Práctica. Paralelización con MPI

Computación de Altas Prestaciones

Carlos García Sánchez

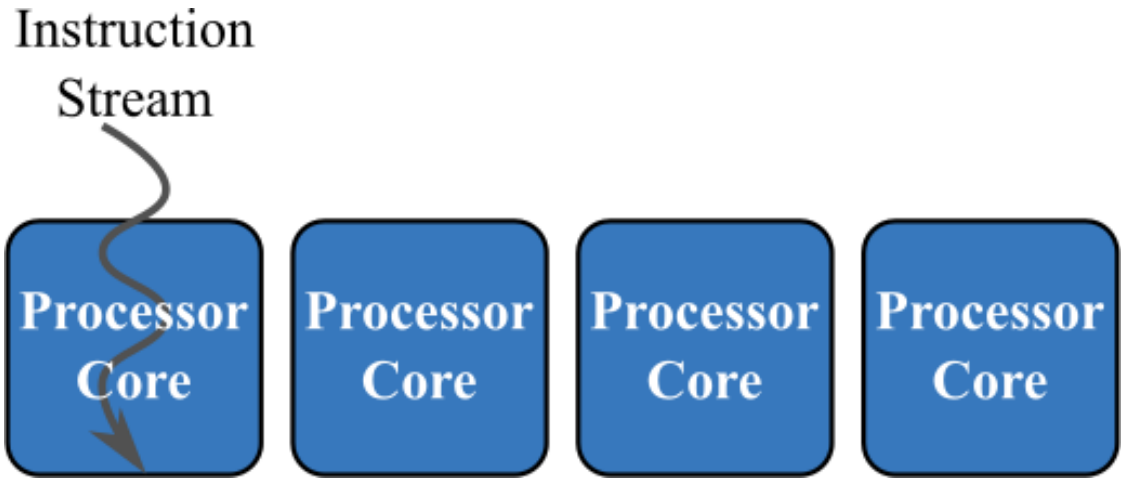
8 de noviembre de 2021

- "MPI: The Complete Reference"
- "Beginning MPI (An Introduction in C)"
- "Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition", James Jeffers, James Reinders, Avinash Sodani

Outline

Secuencial vs Paralelo

- Aplicación secuencial en sistema con varios cores

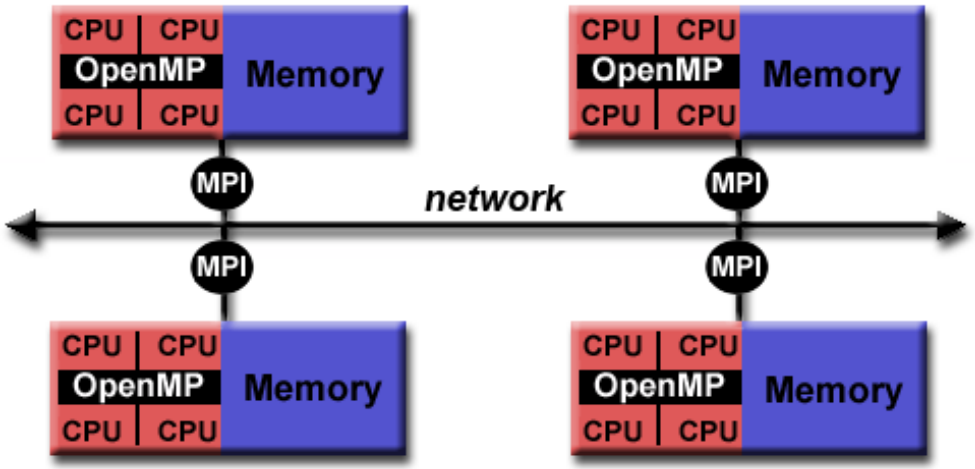


Modelos de memoria Sist. Paralelos

- Memoria compartida vs Memoria Distribuida

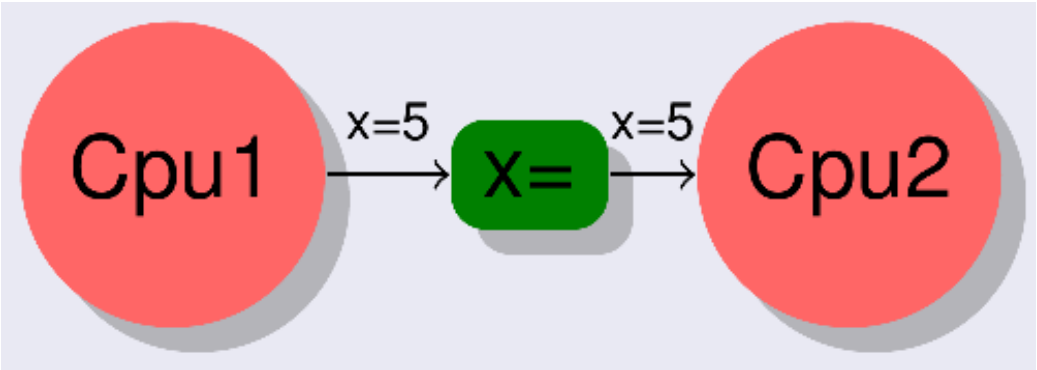
Memoria distribuida

- Estas máquinas surgen de forma natural al conectar distintas máquinas en red y ponerlas a cooperar
- Comunicación y sincronización a través de paso de mensajes (MPI): No comparten memoria
- La **red** es clave para un buen rendimiento



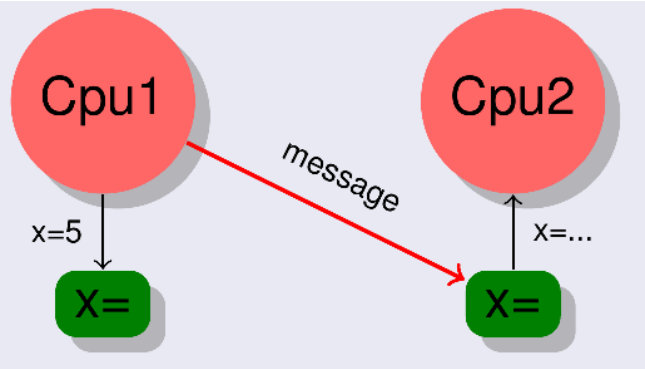
Modelos de memoria compartida

- 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



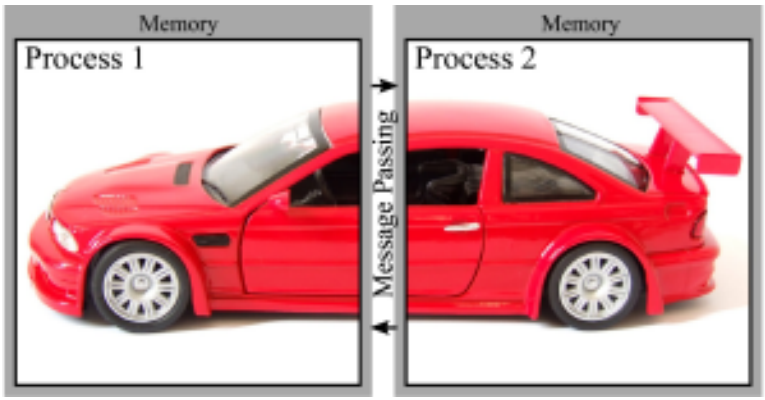
Modelos de memoria distribuida

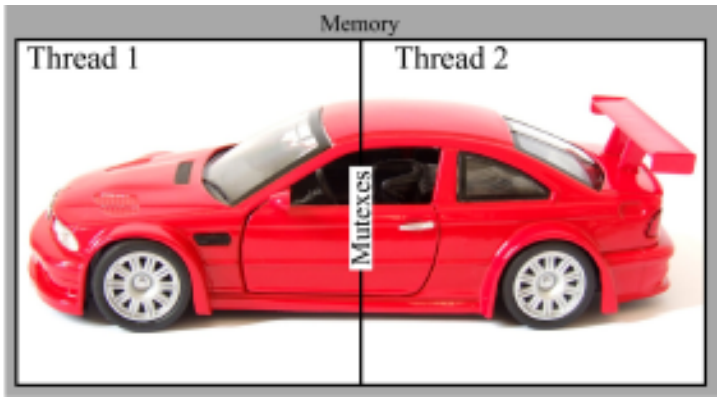
- Cada proceso tiene su propio espacio de direcciones de memoria
  - Ese espacio de memoria no es accesible por otros **procesos**
- La comunicación y sincronización se lleva a cabo explicitamente mediante mensajes



