PRÁCTICAS CMCP

Alumno: Catalin Marian Cociu

E2. Integración numérica

La paralelización del programa se realiza haciendo que cada hilo compute una parte del subtotal de la suma para posteriormente sumar todos los subtotales, es decir, una reducción.

```
#pragma omp parallel for reduction(+ : s)
for (i = 0; i < n; i++)
{
    s += f(a + h * (i + 0.5));
}
```

Para obtener el número de hilos creamos una región paralela como la siguiente:

```
#pragma omp parallel
{
    int id = omp_get_thread_num();
    if (id == 0)
        printf("El numero de hilos es %d\n", omp_get_num_threads());
}
```

E3. Producto matricial

Las 3 versiones del bucle son:

```
#pragma omp parallel for private(j,k)
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    C[i*n+j] = 0.0;
    for (k=0; k<n; k++) {
      C[i*n+j] += A[i*n+k]*B[k*n+j];
    }
  }
}
```

Archivo: matmat-bucle-ex.c

```
for (i=0; i<n; i++) {
  #pragma omp parallel for private(k)
  for (j=0; j<n; j++) {
    C[i*n+j] = 0.0;
    for (k=0; k<n; k++) {
      C[i*n+j] += A[i*n+k]*B[k*n+j];
    }
  }
}
```

Archivo: matmat-bucle-med.c

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    C[i*n+j] = 0.0;
    double sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for (k=0; k<n; k++) {
      sum += A[i*n+k]*B[k*n+j];
    }
    C[i*n+j] = sum;
  }
}
```

Archivo: matmat-bucle-int.c

Para la toma de tiempo y número de hilos:

```
#pragma omp parallel
{
    int id = omp_get_thread_num();
    if(id == 0) printf("El numero de hilos es %d\n", omp_get_num_threads());
}

/* Multiplicación de matrices */
double t1 = omp_get_wtime();
matmat(n,A,B,C);
double t2 = omp_get_wtime();
printf("Duracion: %f\n", t2 - t1);
```

Resultados:

NOTA: Tiempos tomados en la máquina gpu

```
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=4 ./matmat-bucle-ex 1000
El numero de hilos es 4
Duracion: 1.369220
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=4 ./matmat-bucle-ex 1000
El numero de hilos es 4
Duracion: 1.430114
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=4 ./matmat-bucle-med 1000
El numero de hilos es 4
Duracion: 1.363710
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=4 ./matmat-bucle-med 1000
El numero de hilos es 4
Duracion: 1.318147
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=4 ./matmat-bucle-int 1000
El numero de hilos es 4
Duracion: 2.129715
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=4 ./matmat-bucle-int 1000
El numero de hilos es 4
Duracion: 2.109036
```

```
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=32 ./matmat-bucle-ex 2000
El numero de hilos es 32
Duracion: 4.324335
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=16 ./matmat-bucle-ex 2000
El numero de hilos es 16
Duracion: 5.248590
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=32 ./matmat-bucle-med 2000
El numero de hilos es 32
Duracion: 5.176206
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=16 ./matmat-bucle-med 2000
El numero de hilos es 16
Duracion: 5.063314
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=32 ./matmat-bucle-int 2000
El numero de hilos es 32
Duracion: 26.296175
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/openmp$ OMP_NUM_THREADS=16 ./matmat-bucle-int 2000
El numero de hilos es 16
Duracion: 15.954995
```

Se puede apreciar que la versión del producto con peores tiempos es la que paraleliza el bucle interno.

E4. Fractales–Paralelización con Regiones Paralelas

Para obtener el número de hilos:

```
int myId, numThreads;
#pragma omp parallel private(myId, numThreads)
{
  myId = omp_get_thread_num();
  numThreads = omp_get_num_threads();
  if(myId == 0) printf("Threads: %d\n", numThreads);
}
```

Para paralelizar el bucle:

```
/* For each vertical line... */
double t1 = omp_get_wtime();
#pragma omp parallel for schedule(runtime) private(i, x, y, k, u, v, w, a, b, value)
for(j = 0; j < height; j++) {
  y = uly - inc * (j+1);
  /* For each  horizontal line... */
  for(i = 0; i < width; i++) {
```

Resultados (la máquina Linux de polilabs vpn):

