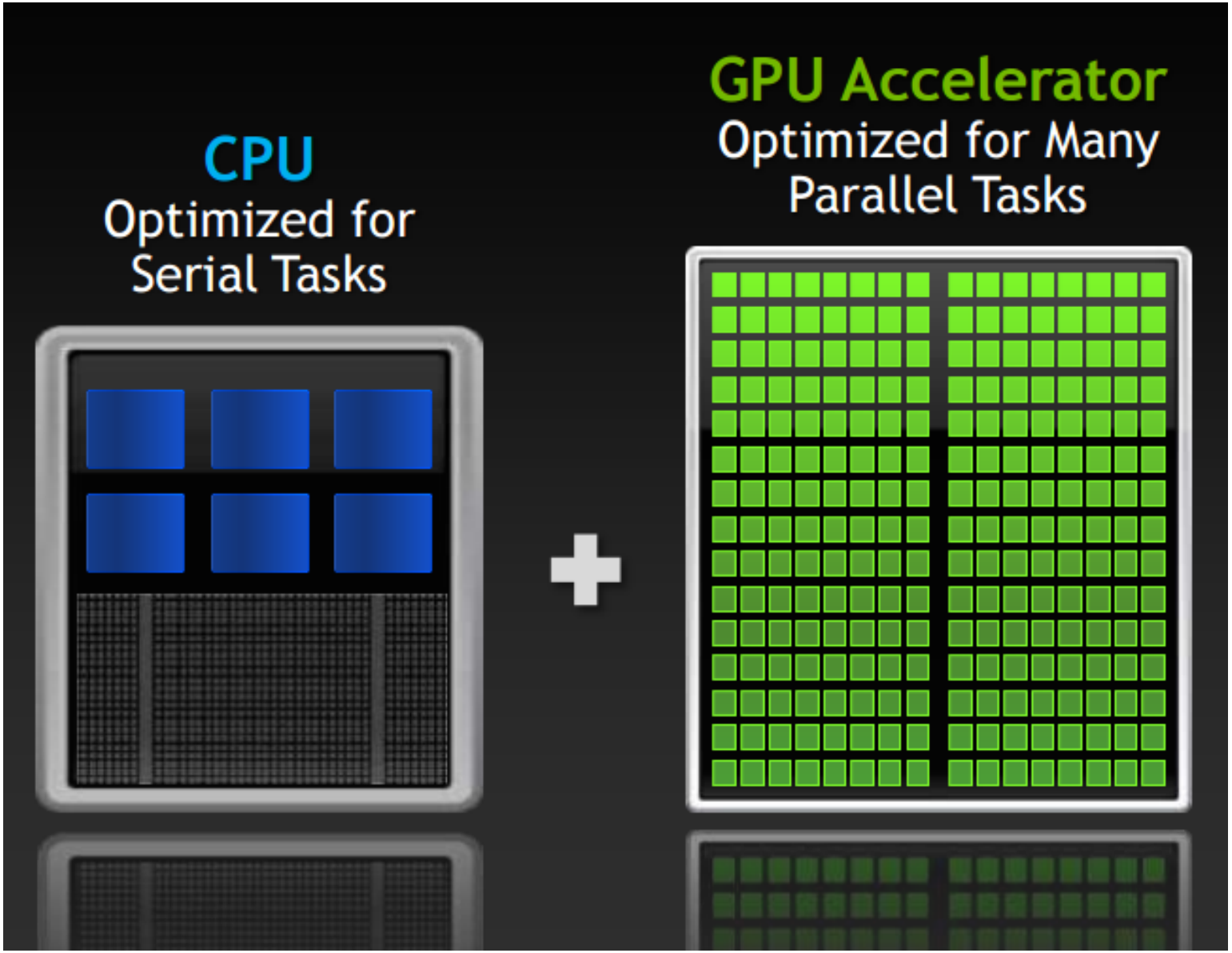
Introducción a OPENACC

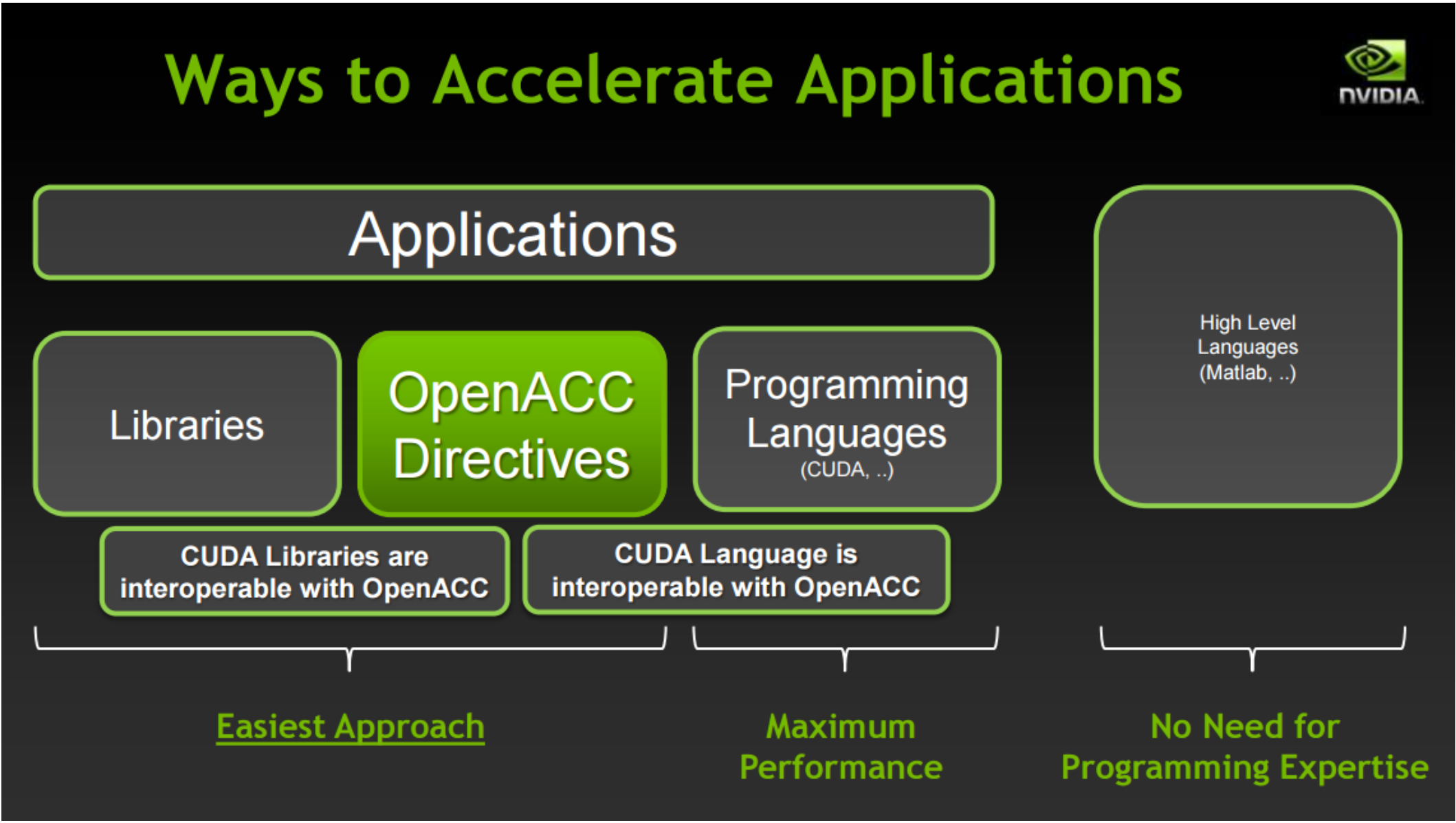
Clase 8

Computación heterogénea