Modelos de memoria compartida vs distribuida

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





Variables de entorno

Compilador de Intel

- Compilador de Intel (icc, icpc, ifort...)
- Intel® MPI Library incluido en Intel® oneAPI HPC Toolkit [^1]: `mpiicc`, `mpiicpc`, `mpiifort`
  - GNU-GCC: `mpicc`, `mpic++`, `mpifort`

Terminal #1

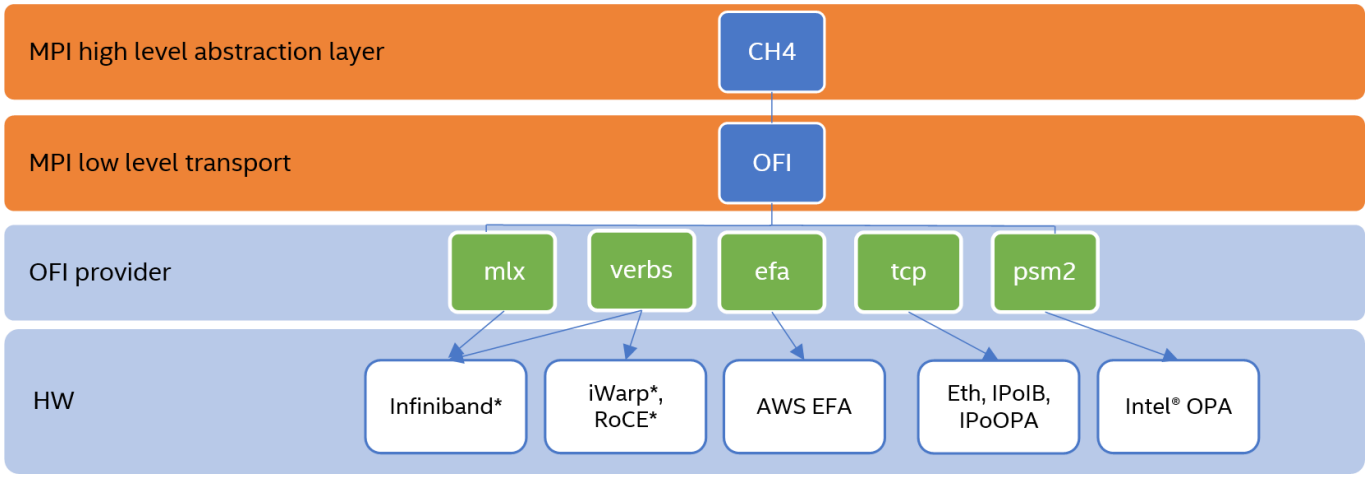
```
user@lab:~ $ source /opt/intel/oneapi/setvars.sh
:: initializing oneAPI environment ...
   bash: BASH_VERSION = 4.4.20(1)-release
:: advisor -- latest
:: ccl -- latest
:: clck -- latest
:: compiler -- latest
:: dal -- latest
:: debugger -- latest
:: dev-utilities -- latest
:: dnnl -- latest
:: dpcpp-ct -- latest
:: dpl -- latest
:: inspector -- latest
:: intelpython -- latest
:: ipp -- latest
:: ippcp -- latest
:: ipp -- latest
:: itac -- latest
:: mkl -- latest
:: mpi -- latest
:: tbb -- latest
:: vpl -- latest
:: vtune -- latest
:: oneAPI environment initialized ::
```

Librería Intel MPI

- Implementa el interfaz MPI con la especificación versión 3.1 (MPI-3.1)
- Destacamos las siguientes características
  - Escalable hasta 340K procesadores
  - Baja sobrecarga en las Comunicaciones
  - Utilidad de tunning para acelerar el desarrollo de aplicaciones
  - Independencia en la tecnología de interconexión y de los fabricantes
  - Soporte C, C++, Fortran 77, Fortan 90, Fortran 2008
- Incluido en el **Intel® oneAPI HPC Toolkit** (el kit también incluye Intel Traze Analyzer/Collector e Intel Cluster Checker)

Librería Intel MPI

- Automáticamente selecciona la mejor red disponible en el sistema
- Abstracción a bajo nivel para redes de alto rendimiento mediante libfabric
  - I_MPI_OFI_PROVIDER selecciona el proveedor
  - Ethernet (tcp), Intel Omni-Path (psm2), Infiniband (mlx), AWS EFA (efa)



Ejemplo 0

- *Hello World*

hello.c

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\
        n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

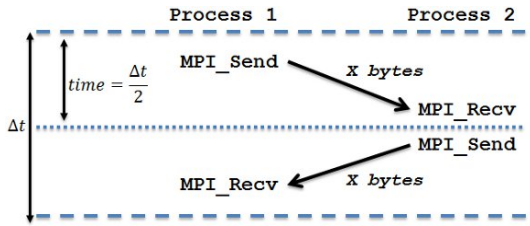
Ejemplo 0

Terminal #1

```
carlos@posets:$ mpicc -o hello hello.c
carlos@posets:$ mpirun -np 4 ./main
Hello world from processor 7picos, rank 0 out of 4 processors
Hello world from processor 7picos, rank 1 out of 4 processors
Hello world from processor 7picos, rank 2 out of 4 processors
Hello world from processor 7picos, rank 3 out of 4 processors
```

Ejemplo 1: ping-pong

- MPI_Send & MPI_Recv



pingpong.c

```
int main( int argc, char *argv[] ) {
    int rank;
    int size;
    int N;

    setbuf (stdout, NULL);

    MPI_Init (&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    N = atoi(argv[1]);

    if (rank == 0) ping(N);
    else pong(N);

    MPI_Finalize();

    return 0;
}
```

Ejemplo 1: ping-pong

ping.c

```
void ping(int N) {
    MPI_Status status;

    double *buffer_send = (double*)malloc(N*sizeof(double));
    double *buffer_recv = (double*)malloc(N*sizeof(double));


    for (int i=0; i<N; i++) buffer_send[i]=i;

    double t1 = MPI_Wtime();
    MPI_Send(...);
    MPI_Recv(...);
    double t2 = MPI_Wtime();

    for (int i=0; i<N; i++)
        if (buffer_send[i]!=buffer_recv[i]) {
            printf("ERROR in COMM: buffers differs\n");
            break;
        }

    printf("Ping pong done in %f secs.\n", t2-t1);

    free(buffer_send);
    free(buffer_recv);
}
```

Ejemplo 1: ping-pong

pong.c

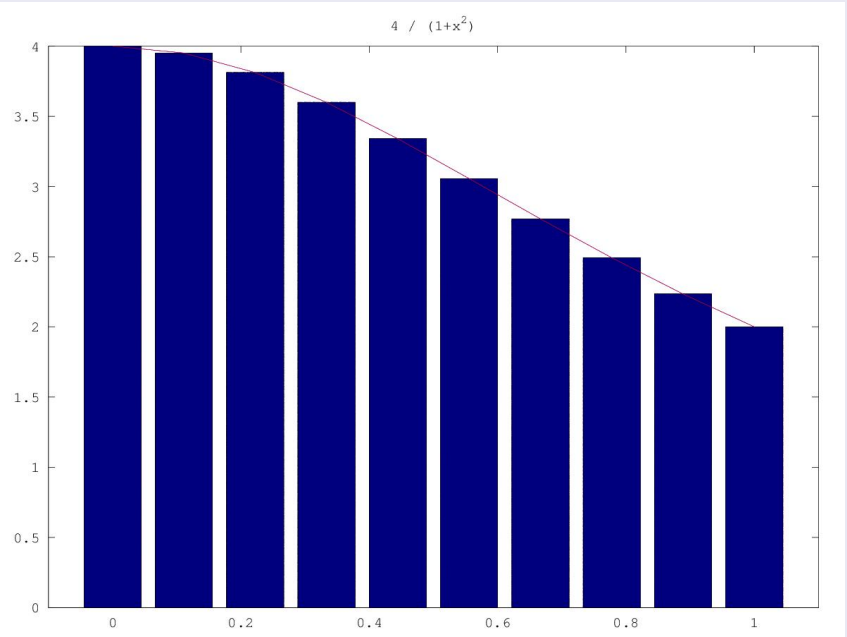
```
void pong(int N) {
    MPI_Status status;
    double *buffer = (double*)malloc(N*sizeof(double));

    MPI_Recv(...);
    MPI_Send(...);

    free(buffer);
}
```