```
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=guided ./mandel -width 8000 -height 4000
 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 2.785710
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=guided ./mandel -width 8000 -height 4000
 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 2.819833
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=guided ./mandel -width 8000 -height 4000
 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 2.774140
```

```
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=static ./mandel -width 8000 -height 4000
 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 3.073317
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=static ./mandel -width 8000 -height 4000
 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 3.032186
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=static ./mandel -width 8000 -height 4000
 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 3.117630
```

```
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=dynamic ./mandel -width 8000 -height 400
0 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 2.768151
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=dynamic ./mandel -width 8000 -height 400
0 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 2.761070
cmarian@alumno.upv.es@ldsic-vdi02:~/W/CMCP/src-cmcp/openmp/fractal$ OMP_SCHEDULE=dynamic ./mandel -width 8000 -height 400
0 -ulx -1.37 -uly 0.014 -lly 0.0135
Threads: 4
El tiempo es: 2.768673
```

E5 Información de la GPU

Para imprimir el ancho de memoria del bus:

```
#endif
    printf("  Bus memory width:                                %d bits\n",
    deviceProp.memoryBusWidth);
}
```

Ejecución dquery:

```
Device 1: "Quadro RTX 5000"
  Major revision number:                         7
  Minor revision number:                         5
  Total amount of global memory:                 16905469952 bytes
  Number of multiprocessors:                     48
  Number of cores:                               3072
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Clock rate:                                    1.81 GHz
  Concurrent copy and execution:                 Yes
  Bus memory width:                              256 bits
```

E6. Suma de vectores

```cpp
__global__ void vecAddKernel(float *A, float *B, float *C, int n)
{

    // Calculate global thread index based on the block and thread indices ----

    // INSERT KERNEL CODE HERE
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    // Use global index to determine which elements to read, add, and write ---

    // INSERT KERNEL CODE HERE
    if (i < n)
        C[i] = A[i] + B[i];
}
```

```cpp
// Allocate device variables ----------------------------------------------

printf("Allocating device variables...");
fflush(stdout);
startTime(&timer);

// INSERT CODE HERE
long size = sizeof(float) * n;
float *d_A, *d_B, *d_C;
int nhilos = 256;
int nbloques = ceil(float(n) / nhilos);

cudaMalloc((void **)&d_A, size);
cudaMalloc((void **)&d_B, size);
cudaMalloc((void **)&d_C, size);

cudaDeviceSynchronize();
stopTime(&timer);
printf("%f s\n", elapsedTime(timer));
```

```
// Copy host variables to device ---------------------------------------
printf("Copying data from host to device...");
fflush(stdout);
startTime(&timer);

// INSERT CODE HERE

cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

cudaDeviceSynchronize();
stopTime(&timer);
printf("%f s\n", elapsedTime(timer));

// Launch kernel -------------------------------------------------------
printf("Launching kernel...");
fflush(stdout);
startTime(&timer);

// INSERT CODE HERE
vecAddKernel<<<nbloques, nhilos>>>(d_A, d_B, d_C, n);
```

```
// Copy device variables from host ---------------------------------------

printf("Copying data from device to host...");
fflush(stdout);
startTime(&timer);

// INSERT CODE HERE

cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

cudaDeviceSynchronize();
stopTime(&timer);
printf("%f s\n", elapsedTime(timer));

// Verify correctness --------------------------------------------------

printf("Verifying results...");
fflush(stdout);

verify(h_A, h_B, h_C, n);
```

```
// Free memory ---------------------------------------------------------

free(h_A);
free(h_B);
free(h_C);

// INSERT CODE HERE
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
```

```
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/cuda/vecadd$ ./vecadd 900

Setting up the problem...0.000058 s
    Vector size = 900
Allocating device variables...2.096677 s
Copying data from host to device...0.000021 s
Launching kernel...0.000016 s
Copying data from device to host...0.000011 s
Verifying results...TEST PASSED
```

E7. Suma de matrices

1. Completar el kernel:

```cpp
__global__ void matAddKernel(float* A, float* B, float* C, int n) {

    // Calculate global thread indices based on the block and thread indices ----
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;


