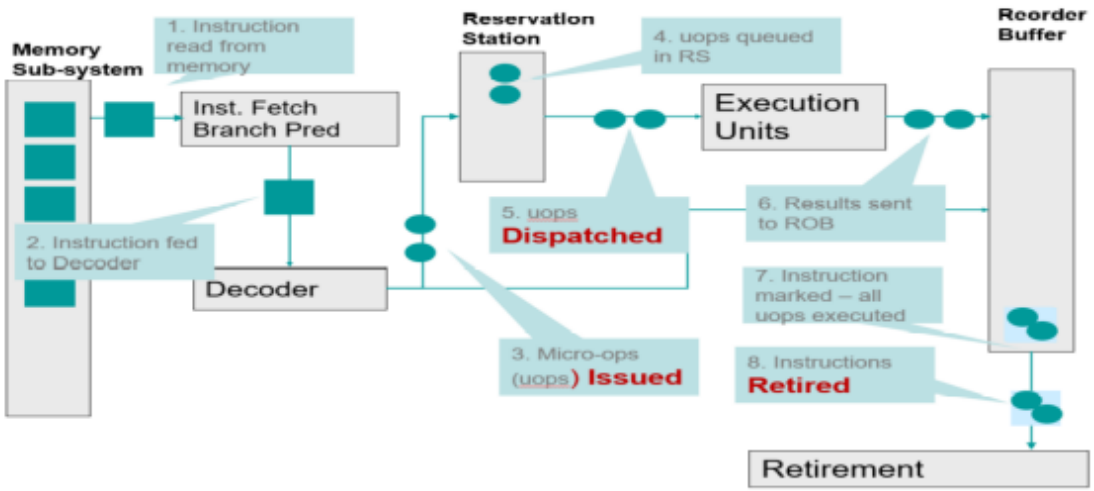
Intel VTune: Microachitecture

- **Microarchitecture analysis Group** introduce un tipo de análisis que ayuda a estimar los motivos de las ineficiencias en los procesadores modernos
  - *Microarchitecture Exploration* ayuda a identificar los problemas más communes que afectan al rendimiento de la aplicación. Se puede considerer este análisis como punto de partida al analísis a nivel hw
  - *Memory Access* mide una serie de métricas para identificar accesos a los diferentes niveles de la jerarquía de memoria (como por ejemplo en arquitecturas NUMA)

Intel VTune: Microachitecture

- Una vez completado el análisis **Hotspots** para conocer ineficiencias en tu código...
  - Se recomienda efectuar el análisis Microarchitecture Exploration analysis para comprender como las ineficiencias se manifiestan en el uso del pipeline del core
  - VTune Profiler recolecta una lista complete de eventos hw
  - Calcula unas métricas predefinidas = identifica a nivel hw problemas asociados a ineficiencias

Intel VTune: Microachitecture

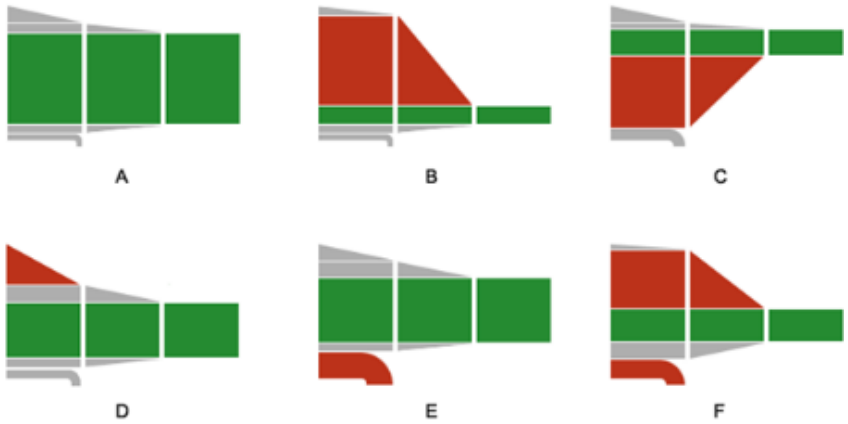


Intel VTune: Microachitecture

- Backend bound
- Uops posibles que no se puede lanzar al Fetch+Dec
- Retiring
- Uops completadas (1uop/cycle)
- Bad speculation
- Uops especuladas de forma incorrecta y no “retiradas”
- Backend bound
- Uops completadas, pero no 1 por ciclo (fallos cache)