Ejemplo 2

- Cálculo de PI

Suma de rectángulos

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$
$$\pi \approx \sum_{i=0}^{N} F(x_i)\Delta x$$



Ejemplo 2

piserial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char* argv[])
{
   double pi, exactpi;
   double x, f, area;
   int N;

   if (argc!=2) {
      printf("Execution: ./exec N\n");
      exit(-1);
   } else N = atoi(argv[1]);


   printf("Computing approximation to pi using N = %d\n", N);

   area = 0.0;

   for (int i=0; i<N; i++)
   {
      x = (i+0.5)/N;
      f = 4.0/(1.0 + x*x);
      area = f*(1.0/N); //F(x)*deltax
      pi+=area;
   }


   exactpi = 4.0*atan(1.0);

   printf("pi = %f, %% error = %e\n", pi, fabs(100.0*(pi-exactpi)/
        exactpi));

   return 0;
}
```

Ejemplo 2

- Paralelización: **bucle for**
- Distribución de bloques de iteraciones
  - chunck=N/nprocs
  - Ej: 2 procs:
    - rank=0 realizaría i=0 a N/2
    - rank=1 realizaría i=N/2 a N
- Reducción de las *pi_local* en **pi**

piparallel.c

```
...
pi_local = 0.0;

for (int i=?; i<?; i++)
{
    x = (i+0.5)/N;
    f = 4.0/(1.0 + x*x);
    area = f*(1.0/N); //F(x)*deltax
    pi_local += area;
}

// REDUCE pi_local
MPI_Reduce(...);
...
```

Ejemplo 3: Deadlocks

- Ejemplo **deadlock.c**
  - “‘sendfirst=?¿“‘si es 1 ambos envían primero

deadlock.c

```
/* now for the comms */
if (sendfirst == 1) {
  printf("process %d of %d. Sending..\n",rank,size);
  /* first send.. */
  MPI_Ssend(message, BUFSIZ, MPI_CHAR, other, tag, MPI_COMM_WORLD);
  /* ..then receive */
  MPI_Recv(message, BUFSIZ, MPI_CHAR, other, tag ,MPI_COMM_WORLD, &
      status);
  /*
  ** We _may_ get here..
  ** but only if the blocking function MPI_Send() has access to a
      system buffer
  */
  printf("process %d of %d. Received.\n",rank,size);
} else {
  printf("process %d of %d. Receiving..\n",rank,size);
  MPI_Recv(message, BUFSIZ, MPI_CHAR, other, tag, MPI_COMM_WORLD, &
      status);
  MPI_Ssend(message, BUFSIZ ,MPI_CHAR, other, tag, MPI_COMM_WORLD);
  /*
  ** The blocking function MPI_Recv() cannot return and so we cannot
      get here..
  */
  printf("We'll never get here!\n");
}
```

Ejemplo 3: Deadlocks

- Ejemplo **deadlock_tag.c**
  - tags de MPI_Ssend y MPI_Revc no se pueden emparejar

deadlock_tag.c

```
/* Call ping-pong functions */
if (rank == 0) {
   MPI_Ssend(buffer1, length, MPI_FLOAT, 1, tag1,
     MPI_COMM_WORLD);

   MPI_Recv(buffer2, length, MPI_FLOAT, 1, tag2,
       MPI_COMM_WORLD, &status);

   printf("pingpong performed\n");

   MPI_Ssend(buffer1, length, MPI_FLOAT, 1, tag1,
     MPI_COMM_WORLD);

   MPI_Ssend(buffer2, length, MPI_FLOAT, 1, tag2,
     MPI_COMM_WORLD);
}
else if (rank == 1) {

   MPI_Recv(buffer1, length, MPI_FLOAT, 0, tag1,
       MPI_COMM_WORLD, &status);

   MPI_Ssend(buffer2, length, MPI_FLOAT, 0, tag2,
     MPI_COMM_WORLD);

   MPI_Recv(buffer1, length, MPI_FLOAT, 0, tag2,
       MPI_COMM_WORLD, &status);

   MPI_Recv(buffer2, length, MPI_FLOAT, 0, tag1,
       MPI_COMM_WORLD, &status);

}
```

Ejemplo 4: ecuación del calor

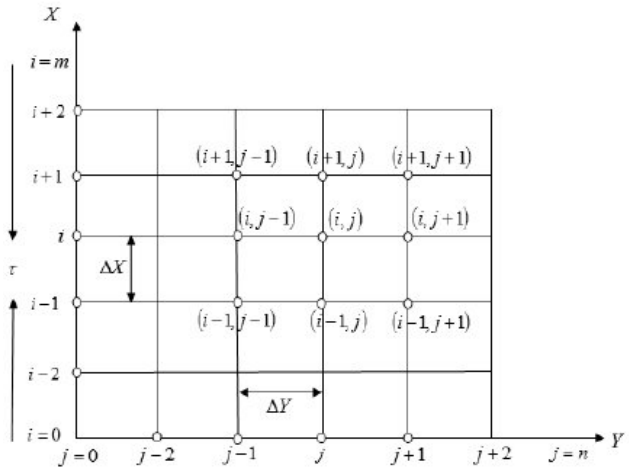
- Resolución Numérica de Problemas de Transmisión de Calor

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 4: 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 4: 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 4: ecuación del calor

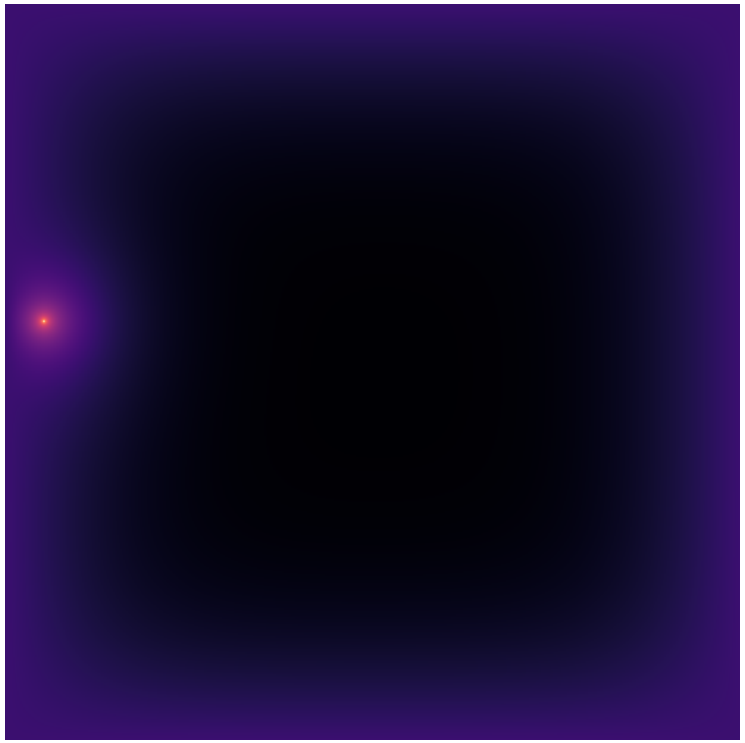
- Inicializa con un valor rand las coordenadas de la fuente de calor: `source_x, source_y`
- Inicializa las condiciones de contorno
- Ejecución del bucle principal `(it < MAX_ITERATIONS) && (t_diff > MIN_DELTA)`
- Salida en fichero .png que muestra el calor en una placa 2D

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 4: ecuación del calor

- Tras 20000 iteraciones



Ejemplo 34: ecuación del calor

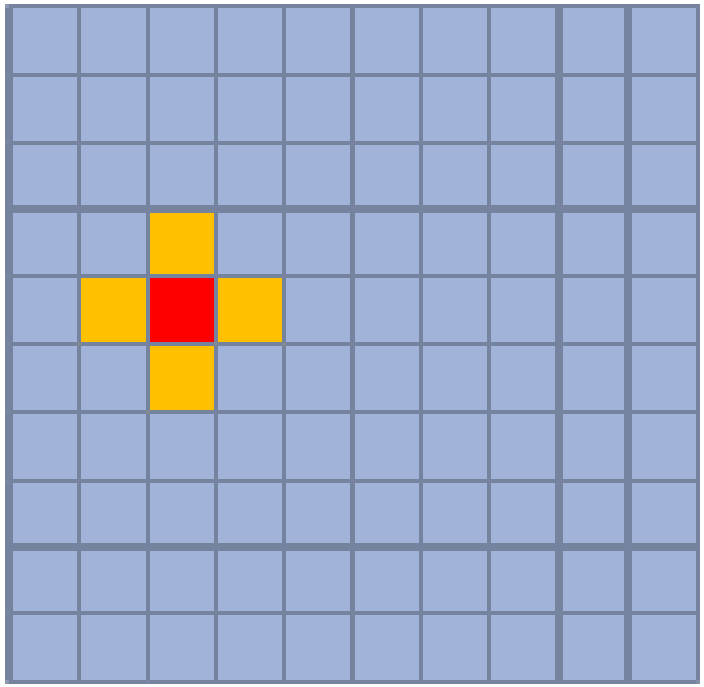
- Las rutinas a paralelizar son **step** y **diff**
- **step**: esquema paralelización basado en **descomposición de dominios**
- **diff**: reducción del *maxdiff*

heat_core.c