    // Use global indices to determine which elements to read, add, and write ---
    if (i<n && j<n) C[i + n*j] = A[i+ n*j] + B[i + n*j];

}
```

2. Resultados:

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd$ ./matadd 20000

Setting up the problem...5.980121 s
    Matrix size = 20000
Allocating device variables...2.074115 s
Copying data from host to device...0.370446 s

Thread block: 16 x 16
Launching kernel...0.012615 s
Copying data from device to host...0.744494 s
Verifying results...TEST PASSED
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd$ ./matadd 30000

Setting up the problem...13.248618 s
    Matrix size = 30000
Allocating device variables...2.080099 s
Copying data from host to device...0.755273 s

Thread block: 16 x 16
Launching kernel...0.028372 s
Copying data from device to host...1.684783 s
Verifying results...TEST PASSED
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd$ ./matadd 20000

Setting up the problem...5.854697 s
    Matrix size = 20000
Allocating device variables...2.076535 s
Copying data from host to device...0.358942 s

Thread block: 32 x 32
Launching kernel...0.012645 s
Copying data from device to host...0.742324 s
Verifying results...TEST PASSED
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd$ ./matadd 30000

Setting up the problem...13.204714 s
    Matrix size = 30000
Allocating device variables...2.081353 s
Copying data from host to device...0.805229 s

Thread block: 32 x 32
Launching kernel...0.028452 s
Copying data from device to host...1.684962 s
Verifying results...TEST PASSED
```

Comprobamos que superando el número de hilos por bloque se produce un fallo:

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd$ ./matadd 30000

Setting up the problem...13.287187 s
    Matrix size = 30000
Allocating device variables...2.036900 s
Copying data from host to device...0.811397 s

Thread block: 256 x 256
Launching kernel...[matadd.cu:85] Unable to launch kernel
```

3. **Versión unidimensional:**

Kernel:

```
__global__ void matAddKernel(float* A, float* B, float* C, int n) {

    // Calculate global thread indices based on the block and thread indices
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j;


    // Use global indices to determine which elements to read, add, and write
    if (i<n) {
        for(j = 0; j < n; j++)
            C[i + n*j] = A[i+ n*j] + B[i + n*j];
    }

}
```

Invocación del kernel:

```
// Launch kernel ----------------------------------------------------------

int numThreads = 16;
printf("\nThread block: %d x 1 \n", numThreads);
printf("Launching kernel..."); fflush(stdout);
startTime(&timer);
dim3 nbloques(ceil(float(n)/numThreads));
matAddKernel<<<nbloques,numThreads>>>(d_A, d_B, d_C, n);
```

Resultados:

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd2$ ./matadd 20000

Setting up the problem...5.866638 s
    Matrix size = 20000
Allocating device variables...2.039417 s
Copying data from host to device...0.383303 s

Thread block: 16 x 1
Launching kernel...0.034482 s
Copying data from device to host...0.755446 s
Verifying results...TEST PASSED
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd2$ ./matadd 30000

Setting up the problem...13.244418 s
    Matrix size = 30000
Allocating device variables...2.064454 s
Copying data from host to device...0.826454 s

Thread block: 16 x 1
Launching kernel...0.079017 s
Copying data from device to host...1.692725 s
Verifying results...TEST PASSED
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd2$ ./matadd 20000

Setting up the problem...5.870152 s
    Matrix size = 20000
Allocating device variables...2.041578 s
Copying data from host to device...0.360833 s

Thread block: 32 x 1
Launching kernel...0.018357 s
Copying data from device to host...0.752701 s
Verifying results...TEST PASSED
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matadd2$ ./matadd 30000

Setting up the problem...13.364884 s
    Matrix size = 30000
Allocating device variables...2.041966 s
Copying data from host to device...0.795954 s

Thread block: 32 x 1
Launching kernel...0.050929 s
Copying data from device to host...1.679847 s
Verifying results...TEST PASSED
```

El rendimiento de la versión unidimensional es inferior a la versión bidimensional en todos los casos probados.

