

Introducción a la programación de aceleradores gráficos

Curso de Introducción a la Programación de Altas Prestaciones

Rocío Carratalá, Adrián Castelló, Sandra Catalán y Sergio Iserte



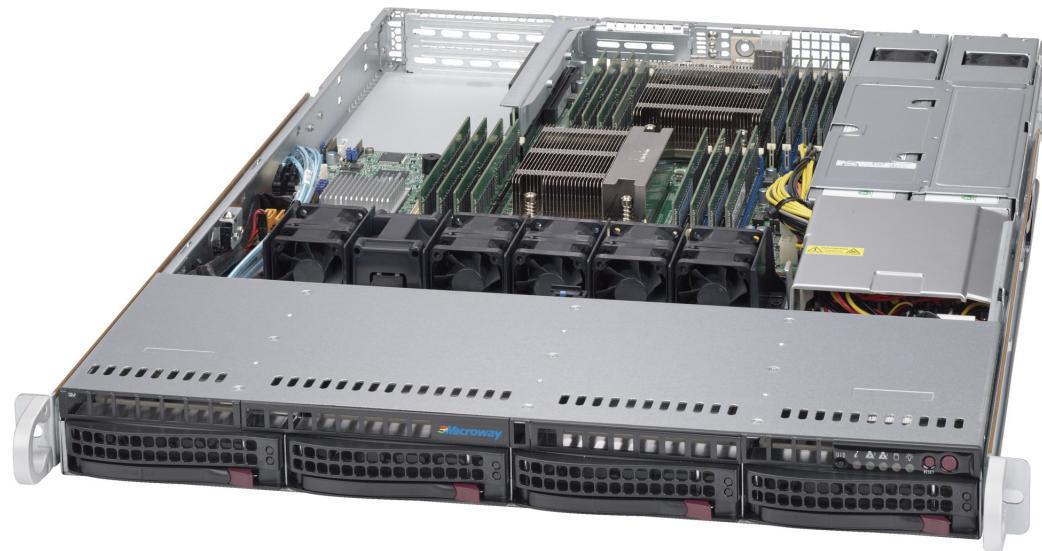


NVIDIA®

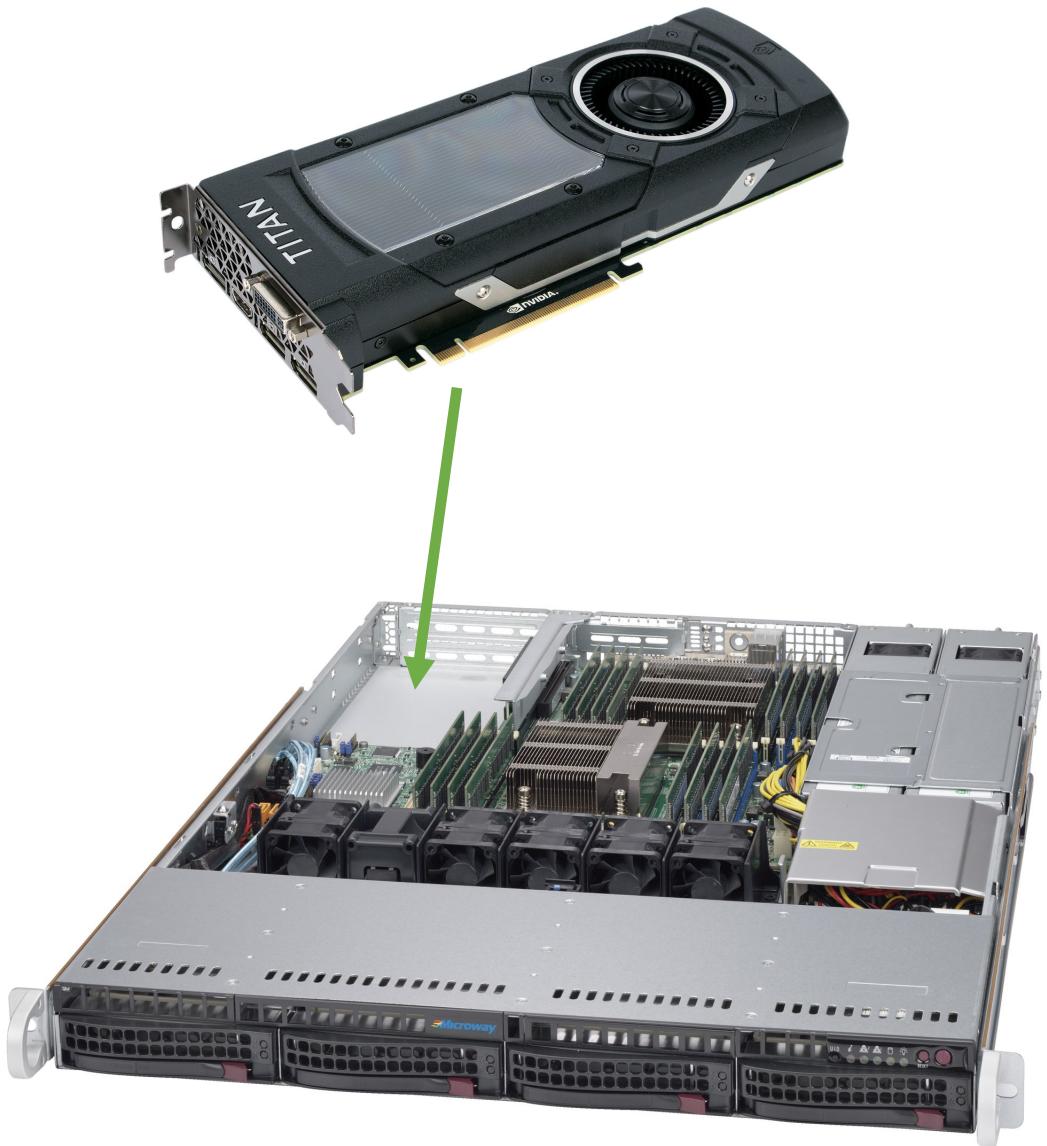




VS

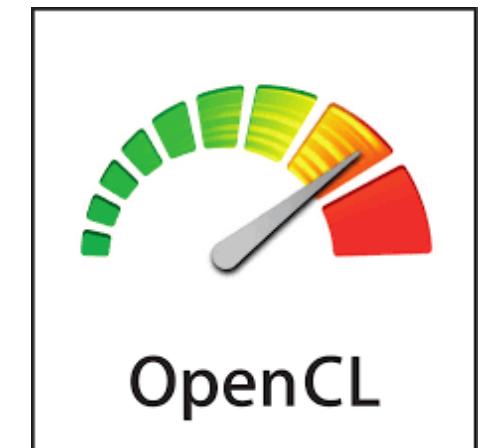
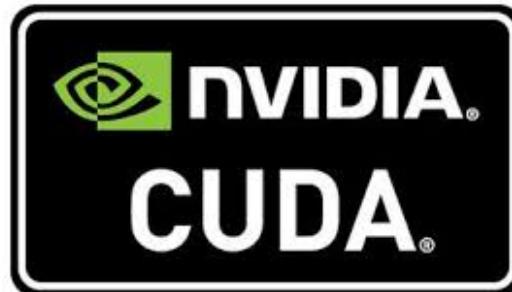


Dispositivo	NVIDIA Titan X	Intel Skylake Xeon E5