```
static void step(unsigned int source_x, unsigned int source_y, const
     float * current, float * next) {

   for (unsigned int y = 1; y < N-1; ++y) {
      for (unsigned int x = 1; x < N-1; ++x) {
         if ((y == source_y) && (x == source_x)) {
            continue;
         }
         next[idx(x, y, N)] = (current[idx(x, y-1, N)] +
         current[idx(x-1, y, N)] +
         current[idx(x+1, y, N)] +
         current[idx(x, y+1, N)]) / 4.0f;
      }
   }
}

static float diff(const float * current, const float * next) {
   float maxdiff = 0.0f;
   for (unsigned int y = 1; y < N-1; ++y) {
      for (unsigned int x = 1; x < N-1; ++x) {
         maxdiff = fmaxf(maxdiff, fabsf(next[idx(x, y, N)] - current[
              idx(x, y, N)]));
      }
   }
   return maxdiff;
}
```

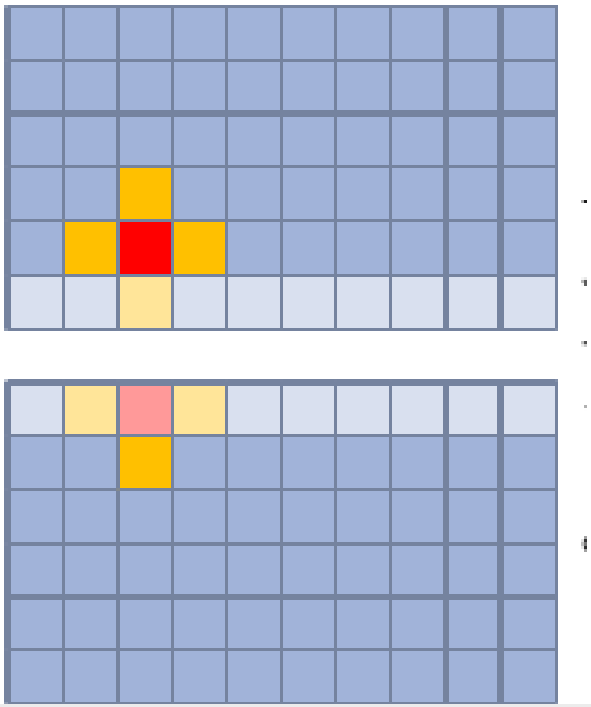
Ejemplo 4: ecuación del calor

- Lo más común es aplicar una descomposición interdependiente de los datos en diferentes procesos



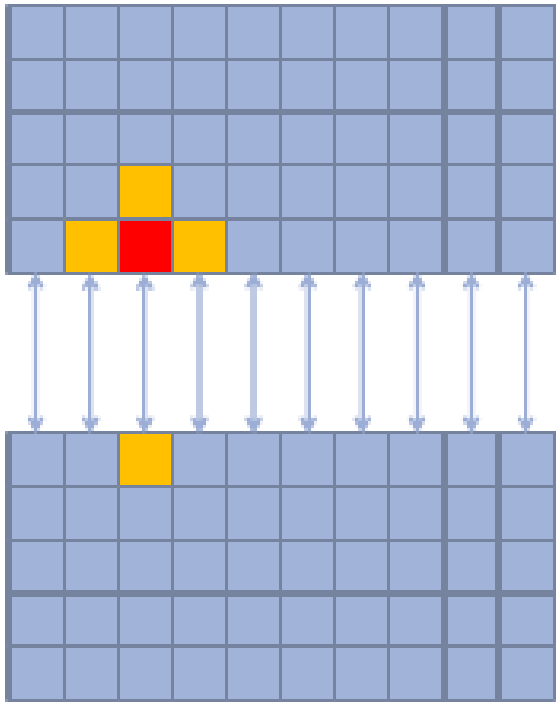
Ejemplo 4: ecuación del calor

- Lo más común es aplicar una descomposición interdependiente de los datos en diferentes procesos
  - Pero **no parece eficiente** tener que comunicar cada dato cada vez que procesa una celda



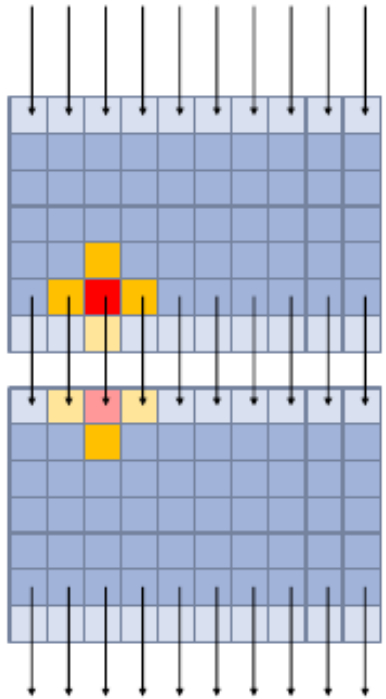
Ejemplo 4: ecuación del calor

- Lo más común es aplicar una descomposición interdependiente de los datos en diferentes procesos
  - Es más conveniente añadir una filas de celdas “ghost” y comunicar las fronteras



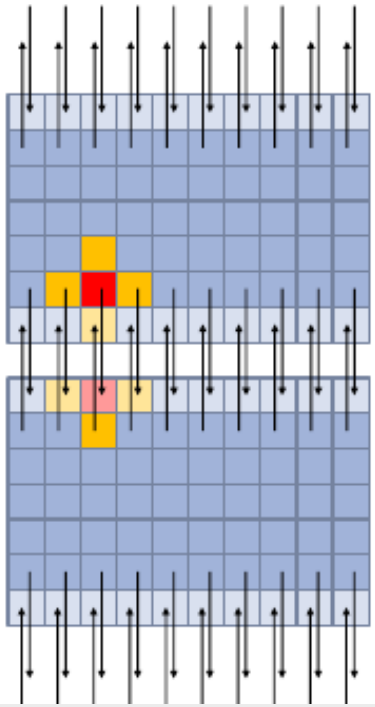
Ejemplo 4: ecuación del calor

- Lo más común es aplicar una descomposición interdependiente de los datos en diferentes procesos
  - Es más conveniente añadir una filas de celdas “ghost” y comunicar las fronteras en cada iteración



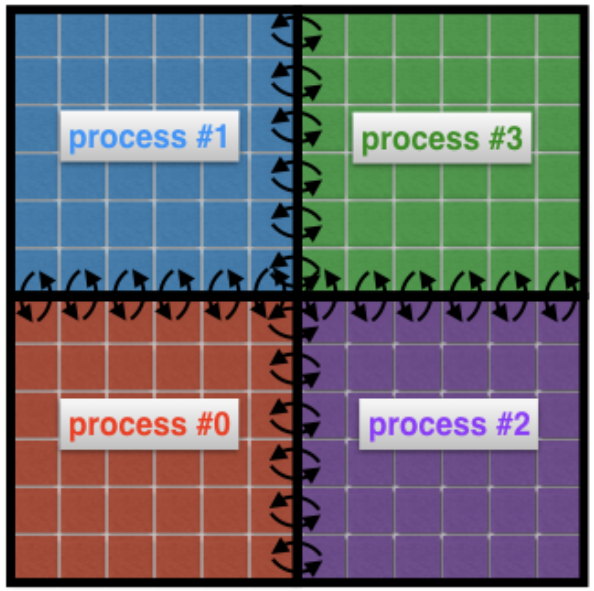
Ejemplo 4: ecuación del calor

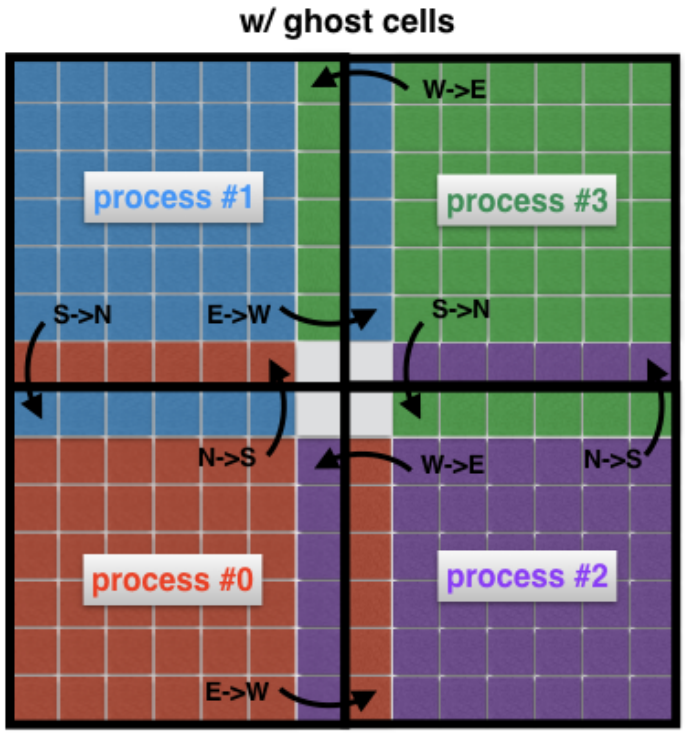
- Lo más común es aplicar una descomposición interdependiente de los datos en diferentes procesos
  - Más que comunicar, es converniente **intercambiar las celdas ghost**: MPI_SEND+MPI_RECV



Ejemplo 5: ecuación del calor (Opcional)

- Aplicar una descomposición de dominios 2D
  - NOTA: Para el intercambio de columnas crear una estructura del tipo MPI_VECTOR





Análisis y tunning

- La librería Intel® MPI permite controlar la información de depuración con la variable de entorno **I_MPI_DEBUG**

Terminal #1