E8. Producto matriz por vector

1. Versión mejorada del kernel visto en clase:

```
__global__ void matvec_kernel(int n, double *A, double *x, double *y)
{
  int i, j, k;
  double res = 0.0;
  __shared__ double buff[BS];
  i = blockDim.x * blockIdx.x + threadIdx.x;
  for (k = 0; k < n; k += BS)
  {
    buff[threadIdx.x] = x[k + threadIdx.x];
    __syncthreads();
    for (j = k; j < min(k + BS, n); j++)
    {
      res += A[i + j * n] * buff[j - k];
    }
    __syncthreads();
  }
  if (i < n)
    y[i] = res;
}
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matvec$ ./matvec 20000

Setting up the problem...2.944880 s
    Matrix size = 20000
Allocating device variables...2.050536 s
Copying data from host to device...0.359798 s
Launching kernel...0.007941 s
Copying data from device to host...0.000094 s
Verifying results... PASS
cmarian@alumno.upv.es@gpu:~/cuda/matvec$ ./matvec 40000

Setting up the problem...11.881755 s
    Matrix size = 40000
Allocating device variables...2.063740 s
Copying data from host to device...1.452205 s
Launching kernel...0.031864 s
Copying data from device to host...0.000196 s
Verifying results... PASS
```

2. Versión de simple precisión:

```
__global__ void matvec_kernel(int n, float *A, float *x, float *y)
{
  int i, j, k;
  float res = 0.0;
  __shared__ float buff[BS];
  i = blockDim.x * blockIdx.x + threadIdx.x;
  for (k = 0; k < n; k += BS)
  {
    buff[threadIdx.x] = x[k + threadIdx.x];
    __syncthreads();
    for (j = k; j < min(k + BS, n); j++)
    {
      res += A[i + j * (size_t)n] * buff[j - k];
    }
    __syncthreads();
  }
  if (i < n)
    y[i] = res;
}
```

```
// Check the result of matvec (in CPU)
void verify(int n, float *A, float *x, float *y)
{
  int i, j;
  float *z, err = 0.0, nrm = 0.0;

  z = (float *)malloc(n * sizeof(float));
  for (i = 0; i < n; i++)
  {
    z[i] = 0.0;
    for (j = 0; j < n; j++)
    {
      z[i] += A[i + j * (size_t)n] * x[j];
    }
    err += fabsf(z[i] - y[i]);
    nrm += fabsf(z[i]);
  }
  if (err / nrm > 1e-5)
    printf(" relative error = %g\n", err / nrm);
  else
    printf(" PASS\n");
  free(z);
}
```

```
cmarian@alumno.upv.es@gpu:~/cuda/matvec$ ./matvec 20000

Setting up the problem...2.643806 s
    Matrix size = 20000
Allocating device variables...0.107897 s
Copying data from host to device...0.195701 s
Launching kernel...0.003983 s
Copying data from device to host...0.000068 s
Verifying results... PASS
cmarian@alumno.upv.es@gpu:~/cuda/matvec$ ./matvec 40000

Setting up the problem...10.158648 s
    Matrix size = 40000
Allocating device variables...2.065092 s
Copying data from host to device...0.666677 s
Launching kernel...0.015976 s
Copying data from device to host...0.000113 s
Verifying results... PASS
```

Se puede apreciar que esta versión obtiene unos tiempos 2 veces más rápidos.

E9. Multiplicación de matrices

Después de compilar las 2 versiones pedidas con el comando proporcionado en el boletín, se han ejecutado las siguientes pruebas:

```
cmarian@alumno.upv.es@gpu:~/cuda/matrixMulCUBLAS$ ./matrixMulCUBLAS sizemult=5
[Matrix Multiply CUBLAS] - Starting...
GPU Device 0: "Turing" with compute capability 7.5

GPU Device 0: "Quadro RTX 5000" with compute capability 7.5

MatrixA(640,480), MatrixB(480,320), MatrixC(640,320)
Computing result using CUBLAS...done.
Performance= 5173.60 GFlop/s, Time= 0.038 msec, Size= 196608000 Ops
Computing result using host CPU...done.
Comparing CUBLAS Matrix Multiply with CPU results: PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
cmarian@alumno.upv.es@gpu:~/cuda/matrixMulCUBLAS$ ./../matrixMul/matrixMul -wA=480 -hA=640 -hB=480 -wB=320
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Turing" with compute capability 7.5

MatrixA(480,640), MatrixB(320,480)
Computing result using CUDA Kernel...
done
Performance= 1143.24 GFlop/s, Time= 0.172 msec, Size= 196608000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performancemeasurements. Results may vary when GPU Boost is enabled.
```