Numero de Núcleos	3072	Hasta 28
Memoria (GB)	12	Depende del sistema
GFlops	11.000	4.480

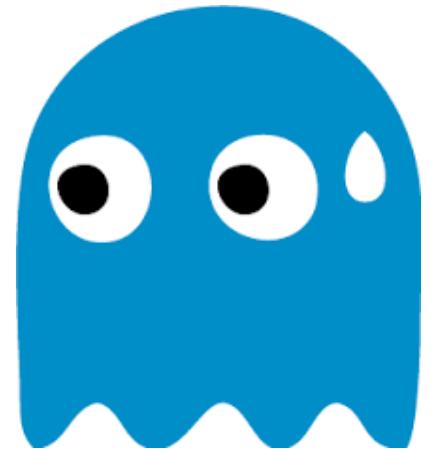


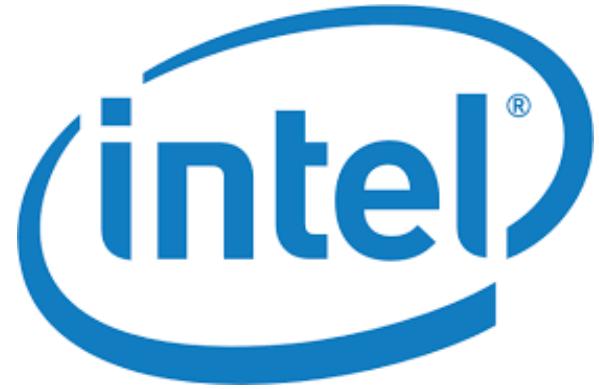
Programando tarjetas gráficas

- Antes de 2007
 - El programador se apañaba con códigos tipo ensamblador
- CUDA
 - Última versión CUDA 9.5
- OpenCL
 - También para gráficas no nVIDIA
 - Última versión 2.0
- OpenACC
 - Basado en directivas
 - Última versión 2.5