```
user@lab:$  mpirun  -genv I_MPI_DEBUG=2 -n 2 ./testc
-genv I_MPI_DEBUG=2 -n 2 testc[1] MPI startup(): Internal info: pinning initialization was done[0]
```

- También permite conocer el "tiempo" en el que se produce cada uno de los eventos

Terminal #1

```
user@lab:$ mpirun -genv I_MPI_DEBUG=2,time,norank -n 2 ./testc
11:59:59 MPI startup(): Multi-threaded optimized library
```

- O incluso redirigir la salida de la depuración con la variable de entorno **I_MPI_DEBUG_OUTPUT**

Terminal #1

```
user@lab:$ mpirun -genv I_MPI_DEBUG=2 -genv I_MPI_DEBUG_OUTPUT=/tmp/debug_output.txt -
n 2 ./testc
```

Distribución de procesos MPI

- Información de ejecución con **I_MPI_DEBUG**
- Imprime información de depuración cuando el programa MPI comienza
- Valores entre 1 y 6 con niveles de detalla (0: no debug, 6: máximo)
- Nivel razonable, I_MPI_DEBUG=4, imprime información sobre:
  - Process pinning
  - Interfaz de red usado
  - Variables de entorno seleccionadas en Intel MPI Library
- Más info en Developer Reference[1]

[1] MPI Library Developer Reference:
`https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top.html`

Herramienta de análisis

- El **Intel® Trace Analyzer**[2] permite perfilar la ejecución de la biblioteca de MPI
  - Intercepta las llamadas a la biblioteca MPI y genera un fichero de traza
  - Dicho fichero se puede analizar con la herramienta Intel® Trace Analyzer

[2] Intel® Trace Analyzer User and Reference Guide
`https://software.intel.com/content/www/us/en/develop/documentation/ita-user-and-reference-guide/top.html`

Herramienta de análisis

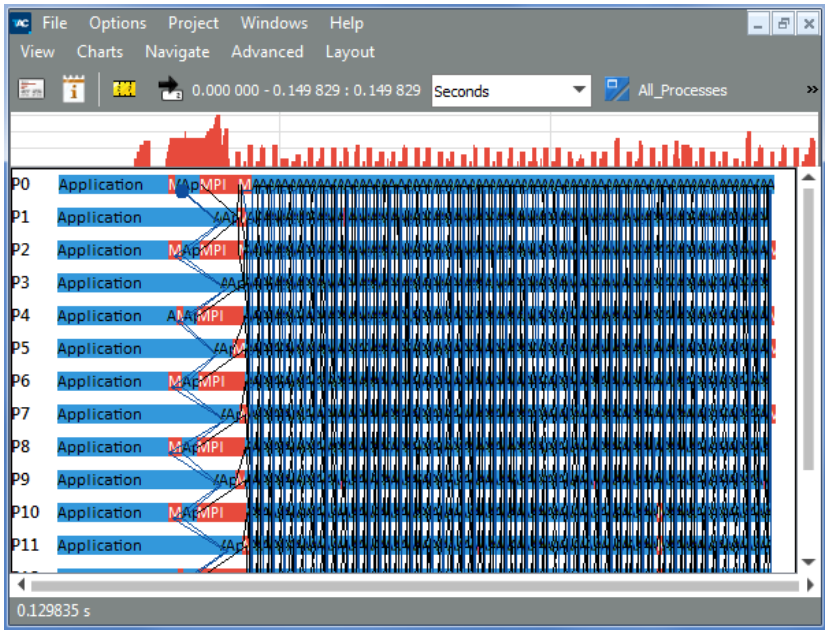
- El **Intel® Trace Analyzer** permite perfilar la ejecución de la biblioteca de MPI
- Pasos
  1. Generar la traza en la invocación del `mpirun` con la opción `-trace (-t)`
     - Tras la ejecución un fichero .stf se genera
  2. Con la herramienta `tracealalyzer` se puede visualizar la traza

Terminal #1