NOTA: por alguna razón la versión matrixMul muestra las dimensiones de las matrices al revés, pero las dos pruebas usan matrices del mismo tamaño.

```
cmarian@alumno.upv.es@gpu:~/cuda/matrixMulCUBLAS$ ./matrixMulCUBLAS sizemult=8
[Matrix Multiply CUBLAS] - Starting...
GPU Device 0: "Turing" with compute capability 7.5

GPU Device 0: "Quadro RTX 5000" with compute capability 7.5

MatrixA(1024,768), MatrixB(768,512), MatrixC(1024,512)
Computing result using CUBLAS...done.
Performance= 8302.08 GFlop/s, Time= 0.097 msec, Size= 805306368 Ops
Computing result using host CPU...done.
Comparing CUBLAS Matrix Multiply with CPU results: PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
cmarian@alumno.upv.es@gpu:~/cuda/matrixMulCUBLAS$ ./../matrixMul/matrixMul -hA=1024 -wA=768 -hB=768 -wB=512
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Turing" with compute capability 7.5

MatrixA(768,1024), MatrixB(512,768)
Computing result using CUDA Kernel...
done
Performance= 1304.73 GFlop/s, Time= 0.617 msec, Size= 805306368 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performancemeasurements. Results may vary when GPU Boost is enabled.
```

Se puede apreciar que versión que hace uso de la librería cuBLAS tiene un rendimiento muy superior.

E.10 Ejecución de programas

Ejecución del programa hellow en los nodos de kahan:

```
cmarian@alumno.upv.es@kahan:~/mpi$ sbatch jobmpi.sh                    Submitted batch job 30881
cmarian@alumno.upv.es@kahan:~/mpi$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             30873      cmcp jobmpi.s ppermon1 PD       0:00      4 (Resources)
             30874      cmcp jobmpi.s koprosal PD       0:00      4 (Priority)
             30875      cmcp jobmpi.s gaguecha PD       0:00      4 (Priority)
             30876      cmcp hello.sh mgarmon4 PD       0:00      4 (Priority)
             30877      cmcp jobmpi.s fdomlor@ PD       0:00      4 (Priority)
             30878      cmcp  ej10.sh groicle@ PD       0:00      4 (Priority)
             30879      cmcp jobmpi.s gaguecha PD       0:00      4 (Priority)
             30880      cmcp hello_mp jogugi@a PD       0:00      4 (Priority)
             30881      cmcp jobmpi.s cmarian@ PD       0:00      4 (Priority)
             30882      cmcp jobhello jlpulgav PD       0:00      2 (Resources)
                                 cmarian@alumno.upv.es@kahan:~/mpi$ ls
cpi.c   hellow.c   poisson.c  slurm-30865.out
hellow  jobmpi.sh  ring_c.c
cmarian@alumno.upv.es@kahan:~/mpi$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
cmarian@alumno.upv.es@kahan:~/mpi$ cat slurm-30881.out Hello, world, I am 0 of 4
Hello, world, I am 3 of 4
Hello, world, I am 1 of 4
Hello, world, I am 2 of 4
cmarian@alumno.upv.es@kahan:~/mpi$
```

Para ejecutar el programa se ha creado un script ‘jobmpi’ como el siguiente:

```
cmarian@alumno.upv.es@gpu:~/W/CMCP/src-cmcp/mpi$ cat jobmpi.sh
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --ntasks=4
#SBATCH --time=5:00
#SBATCH --partition=cmcp

mpiexec ./hellow
```

Se ha utilizado el comando ‘sbatch’ para lanzar el script como un job en el sistema de colas de kahan y el comando ‘squeue’ para ver los Jobs encolados. El job genera un archivo slurm-id.out que contiene la salida de la ejecución de nuestro programa.