OpenACC
Directives for Accelerators



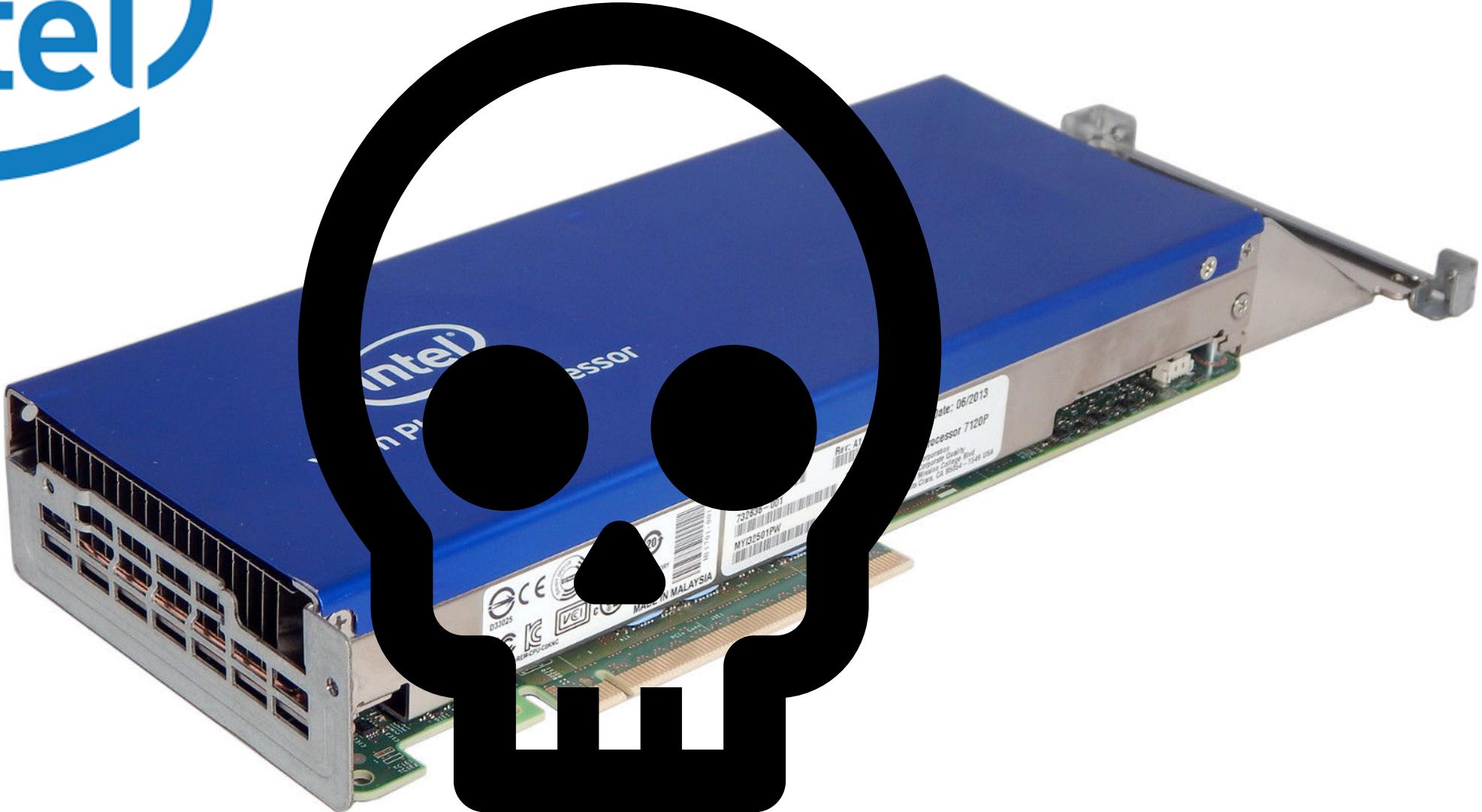
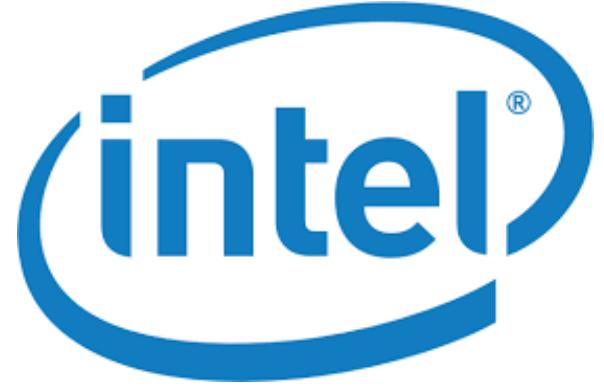




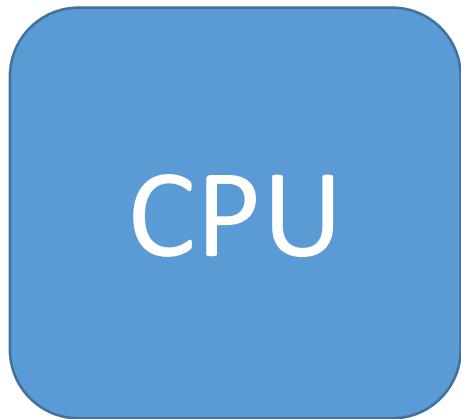
VS



Dispositivo	NVIDIA Titan X	Intel Xeon Phi KNL
Numero de Núcleos	3072	72x4
Memoria (GB)	12	384
GFlops	11.000	6.000
¿Cómo se programa?	CUDA (C) / OpenACC	OpenMP

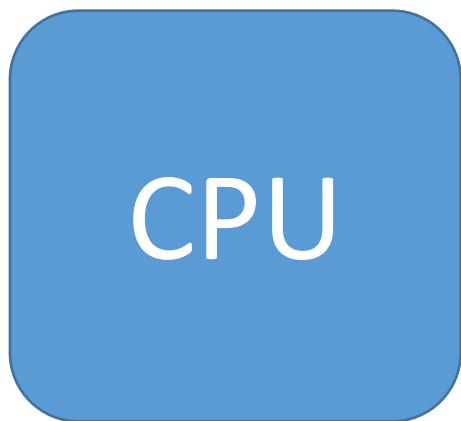


¿Cómo se ejecutan códigos en la GPU?