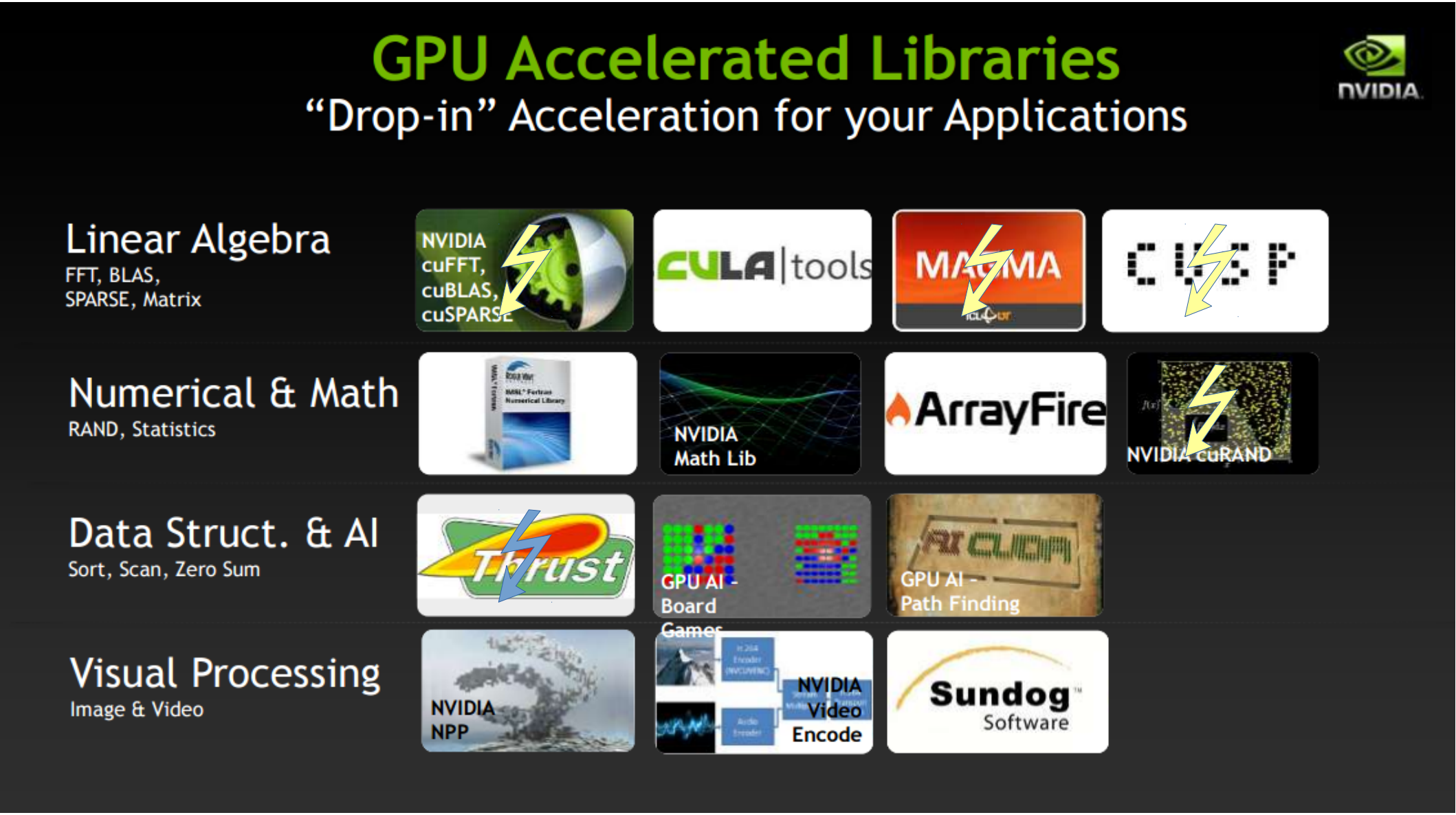
http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Otra forma de acelerar aplicaciones ...



http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Librerias (mas o menos portables)



¿ Que es OPENACC ?

http://www.openacc.org/

¿ Para quien es OPENACC ?