```
user@lab:$ mpirun -trace -n 4 ./myprog
user@lab:$ traceanalyzer ./myprog.stf &
```

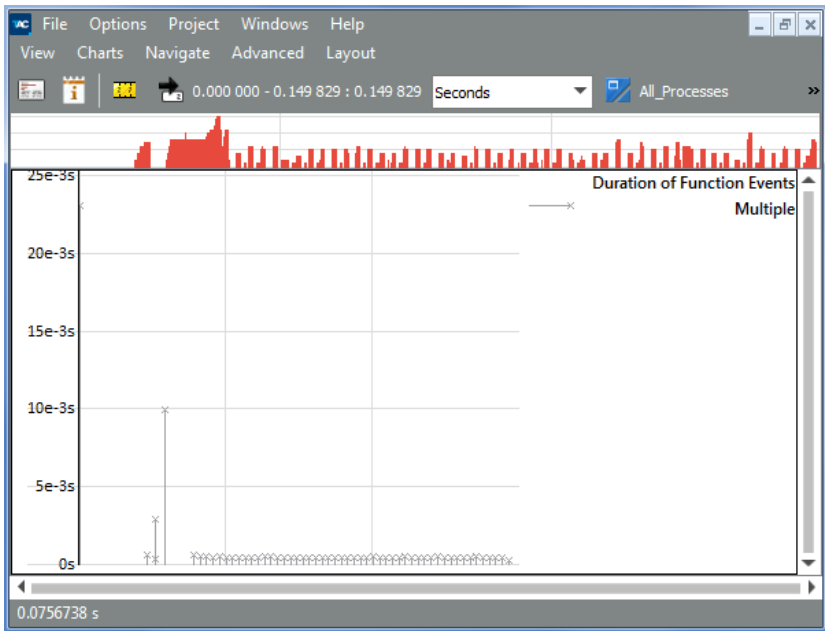
Herramienta de análisis (gráficos)

- En *Charts* > *Event Timeline*
  - Barras horizontales: llamadas a los procesos
  - Líneas negras: mensajes enviados entre procesos
  - Líneas azules: representan operaciones colectivas



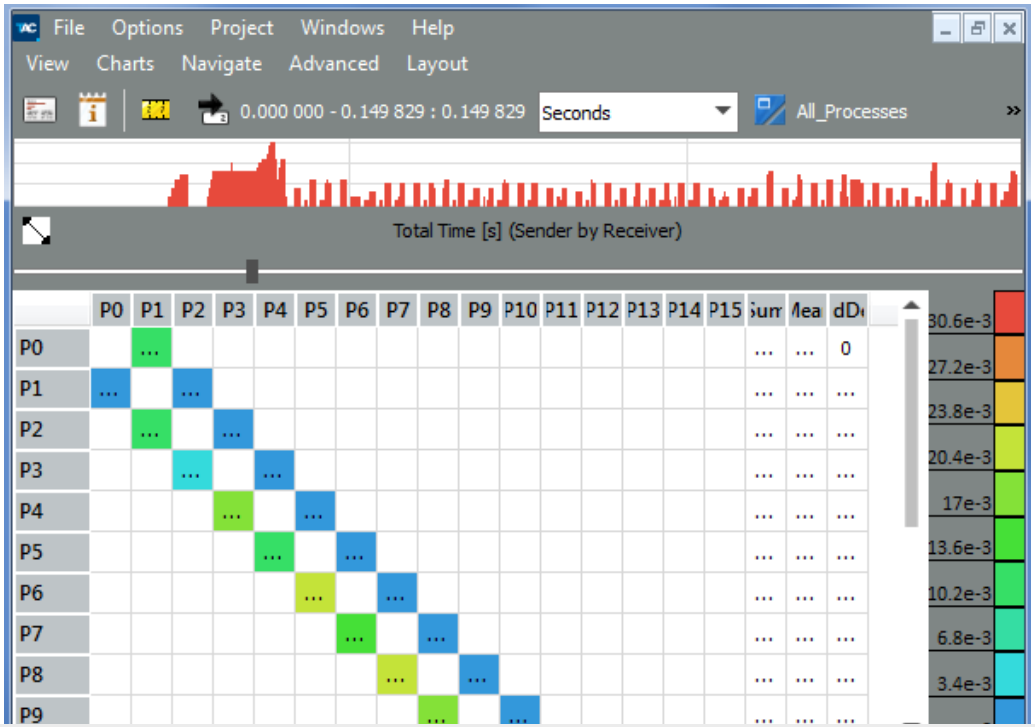
Herramienta de análisis (gráficos)

- En *Charts* > *Qualitative Timeline*
  - Representa el volumen de datos de los mensajes a lo largo del tiempo



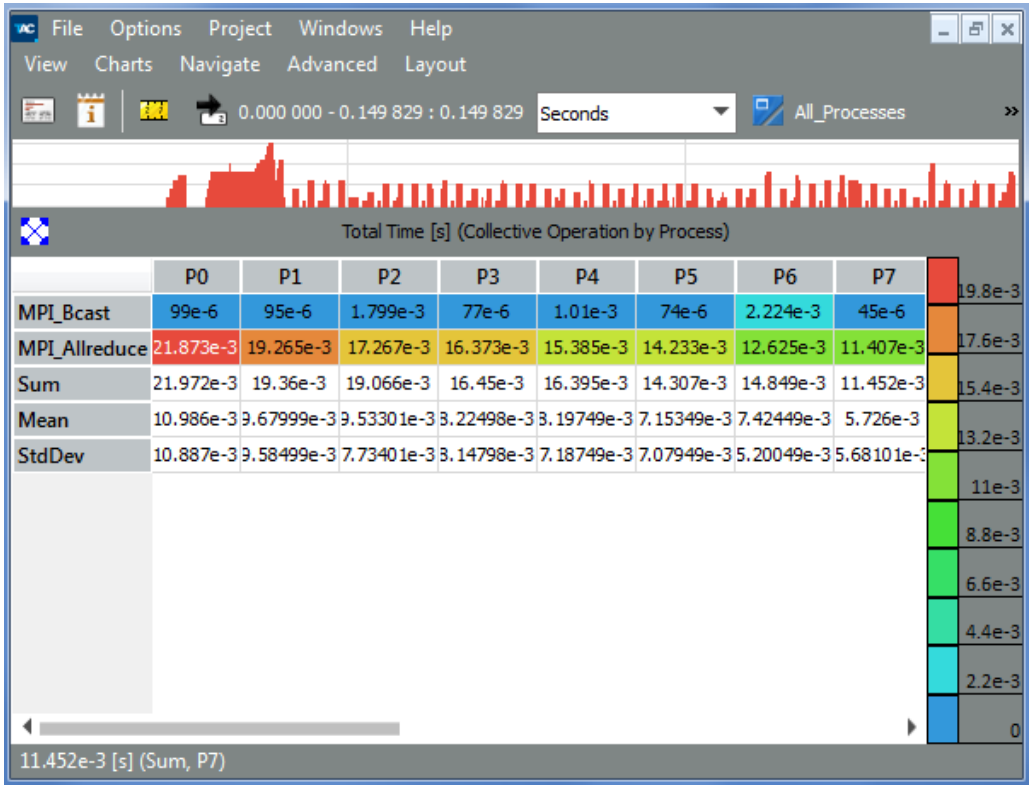
Herramienta de análisis (gráficos)

- En *Charts* > *Message Profile*
  - Clasifica los mensajes por agrupaciones
  - Muestra el tiempo total empleado en transferir mensajes del remitente i al receptor j

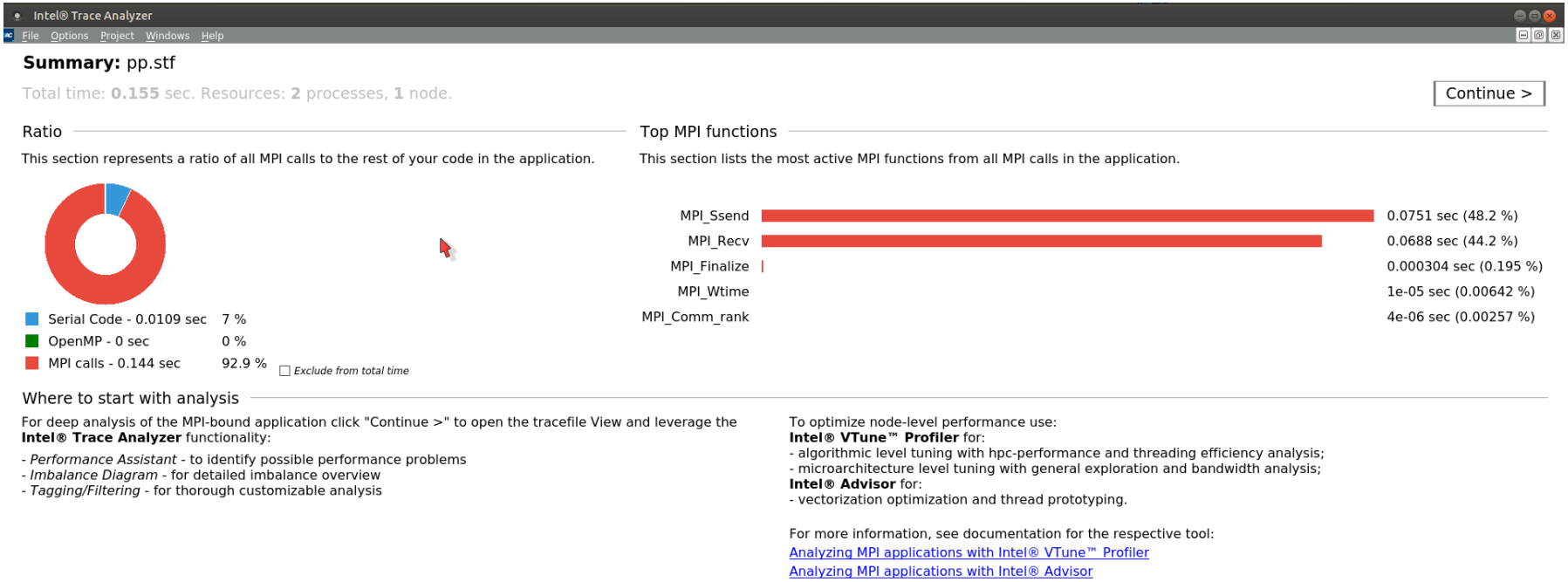


Herramienta de análisis (gráficos)

- En *Charts* > *Collective Operations Profile*
  - Muestra el perfil de las comunicaciones colectivas

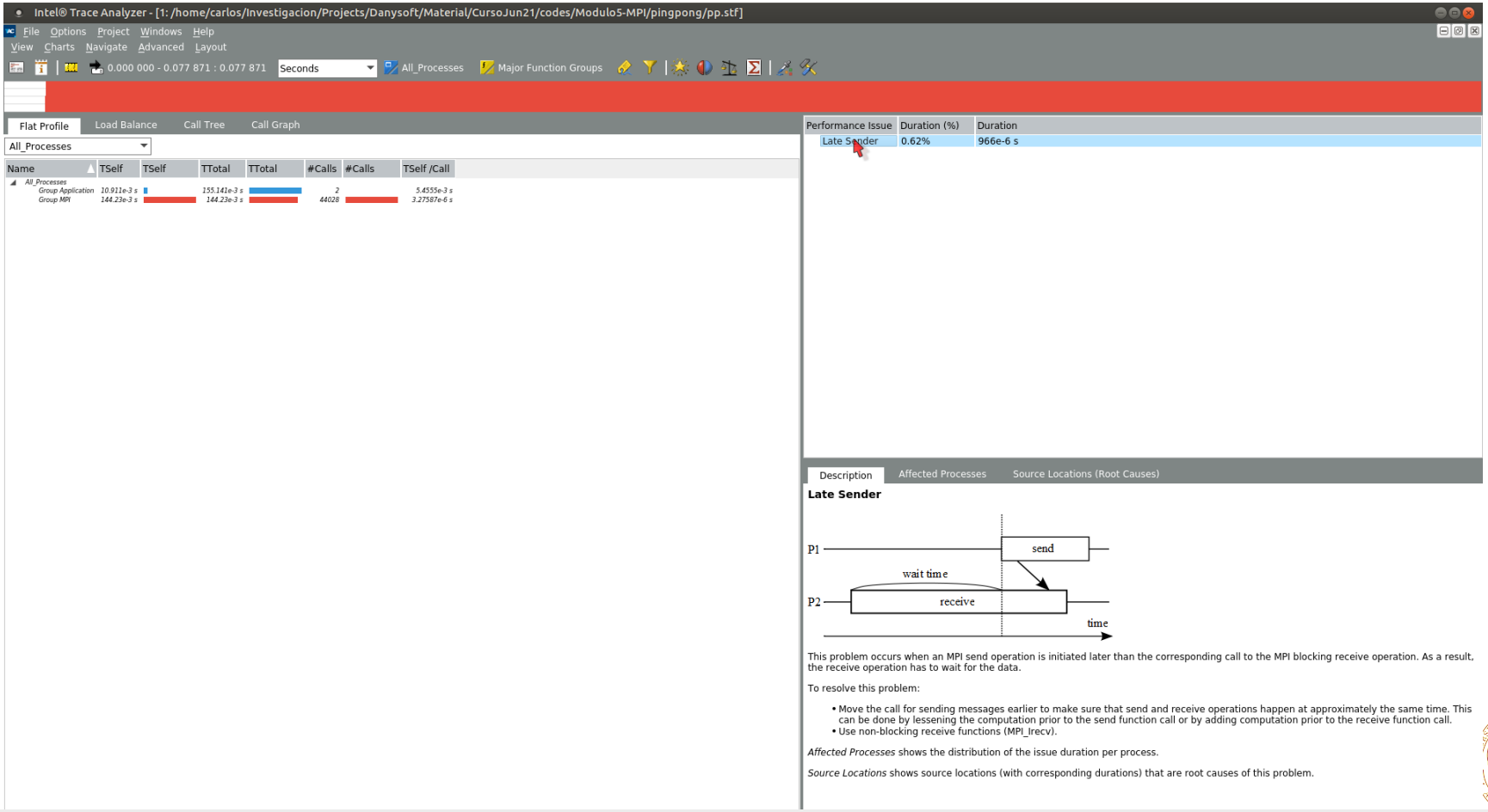