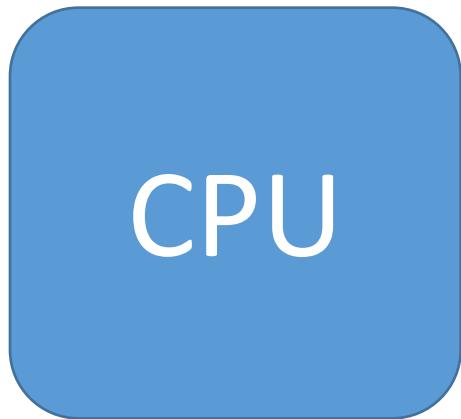


¿Cómo se ejecutan códigos en la GPU?

Código para CPU



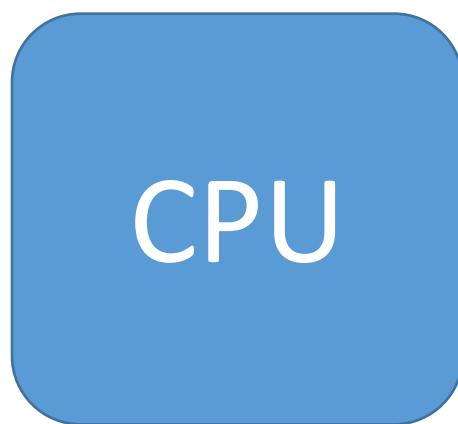
¿Cómo se ejecutan códigos en la GPU?



Código para GPU



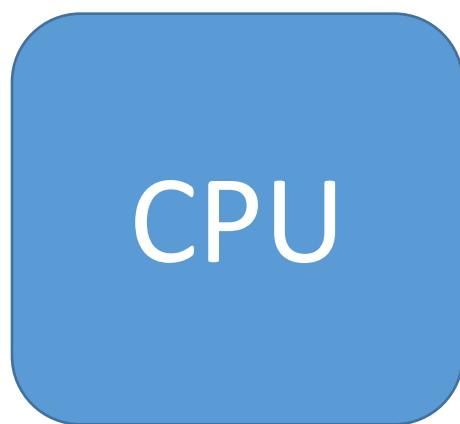
¿Cómo se ejecutan códigos en la GPU?



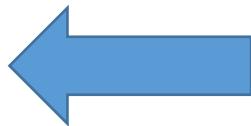
Se copian datos a la GPU



¿Cómo se ejecutan códigos en la GPU?

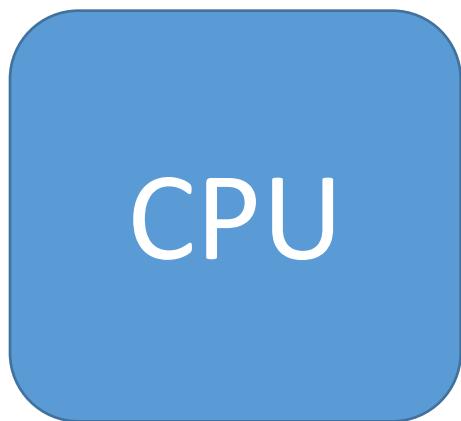


Se copian datos a la CPU



¿Cómo se ejecutan códigos en la GPU?

Código para CPU





Código en C

```
void saxpy(int n, float a, float *x, float *y)
{
    for(int i = 0; i < n; i++)
        y[i] = a*x[i] + y[i];
}
int main(void)
{
    int N = 1<<20;
    float *x, *y;

    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));

    for (int i = 0; i < N; i++) {
        x[i] = 1.0f; y[i] = 2.0f;
    }

    saxpy(N, 2.0f, x, y);

    free(x);
    free(y);
}
```

Código en CUDA

```
__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;

    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));
    cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

    for (int i = 0; i < N; i++)
        x[i] = 1.0f; y[i] = 2.0f;

    cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

    saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);

    cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

    float maxError = 0.0f;
    for (int i = 0; i < N; i++)
        maxError = max(maxError, abs(y[i]-4.0f));
    printf("Max error: %f\n", maxError);

    cudaFree(d_x); cudaFree(d_y);
    free(x); free(y);
}
```

```

__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;

    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));
    cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

    for (int i = 0; i < N; i++) {
        x[i] = 1.0f; y[i] = 2.0f;
    }

    cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

    saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);

    cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

    float maxError = 0.0f;
    for (int i = 0; i < N; i++)
        maxError = max(maxError, abs(y[i]-4.0f));

    printf("Max error: %f\n", maxError);

    cudaFree(d_x);
    cudaFree(d_y);
    free(x);
    free(y);
}

```

¡Nuevas palabras reservada!