Ejecución del programa hellow en el nodo frontend:

```
cmarian@alumno.upv.es@kahan:~/mpi$ mpiexec -n 4 hellow
Hello, world, I am 0 of 4
Hello, world, I am 1 of 4
Hello, world, I am 2 of 4
Hello, world, I am 3 of 4
cmarian@alumno.upv.es@kahan:~/mpi$
```

E11. Comunicación en anillo

Modificaciones para imprimir el tiempo que tarda la ejecución

1. Declaramos dos variables de tipo double: start y end
2. Registramos tiempo antes del envío del primer mensaje

```
if (0 == rank) {
    message = 10;

    printf("Process 0 sending %d to %d, tag %d (%d processes in ring)\n",
           message, next, tag, size);
    start = MPI_Wtime();
    MPI_Send(&message, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
    printf("Process 0 sent to %d\n", next);
}
```

3. Registramos tiempo después de la última recepción e imprimimos el tiempo total

```
/* The last process does one extra send to process 0, which needs
   to be received before the program can exit */

if (0 == rank) {
    MPI_Recv(&message, 1, MPI_INT, prev, tag, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    end = MPI_Wtime();
    printf("Elapsed time: %f \n", end - start);
}

/* All done */
```

Salida de la ejecución:

```
cmarian@alumno.upv.es@kahan:~/mpi$ cat slurm-30936.out
Process 0 sending 10 to 1, tag 201 (4 processes in ring)
Process 0 sent to 1
Process 0 decremented value: 9
Process 0 decremented value: 8
Process 0 decremented value: 7
Process 0 decremented value: 6
Process 0 decremented value: 5
Process 0 decremented value: 4
Process 0 decremented value: 3
Process 0 decremented value: 2
Process 0 decremented value: 1
Process 0 decremented value: 0
Process 0 exiting
Elapsed time: 0.057526
Process 2 exiting
Process 3 exiting
Process 1 exiting
```

E12. Cálculo de Pi

Modificaciones en el programa:

```
mypi = h * sum;
/* Communication: receive in process 0 the value of mypi from the rest of processes
   and accumulate it on pi in process 0 */
if (myid == 0) {
    printf("Partial value of pi from process 0: %f \n", mypi);
    MPI_Status status;
    pi = mypi;
    for (p=1; p<numprocs; p++) {
        MPI_Recv(&aux, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 22, MPI_COMM_WORLD, &status);
        printf("Partial value of pi from process %d: %f \n", status.MPI_SOURCE, aux);
        pi += aux;
    }
} else {
    MPI_Send(&mypi, 1, MPI_DOUBLE, 0, 22, MPI_COMM_WORLD);
}
/* End communication */
```

Sustituimos el proceso p por MPI_ANY_SOURCE para poder recibir desde cualquier proceso y obtenemos el proceso de origen utilizando una variable de tipo MPI_Status que contiene información adicional sobre el mensaje recibido. También agregamos dos instrucciones printf, una para el proceso 0 y otra para el resto, para saber el valor parcial de p calculado por cada proceso.

Resultados:

```
cmarian@alumno.upv.es@kahan:~/W/CMCP/src-cmcp/mpi$ mpiexec -n 4 ./cpi
Enter the number of intervals: (0 quits)
18
Partial value of pi from process 0: 0.883581
Partial value of pi from process 1: 0.854207
Partial value of pi from process 2: 0.715175
Partial value of pi from process 3: 0.688887
pi is approximately 3.1418498551793710, Error is 0.0002572015895779
Enter the number of intervals: (0 quits)
50
Partial value of pi from process 0: 0.820512
Partial value of pi from process 1: 0.810310
Partial value of pi from process 2: 0.760303
Partial value of pi from process 3: 0.750501
pi is approximately 3.1416259869230037, Error is 0.0000333333332105
Enter the number of intervals: (0 quits)
```