Herramienta de análisis (Ejemplo 1: ping-pong)



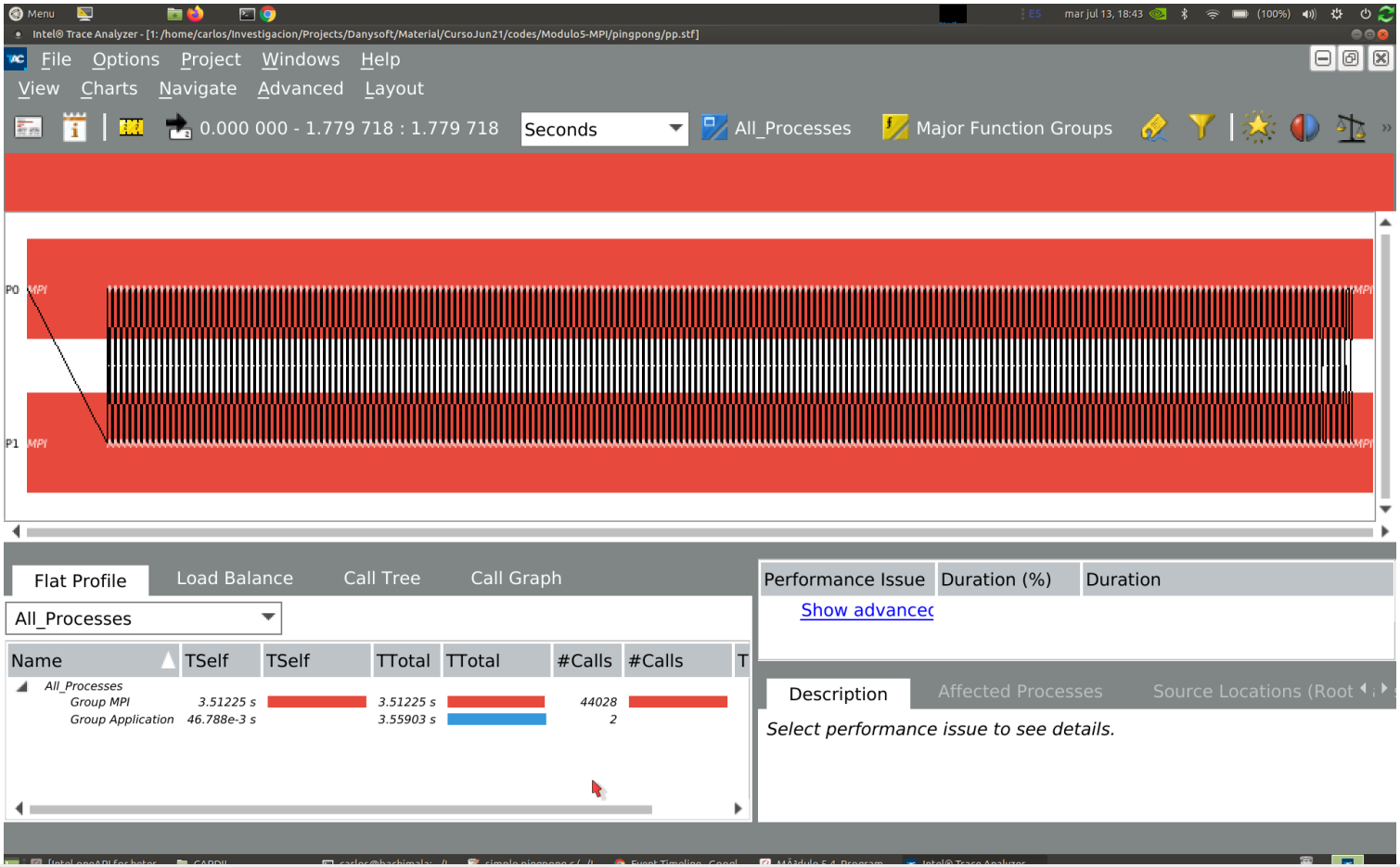
Herramienta de análisis (Ejemplo 1: ping-pong)

- En *Charts->Performance Assistant*



Herramienta de análisis (Ejemplo 1: ping-pong)

- En *Charts* > *Message Profile*



Detección Deadlocks (Ejemplo 3: deadlocks)

- La herramienta *checker library (VTmc.so)* permite lincar dinámica la ejecución de un programa MPI para evaluar posibles deadlocks[3]
  - `-check_mpi` dinámicamente linca con la librería de verificación de corrección (VTmc.so)
  - `-genv VT_CHECK_TRACING on` activa la escritura en el fichero de traza .stf para su posterior análisis con Intel® Trace Analyzer
  - `-genv VT_DEADLOCK_TIMEOUT 20s` si no hay progreso pasado el TIMEOUT escribe la traza y aborta la ejecución (asume DEADLOCK)
  - `-genv VT_DEADLOCK_WARNING 25s` muestra un warning pasado el DEADLOCK (puede producirse desbalanceo de carga o deadlock)

[3] Herramienta de chequeo de funcionamiento correcto

Detección Deadlocks

- Uso de la herramienta *checker library (VTmc.so)*

Terminal #1