__global__
blockIdx.x
blockDim.x
threadIdx.x
cudaMalloc
cudaMemcpy
cudaMemcpyHostToDevice
saxpy<<<(N+255)/256, 256>>>
cudaMemcpyDeviceToHost
cudaFree

```
__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}
```

```
int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;
```

```
x = (float*)malloc(N*sizeof(float));
y = (float*)malloc(N*sizeof(float));
cudaMalloc(&d_x, N*sizeof(float));
cudaMalloc(&d_y, N*sizeof(float));
```

```
for (int i = 0; i < N; i++) {
    x[i] = 1.0f; y[i] = 2.0f;
}
```

```
cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);
```

```
saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
```

```
cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
    maxError = max(maxError, abs(y[i]-4.0f));
```

```
printf("Max error: %f\n", maxError);
```

```
cudaFree(d_x);
cudaFree(d_y);
free(x);
free(y);
```

```
}
```

Gestión de memoria

Copia de datos

Código a ejecutar
(GPU)

mini

Introducción a: **OpenACC** Directives for Accelerators

Adrián Castelló

```

__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;

    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));
    cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

    for (int i = 0; i < N; i++) {
        x[i] = 1.0f; y[i] = 2.0f;
    }

    cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

    saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);

    cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

    float maxError = 0.0f;
    for (int i = 0; i < N; i++)
        maxError = max(maxError, abs(y[i]-4.0f));

    printf("Max error: %f\n", maxError);

    cudaFree(d_x);
    cudaFree(d_y);
    free(x);
    free(y);
}

```

Campo Científico	¿Será un paso natural el uso de CUDA?
Informática	Si
Física	No
Química	No
Aeronautica	No
Otros...	No

```

__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;

    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));
    cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

    for (int i = 0; i < N; i++) {
        x[i] = 1.0f; y[i] = 2.0f;
    }

    cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

    saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);

    cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

    float maxError = 0.0f;
    for (int i = 0; i < N; i++)
        maxError = max(maxError, abs(y[i]-4.0f));

    printf("Max error: %f\n", maxError);

    cudaFree(d_x);
    cudaFree(d_y);
    free(x);
    free(y);
}

```

Campo Científico	¿Será un paso natural el uso de CUDA?
Informática	Si
Física	No
Química	No
Aeronautica	No
Otros...	No



Suelen utilizar OpenMP en sus códigos...

OpenACC

Directives for Accelerators

- Es un estándar (como OpenMP)
- Basado en directivas (como OpenMP)
- Se inician con la palabra clave `#pragma` (como OpenMP)
- Abstraen la gestión del entorno (como OpenMP)
- Utilizan GPUs (casi como OpenMP)

```
__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}
```

```
int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;
```

```
x = (float*)malloc(N*sizeof(float));
y = (float*)malloc(N*sizeof(float));
cudaMalloc(&d_x, N*sizeof(float));
cudaMalloc(&d_y, N*sizeof(float));
```

```
for (int i = 0; i < N; i++) {
    x[i] = 1.0f; y[i] = 2.0f;
}
```

```
cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);
```

```
saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
```

```
cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
    maxError = max(maxError, abs(y[i]-4.0f));
```

```
printf("Max error: %f\n", maxError);
```

```
cudaFree(d_x);
cudaFree(d_y);
free(x);
free(y);
```