E13. Ecuación de Poisson - Particionado Unidimensional Vertical

```c
void jacobi_step_parallel(int N, int M, double *x, double *b, double *t, int rank, int size, double *sum) {
    int i, j, n_local = N / size;
    int ld = M + 2;
    double local_sum;


    if (rank % 2 == 0) {
        if (rank > 0) {
            MPI_Send(&x[ld], ld, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&x[0], ld, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        if (rank < size - 1) {
            MPI_Send(&x[n_local * ld], ld, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&x[(n_local + 1) * ld], ld, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
    } else {
        if (rank > 0) {
            MPI_Recv(&x[0], ld, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(&x[ld], ld, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD);
        }
        if (rank < size - 1) {
            MPI_Recv(&x[(n_local + 1) * ld], ld, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(&x[n_local * ld], ld, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD);
        }
    }

    // Calcular la nueva iteración
    for (i = 1; i <= n_local; i++) {
        for (j = 1; j <= M; j++) {
            t[i * ld + j] = (b[i * ld + j] +
                             x[(i + 1) * ld + j] +
                             x[(i - 1) * ld + j] +
                             x[i * ld + (j + 1)] +
                             x[i * ld + (j - 1)]) / 4.0;
            local_sum += (x[i * ld + j] - t[i * ld + j]) * (x[i * ld + j] - t[i * ld + j]);
        }
    }
    *sum = local_sum;
}
```

Se ha optado por utilizar el protocolo de pares-impares para resolver el proceso de la secuencialización.

E14. Cálculo de Pi con colectivas

Reemplazamos el bucle que hacía los envíos con MPI_Bcast y el bucle que hacía la recepción con MPI_Reduce

```c
start = MPI_Wtime();
/* Communication: value of n from process 0 to all other processes */
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
/* End communication */
if (n == 0)
    break;
else {
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x*x));
    }
    mypi = h * sum;
    /* Communication: receive in process 0 the value of mypi from the
ocesses
        and accumulate it on pi in process 0 */
    if (myid == 0) {
        pi = mypi;
    }
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    end = MPI_Wtime();
    /* End communication */
```

E15. Ecuación de Poisson - Programa Completo

```c
void jacobi_poisson(int N, int M, double *x, double *b, int rank, int size) {
    int i, j, k, ld = M + 2, conv, maxit = 50000;
    double *t, s, tol = 1e-6;

    t = (double*)calloc((N/size + 2) * (M + 2), sizeof(double));

    k = 0;
    conv = 0;

    while (!conv && k < maxit) {
        jacobi_step_parallel(N, M, x, b, t, rank, size, &s);

        double global_error;
        MPI_Allreduce(&s, &global_error, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

        conv = (sqrt(global_error) < tol);
        if (rank == 0) {
            printf("Error en iteración %d: %g\n", k, sqrt(global_error));
        }
        k++;

        MPI_Barrier(MPI_COMM_WORLD);

        for (i = 1; i <= N/size; i++) {
            for (j = 1; j <= M; j++) {
                x[i * ld + j] = t[i * ld + j];
            }
        }
    }

    free(t);
}
```

Reemplazos el bucle de calculo de error con una reducción a todos los procesos.

Dentro del método main, hacemos Gather para reunir todos los fragmentos de X y luego imprimimos la solución.

```c
  /* Resolución del sistema por el método de Jacobi */
  jacobi_poisson(N,M,x,b,rank,size);

  if (N <= 60) {
    double *X = NULL;
    if (rank == 0) {
        X = (double *)calloc((N+2) * (M + 2), sizeof(double));
    }
    MPI_Gather(&x[ld], N/size * ld, MPI_DOUBLE, X + ld, N/size * ld, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    /* Imprimir solucion */
    if (rank == 0) {
        for (i = 1; i <= N; i++) {
            for (j = 1; j <= M; j++) {
                printf("%g ", X[i * ld + j]);
            }
            printf("\n");
        }
        free(X);
    }

| }
```