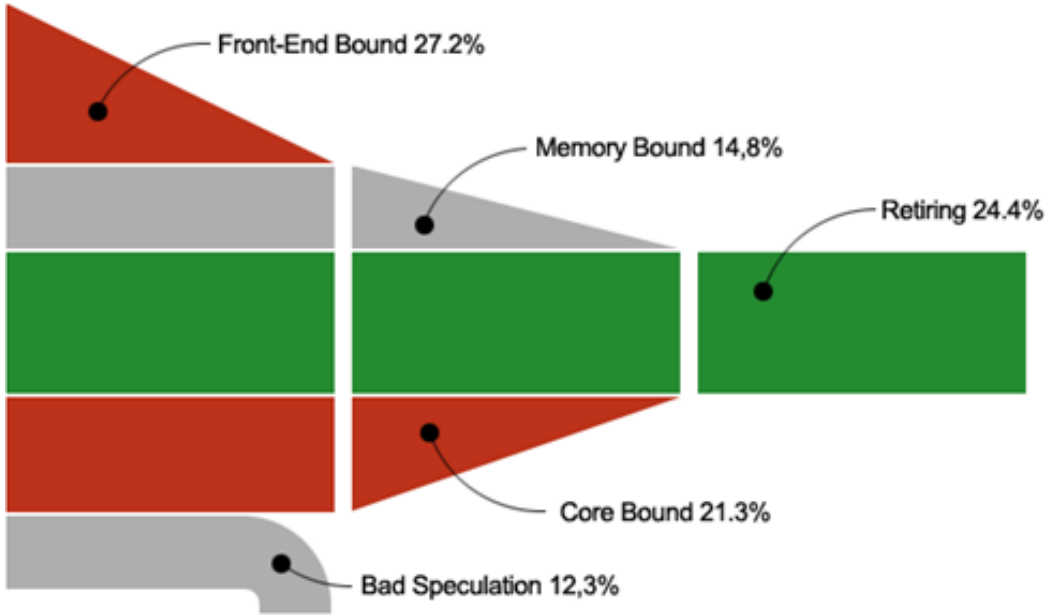
Intel VTune: Microachitecture

- A: sin problemas reseñables
- B: Memory bound
- C: Core bound
- D: Front End bound
- E: fallos en la especulación (por ejemplo *branch misprediction*)
- F: combinación de problemas relacionados con Memoria y mala Especulación



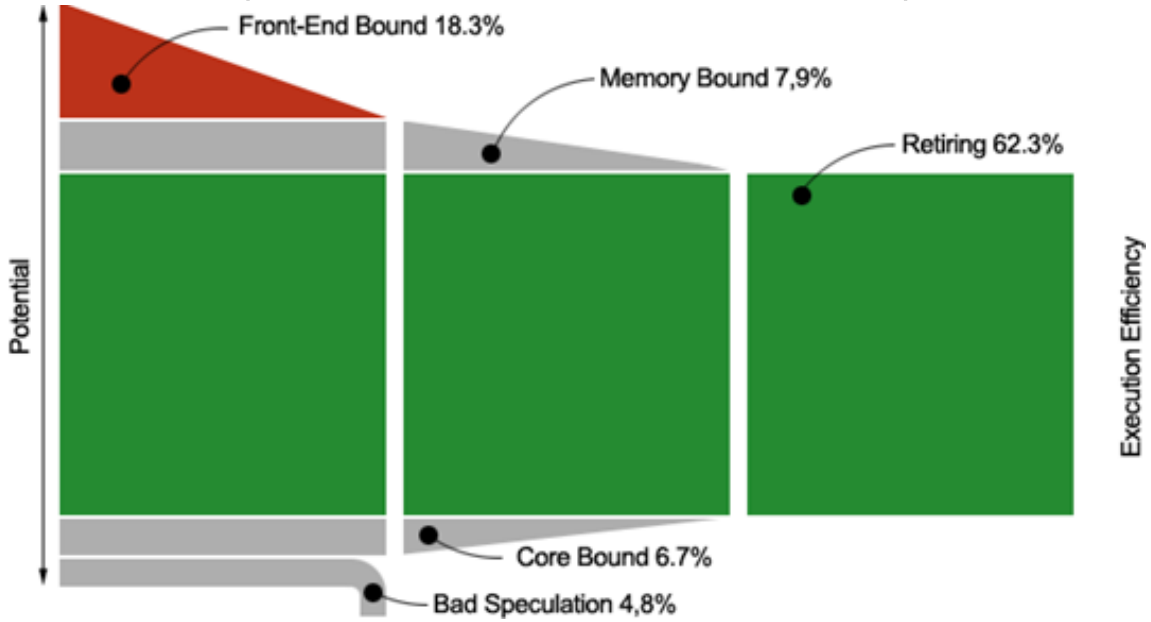
Intel VTune: Microachitecture

- Ejemplo 1
  - Pipe presenta problemas significativos en Front-End Bound y Core Bound issues limitando la eficiencia global al 24.4 %



Intel VTune: Microachitecture

- Ejemplo 2
  - Buena Eficiencia con algún problema en el Front-End (instrucciones independientes)



Intel VTune: Microachitecture