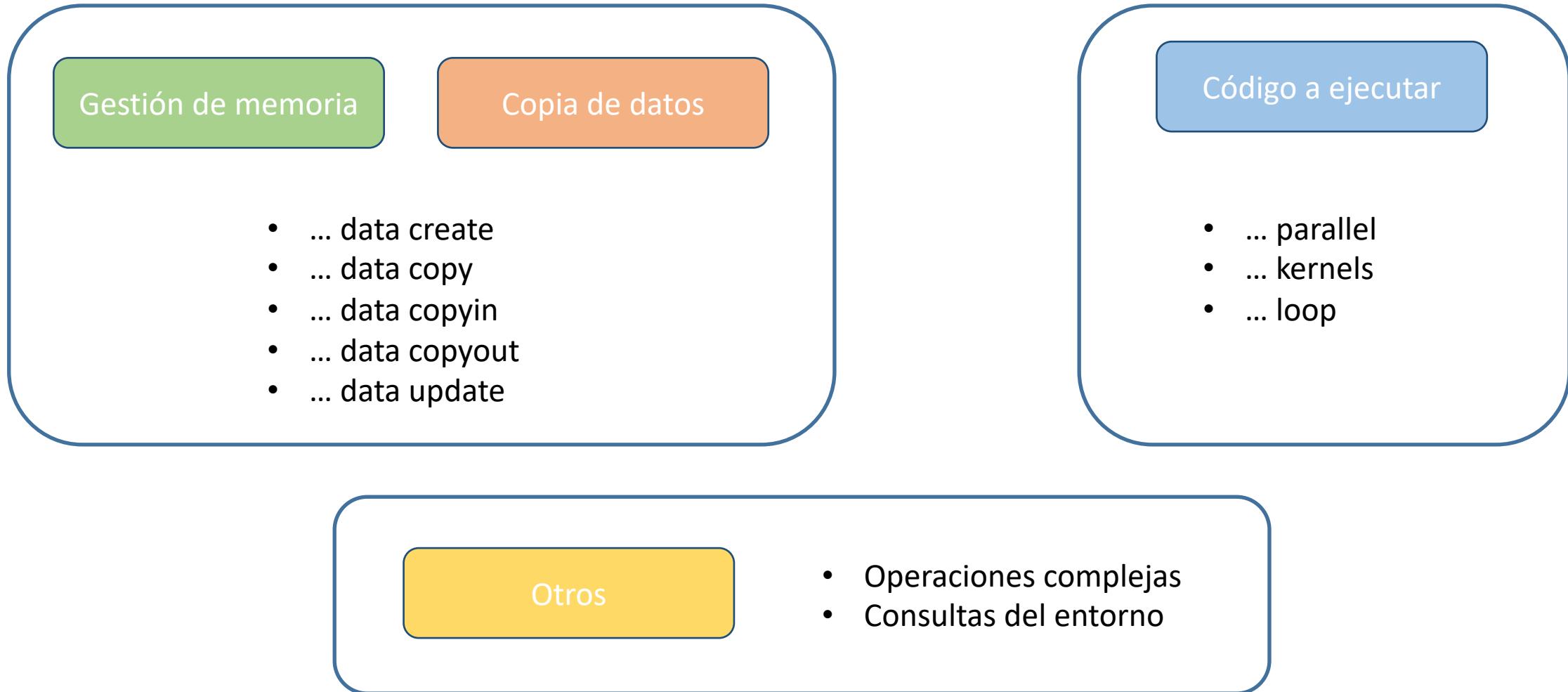
```
}
```

Gestión de memoria

Copia de datos

Código a ejecutar

#pragma acc...



```
__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}
```

```
int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;
```

```
x = (float*)malloc(N*sizeof(float));
y = (float*)malloc(N*sizeof(float));
cudaMalloc(&d_x, N*sizeof(float));
cudaMalloc(&d_y, N*sizeof(float));
```

```
for (int i = 0; i < N; i++) {
    x[i] = 1.0f; y[i] = 2.0f;
}
```

```
cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);
```

```
saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
```

```
cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
    maxError = max(maxError, abs(y[i]-4.0f));
```

```
printf("Max error: %f\n", maxError);
```

```
cudaFree(d_x);
cudaFree(d_y);
free(x);
free(y);
```

```
}
```

Gestión de memoria

Copia de datos

Código a ejecutar

```
int main(void)
{
    int N = 1<<20;
    float *x, *y;

    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));

    for (int i = 0; i < N; i++) {
        x[i] = 1.0f; y[i] = 2.0f;
    }
```

```
#pragma acc data copy(x[0:N],y[0:N])
```

```
{
    #pragma acc kernels
    for(int i = 0; i<N; i++)
        y[i] = a*x[i] + y[i];
}
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
    maxError = max(maxError, abs(y[i]-4.0f));
```

```
printf("Max error: %f\n", maxError);
```

```
free(x);
free(y);
```

```
}
```

```
#pragma acc data copy(x[0:N],y[0:N])
```

- Indica transferencia de datos
 - copyin es para mover datos de la CPU a la GPU
 - copyout mueve datos de la GPU a la CPU
 - copy = copyin + copyout
- Parámetros
 - El nombre de la variable y el rango de datos a mover



```
#pragma acc data create(x[0:N])
```

- Reserva en la GPU memoria para x de tamaño N
 - Útil para datos que van a permanecer solamente en la GPU
- Parámetros
 - El nombre de la variable y el rango de datos a reservar

```
#pragma acc data update (d) (x[:])
```

- Actualiza valores en las memorias
- Parámetros
 - d puede ser device o host, e indica qué memoria queremos actualizar
 - El nombre de la variable y el rango de datos a actualizar

#pragma acc kernels

- Indica que el siguiente código se debe “traducir” a kernel de CUDA
 - ¿Quién crea el código?
 - ¿Cómo se dividen las iteraciones?

```
#pragma acc kernels
for (int i = 0; i < N; i++){
    y[i] = a * x[i] + y[i];
}
```



```
__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a*x[i] + y[i];
}
```

#pragma acc loop

- Indica que el siguiente bucle se debe “traducir” a kernel de CUDA
- Kernels puede contener varios bucles

```
#pragma acc kernels
    #pragma acc loop
    for (int i = 0; i < N; i++){
        #pragma acc loop
            for (int j = 0; j < N; j++){
                y[i*N+j] = a * x[i] + y[i*N+j];
            }
    }
```

Y si yo se más que el compilador...