```
user@lab:$ mpiicc -o deadlock deadlock.c -g
user@lab:$ mpirun  -check_mpi -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -
genv VT_DEADLOCK_WARNING 25s -genv VT_PCTRACE on -n 2 ./deadlock
```

Detección Deadlocks

- Ejemplo **deadlock.c**
  - “‘sendfirst=?¿“‘si es 1 ambos envían primero

deadlock.c

```
/* now for the comms */
if (sendfirst == 1) {
  printf("process %d of %d. Sending..\n",rank,size);
  /* first send.. */
  MPI_Ssend(message, BUFSIZ, MPI_CHAR, other, tag, MPI_COMM_WORLD);
  /* ..then receive */
  MPI_Recv(message, BUFSIZ, MPI_CHAR, other, tag ,MPI_COMM_WORLD, &
       status);
  /*
  ** We _may_ get here..
  ** but only if the blocking function MPI_Send() has access to a
       system buffer
  */
  printf("process %d of %d. Received.\n",rank,size);
} else {
  printf("process %d of %d. Receiving..\n",rank,size);
  MPI_Recv(message, BUFSIZ, MPI_CHAR, other, tag, MPI_COMM_WORLD, &
       status);
  MPI_Ssend(message, BUFSIZ ,MPI_CHAR, other, tag, MPI_COMM_WORLD);
  /*
  ** The blocking function MPI_Recv() cannot return and so we cannot
       get here..
  */
  printf("We'll never get here!\n");
}
```

Detección Deadlocks

- Ejemplo **deadlock.c**

Terminal #1

```
user@lab:$ mpirun -check_mpi -genv VT_DEADLOCK_TIMEOUT 20s -n 2 ./deadlock
....

process 0 of 2. Receiving..
process 1 of 2. Receiving..
[0] ERROR: no progress observed in any process for over 0:20 minutes, aborting application
[0] WARNING: starting premature shutdown

[0] ERROR: GLOBAL:DEADLOCK:HARD: fatal error
[0] ERROR:    Application aborted because no progress was observed for over 0:20 minutes,
[0] ERROR:    check for real deadlock (cycle of processes waiting for data) or
[0] ERROR:    potential deadlock (processes sending data to each other and getting blocked
[0] ERROR:    because the MPI might wait for the corresponding receive).
[0] ERROR:    [0] no progress observed for over 0:20 minutes, process is currently in MPI call:
[0] ERROR:       MPI_Recv(*buf=0x7ffde9f59670, count=8192, datatype=MPI_CHAR, source=1, tag=0, com
[0] ERROR:       main (/home/u78663/Danysoft/Modulo5-MPI/pingpong/deadlock.c:65)
[0] ERROR:       __libc_start_main (/lib/x86_64-linux-gnu/libc-2.27.so)
[0] ERROR:       _start (/home/u78663/Danysoft/Modulo5-MPI/pingpong/deadlock)
[0] ERROR:    [1] no progress observed for over 0:20 minutes, process is currently in MPI call:
[0] ERROR:       MPI_Recv(*buf=0x7ffef01b3de0, count=8192, datatype=MPI_CHAR, source=0, tag=0, com
[0] ERROR:       main (/home/u78663/Danysoft/Modulo5-MPI/pingpong/deadlock.c:65)
[0] ERROR:       __libc_start_main (/lib/x86_64-linux-gnu/libc-2.27.so)
[0] ERROR:       _start (/home/u78663/Danysoft/Modulo5-MPI/pingpong/deadlock)

[0] INFO: GLOBAL:DEADLOCK:HARD: found 1 time (1 error + 0 warnings), 0 reports were suppressed
[0] INFO: Found 1 problem (1 error + 0 warnings), 0 reports were suppressed.


===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   RANK 0 PID 31670 RUNNING AT s001-n020
=   KILLED BY SIGNAL: 9 (Killed)
===================================================================================

===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   RANK 1 PID 31671 RUNNING AT s001-n020
=   KILLED BY SIGNAL: 9 (Killed)
===================================================================================
mpirun -check_mpi -genv VT_CHECK_TRACING on -genv VT_DEADLOCK_TIMEOUT 20s -
genv VT_DEADLOCK_WARNING 25s -genv VT_PCTRACE on -n 2 ./deadlock
```

Detección Deadlocks

- Ejemplo **deadlock.c**

Detección Deadlocks

- Ejemplo **deadlock_tag.c**

deadlock_tag.c

```
/* Call ping-pong functions */
if (rank == 0) {
   MPI_Ssend(buffer1, length, MPI_FLOAT, 1, tag1,
     MPI_COMM_WORLD);

   MPI_Recv(buffer2, length, MPI_FLOAT, 1, tag2,
       MPI_COMM_WORLD, &status);

   printf("pingpong performed\n");

   MPI_Ssend(buffer1, length, MPI_FLOAT, 1, tag1,
     MPI_COMM_WORLD);

   MPI_Ssend(buffer2, length, MPI_FLOAT, 1, tag2,
     MPI_COMM_WORLD);
}
else if (rank == 1) {

   MPI_Recv(buffer1, length, MPI_FLOAT, 0, tag1,
       MPI_COMM_WORLD, &status);

   MPI_Ssend(buffer2, length, MPI_FLOAT, 0, tag2,
     MPI_COMM_WORLD);

   MPI_Recv(buffer1, length, MPI_FLOAT, 0, tag2,
       MPI_COMM_WORLD, &status);

   MPI_Recv(buffer2, length, MPI_FLOAT, 0, tag1,
       MPI_COMM_WORLD, &status);

}
```