II. Ejemplos de programación: Seis formas de implementar SAXPY en GPU

¿Qué es SAXPY? Single-precision Alpha X plus Y. Es parte de la librería BLAS.

```
void saxpy_serial(float ... )
{
 for (int i = 0; i < n; ++i)
   y[i] = a*x[i] + y[i];
}
```

- Utilizando este código básico, ilustraremos seis formas diferentes de programar la GPU:
  - CUDA C.
  - CUBLAS Library.
  - CUDA Fortran.
  - Thrust C++ Template Library.
  - C# con GPU.NET.
  - OpenACC.

1. CUDA C [http://developer.nvidia.com/cuda-toolkit]

Código C estándar

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invocar al kernel SAXPY secuencial (1M elementos)
saxpy_serial(4096*256, 2.0, x, y);
```

Código CUDA equivalente de ejecución paralela en GPU:

```
__global__ void saxpy_parallel(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];
}
// Invocar al kernel SAXPY paralelo (4096 bloques de 256 hilos)
saxpy_parallel<<<4096, 256>>>(4096*256, 2.0, x, y);
```


2. CUBLAS Library [http://developer.nvidia.com/cublas]

Código BLAS secuencial

```
int N = 1 << 20;
// Utiliza la librería BLAS de tu elección

// Invoca a la rutina SAXPY secuencial (1M elementos)
blas_saxpy(4096*256, 2.0, x, 1, y, 1);
```

Código cuBLAS paralelo

```
int N = 1 << 20;
cublasInit();
cublasSetVector (N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector (N, sizeof(y[0]), y, 1, d_y, 1);
// Invoca a la rutina SAXPY paralela (1M elementos)
cublasSaxpy (N, 2.0, d_x, 1, d_y, 1);
cublasGetVector (N, sizeof(y[0], d_y, 1, y, 1);
cublasShutdown();
```

3. CUDA Fortran [http://developer.nvidia.com/cuda-fortran]

Fortran estándar

```
module my module contains
  subroutine saxpy (n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i) + y(i);
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  $ Invoca SAXPY para 1M elementos
  call saxpy(2**20, 2.0, x, y)

end program main
```

Fortran paralelo

```
module mymodule contains
  attributes(global) subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    attributes(value) :: a, n
    i = threadIdx%x + (blockIdx%x-1) * blockDim%x
    if (i<=n) y(i) = a*x(i) + y(i)
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x_d(2**20), y_d(2**20)
  x_d = 1.0, y_d = 2.0

  $ Invoca SAXPY para 1M elementos
  call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)
  y = y_d
end program main
```


4.1. CUDA C++ [http://developer.nvidia.com/cuda-toolkit]

- Con CUDA C++ podemos desarrollar código paralelo genérico, que permite encarar aplicaciones sofisticadas y flexibles con un rico middleware:
  - Jerarquía de clases.
  - Métodos __device__.
  - Plantillas (templates).
  - Sobrecarga de operadores.
  - Functors (objetos función).
  - New/delete en device.
  - ...

```
template <typename T>
struct Functor {
  __device__ Functor(_a) : a(_a) {}
  __device__ T operator(T x) { return a*x; }
  T a;
}

template <typename T, typename Oper>
__global__ void kernel(T *output, int n) {
  Oper op(3.7);
  output = new T[n]; // dynamic allocation
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n)
    output[i] = op(i);  // apply functor
}
```

it

4.2. Desarrollo ágil de código C++ paralelo [http://developer.nvidia.com/thrust]

- Reensambla C++ STL.
- Interfaz de alto nivel.
  - Mejora la productividad.
  - Habilita la portabilidad del rendimiento entre GPUs y CPUs.
- Flexible:
  - Dispone de back-ends para CUDA, OpenMP y TBB.
  - Extensible y parametrizable.
  - Se integra con el software existente.
- Código abierto.

Thrust

```
// generate 32M random numbers on host
thrust::host_vector<int> h_vec(32 << 20);
thrust::generate(h_vec.begin(),
                 h_vec.end(),
                 rand);

// transfer data to device (GPU)
thrust::device_vector<int> d_vec = h_vec;

// sort data on device
thrust::sort(d_vec.begin(), d_vec.end());

// transfer data back to host
thrust::copy(d_vec.begin(),
             d_vec.end(),
             h_vec.begin());
```


4.3. Thrust C++ Template Library

Código C++ secuencial con STL y Boost

```
int N = 1<<20;
std::vector<float> x(N), y(N);
...



// Invocar SAXPY para 1M elementos
std::transform(x.begin(), x.end(),
               y.begin(), x.end(),
               2.0f * _1 + _2);
```

Código C++ paralelo

```
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);
...

...
thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;

// Invocar SAXPY para 1M elementos
thrust::transform(x.begin(), x.end(),
                  y.begin(), y.begin(),
                  2.0f * _1 + _2);
```

http://www.boost.org/libs/lambda

http://developer.nvidia.com/thrust

5. C# con GPU.NET [http://tidepowerd.com]

C# estándar

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

int N = 1<<20;

// Invoca SAXPY para 1M elementos
saxpy(N, 2.0, x, y)
```

C# paralelo

```
[kernel]
private static
void saxpy (int n, float a,
            float[] a, float[] y)
{
  int i = BlockIndex.x * BlockDimension.x +
          ThreadIndex.x;
  if (i < n)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;

Launcher.SetGridSize(4096);
Launcher.SetBlockSize)4096);

// Invoca SAXPY para 1M elementos
saxpy(2**20, 2.0, x, y)
```


6. OpenACC (directivas de compilación) [http://developer.nvidia.com/openacc]

Código C paralelo

```
void saxpy (int n, float a,
            float[] a, float[] y)
{
#pragma acc kernels
  for (int i=0; i<n; i++)
    y[i] = a*x[i] + y[i];
}

...
// Invoca SAXPY para 1M elementos
saxpy(1<<20, 2.0, x, y)
...
```

Código Fortran Paralelo

```
subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
$!acc kernels
  do i=1. n
    y(i) = a*x(i) + y(i)
  enddo
$!acc end kernels
end subroutine saxpy

...
$ Invoca SAXPY para 1M elementos
call saxpy(2**20, 2.0, x_d, y_d)
...
```