- Número de hilos

```
#pragma acc kernels gang (num_gangs) worker (num_workers)
```

Siendo **gang** el equivalente a número de bloques CUDA y **worker** el número de threads en cada bloque

```
saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
```

Y si yo se más que el compilador...

- Número de hilos

```
#pragma acc kernels gang (num_gangs) worker (num_workers)
```

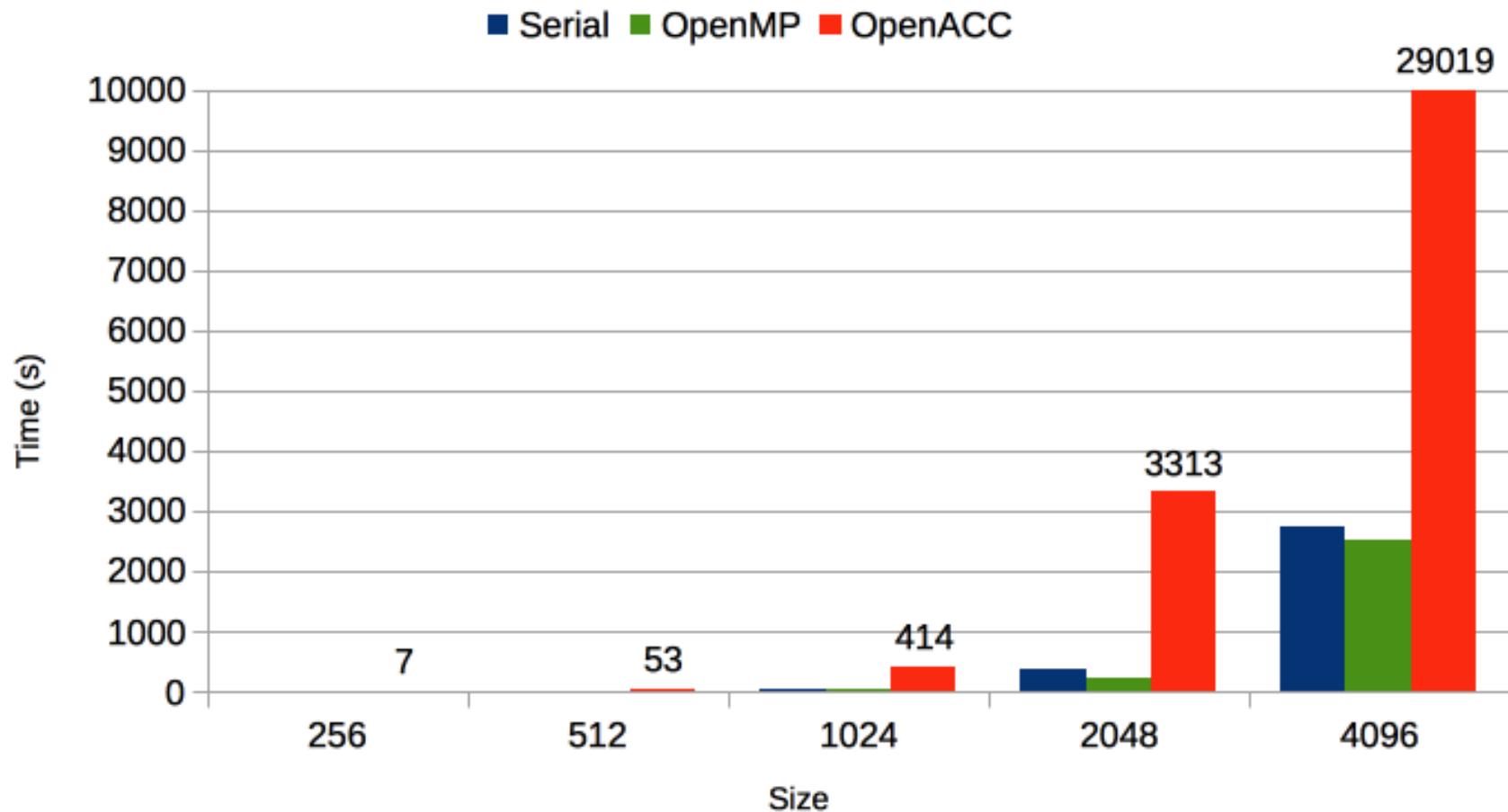
Siendo **gang** el equivalente a número de bloques CUDA y **worker** el número de threads en cada bloque

```
saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
```

¡No es tan simple como parece!

```
#pragma acc data copy(A[0:N*N], B[0:N*N], C[N*N])
{
    #pragma acc kernels
    #pragma acc loop
    for (int i = 0; i < N; i++){
        #pragma acc loop
        for (int j = 0; j < N; j++){
            C[i*N+j] *= 1;
            for (int k = 0; j < N; k++){
                C[i*N+j] += 1 * x[i*N+k] + B[k*N+j];
            }
        }
    }
}
```