E16. Ecuación de Poisson - Particionado Unidimensional Horizontal

```c
void parallel_jacobi_step(int N, int mlocal, double *x, double *b, double *t, double *local_sum, int rank, int size, MPI_Datatype coltype) {
    int i, j, ld = mlocal + 2;

    double local_s = 0.0;
    int right = rank + 1;
    int left = rank - 1;

    if (rank % 2 == 0) {
        if (rank > 0) {
            MPI_Send(&x[1], 1, coltype, left, 1, MPI_COMM_WORLD);
            MPI_Recv(&x[0], 1, coltype, left, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        if (rank < size - 1) {
            MPI_Send(&x[ld-1], 1, coltype, right, 0, MPI_COMM_WORLD);
            MPI_Recv(&x[ld], 1, coltype, right, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
    } else {
        if (rank > 0) {
            MPI_Recv(&x[0], 1, coltype, left, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(&x[1], 1, coltype, left, 1, MPI_COMM_WORLD);
        }
        if (rank < size - 1) {
            MPI_Recv(&x[ld], 1, coltype, right, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(&x[ld-1], 1, coltype, right, 0, MPI_COMM_WORLD);
        }
    }

    for (i = 1; i <= N; i++) {
        for (j = 1; j <= mlocal; j++) {
            t[i*ld+j] = (b[i*ld+j] + x[(i+1)*ld+j] + x[(i-1)*ld+j] + x[i*ld+(j+1)] + x[i*ld+(j-1)])/4.0;
            local_s += (x[i * ld + j] - t[i * ld + j]) * (x[i * ld + j] - t[i * ld + j]);
        }
    }

    *local_sum = local_s;
}
```

```c
void parallel_jacobi_poisson(int N, int mlocal, double *x, double *b, int rank, int size, MPI_Datatype coltype) {
    int i, j, k, ld = mlocal + 2, conv = 0, maxit = 10000;
    double *t, tol = 1e-6, local_sum, global_sum = 0.0;

    t = (double *)calloc((N+2) * (mlocal + 2), sizeof(double));

    k = 0;

    while (!conv && k < maxit) {
        parallel_jacobi_step(N, mlocal, x, b, t, &local_sum, rank, size, coltype);

        MPI_Allreduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

        conv = (sqrt(global_sum) < tol);

        if (rank == 0) {
            printf("Error en iteración %d: %g\n", k, sqrt(global_sum));
        }

        MPI_Barrier(MPI_COMM_WORLD);

        k = k + 1;

        // Actualizar la malla x con los nuevos valores
        for (i = 1; i <= N; i++) {
            for (j = 1; j <= mlocal; j++) {
                x[i * ld + j] = t[i * ld + j];
            }
        }
    }

    free(t); // Liberar memoria
}
```

```
MPI_Comm_rank(comm, &rank);
MPI_Comm_size(comm, &size);
int mlocal = N / size;
ld = mlocal + 2;

// Crear tipo de dato derivado para enviar columnas
MPI_Datatype coltype;
MPI_Type_vector(N+2, 1, mlocal+2, MPI_DOUBLE, &coltype);
MPI_Type_commit(&coltype);

// Redimensionamos el tipo de datopara asegurar que la memoria correctamente con las columnas
MPI_Datatype column_type_resized;
MPI_Type_create_resized(coltype, 0, sizeof(double), &column_type_resized);
MPI_Type_commit(&column_type_resized);

// Reservar memoria para las matrices x y b
x = (double *)calloc((N+2) * (mlocal + 2), sizeof(double));
b = (double *)calloc((N+2) * (mlocal + 2), sizeof(double));

// Inicializar la matriz b
for (i = 1; i <= N; i++) {
    for (j = 1; j <= mlocal; j++) {
        b[i * ld + j] = h * h * f;
    }
}

parallel_jacobi_poisson(N, mlocal, x, b, rank, size, coltype);

if (N <= 60) {
    double *X = NULL;
    if (rank == 0) {
        X = (double *)calloc((N+2) * (M + 2), sizeof(double));
    }

    // Usar MPI_Gather para recoger las columnas calculadas por cada proceso
    MPI_Gather(x + 1, mlocal, column_type_resized, X + 1, mlocal, column_type_resized, 0, MPI_COMM_WORLD);

    // Liberar el tipo de datos derivado
    MPI_Type_free(&column_type_resized);
```

```
        // Liberar el tipo de datos derivado
        MPI_Type_free(&column_type_resized);

        if (rank == 0) {
            for (i = 1; i <= N; i++) {
                for (j = 1; j <= M; j++) {
                    printf("%g ", X[i * ld + j]);
                }
                printf("\n");
            }
            free(X);
        }
    }

    // Liberar memoria
    free(x);
    free(b);

    MPI_Finalize();

    return 0;
}
```