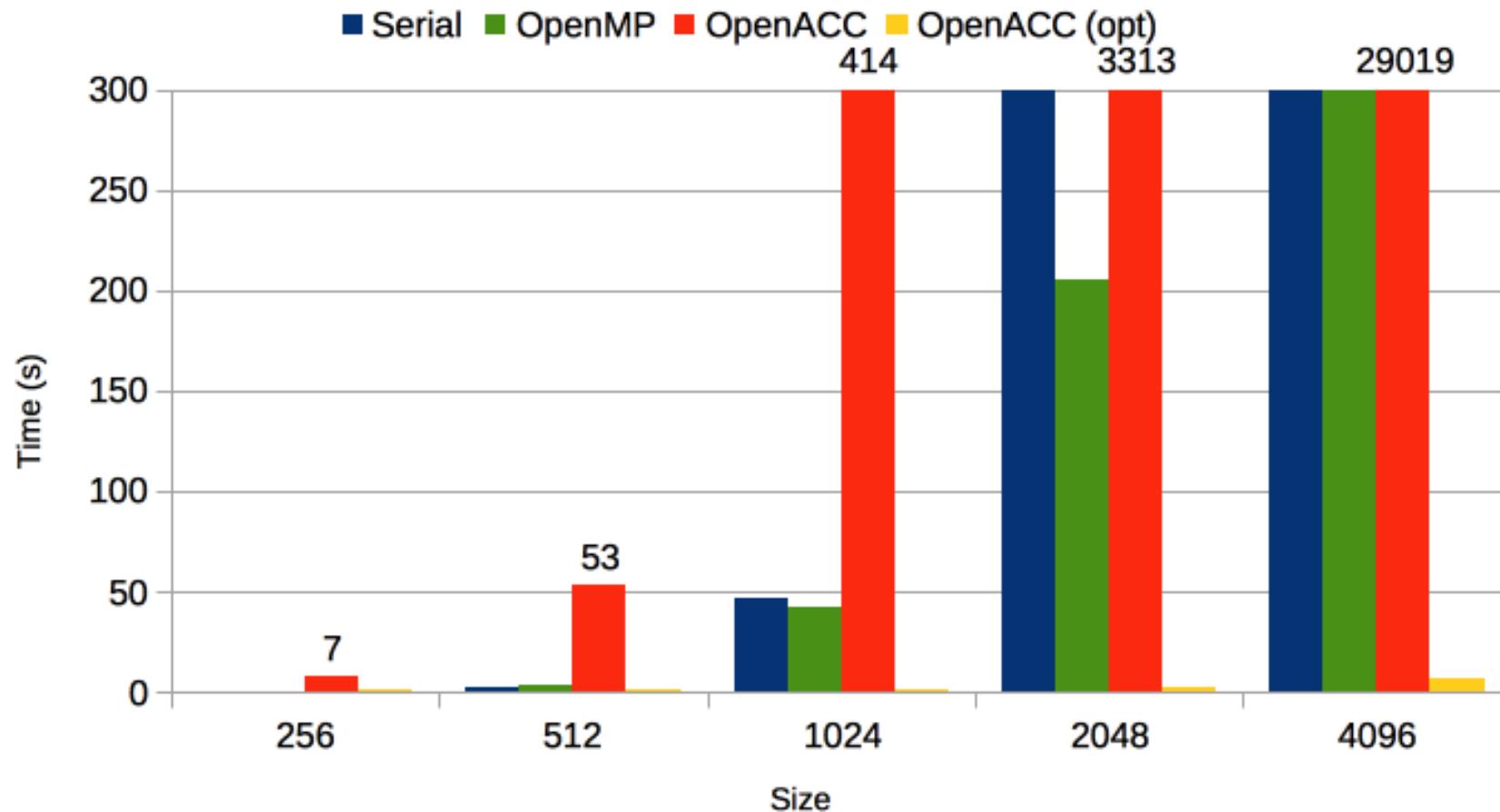
¡No es tan simple como parece!



¡No es tan simple como parece!

```
#pragma acc data copyin(A[0:N*N], B[0:N*N]), copyout(C[N*N])
{
    #pragma acc kernels
    #pragma acc loop
    for (int i = 0; i < N; i++){
        #pragma acc loop
        for (int j = 0; j < N; j++){
            C[i*N+j] *= 1;
            for (int k = 0; j < N; k++){
                C[i*N+j] += 1 * x[i*N+k] + B[k*N+j];
            }
        }
    }
}
```

¡No es tan simple como parece!



OpenACC

Directives for Accelerators

- Orientado a productividad
- Punto de inicio para programar en GPUs
- Cuidado con los pragmas

Ejercicios (con ayuda)



- El “hello world” de OpenACC (Escalar un vector) (test1)
- Multiplicación de matrices (test2)
- Bucle while (test3)
- Laplace
- CUDA

Ejercicios (con ayuda)



- Antes de empezar:
 - cd testX
 - ./carga_entorno.sh
- Para ejecutar un ejemplo
 - cd testX
 - make
 - ./ejecutable

Pasos a seguir para realizar el ejercicio:

1. Intentar entender el código
2. Identifica los datos a enviar a la GPU
3. Identifica el código a ejecutar en la GPU
4. ¡Probarlo!