- **Tienen un código serial C/C++/fortran y quieren intentar acelerarlo “rápidamente” con GPUs:**
  - Modificando mínimamente el código serial original ...
  - Que aún con estas mínimas “adiciones” se pueda correr normalmente en CPU, como siempre ...
  - Sin aprender o sin usar CUDA...
  - Sin pretender máxima-performance pero si mínimo esfuerzo...

(perspectiva personal)

¿ Para quien es OPENACC ?

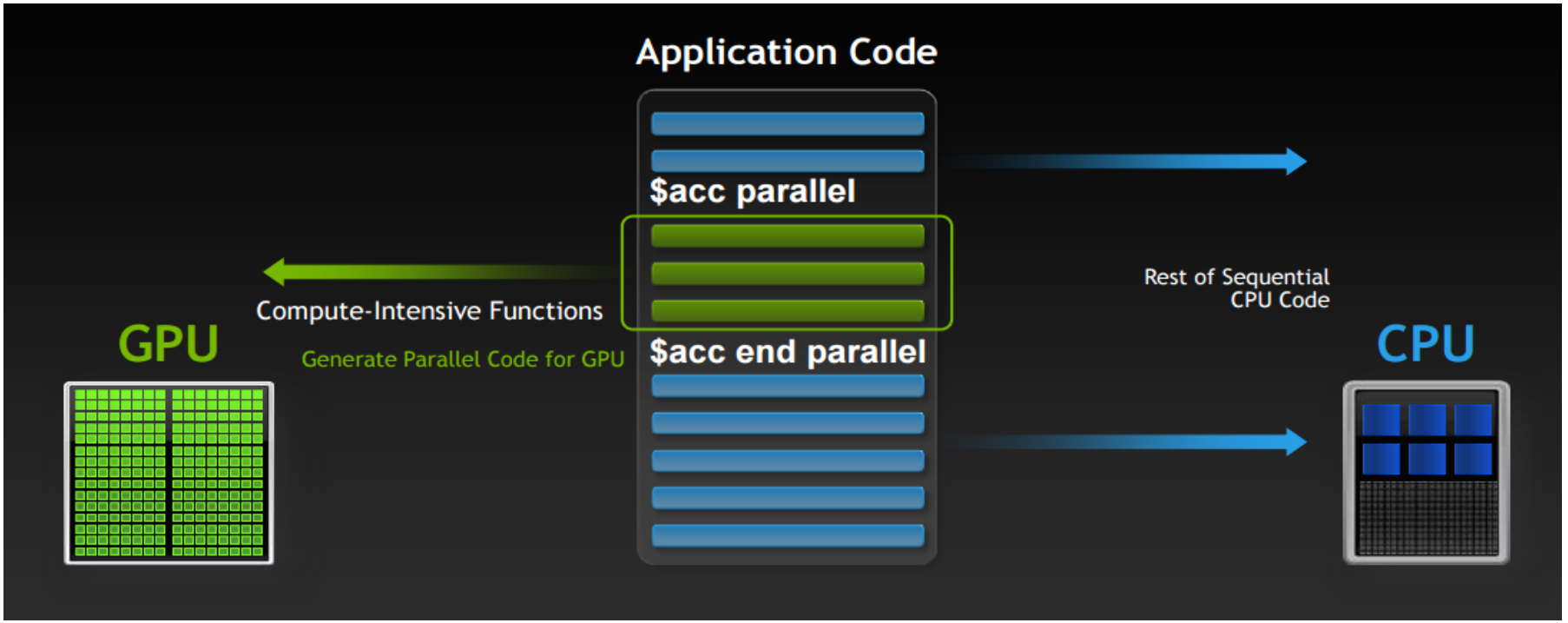
- **Ejemplos**:
  - Estudiantes que heredan códigos seriales pulsudos de sus directores/jefes...
  - Investigadores con poco tiempo para programar y/o aprender a programar otro lenguaje, pero con muchas ideas para testear numéricamente ya ya ya ...
  - Desarrolladores que tiene que mejorar un software serial hecho (mejorar un producto) ...
  - Programadores que quieren paralelizar rutinas de viejas pero muy queridas librerías ...
  - Docentes que quieren enganchar más gente a GPGPU@CAB, y al curso de ICNPG ...

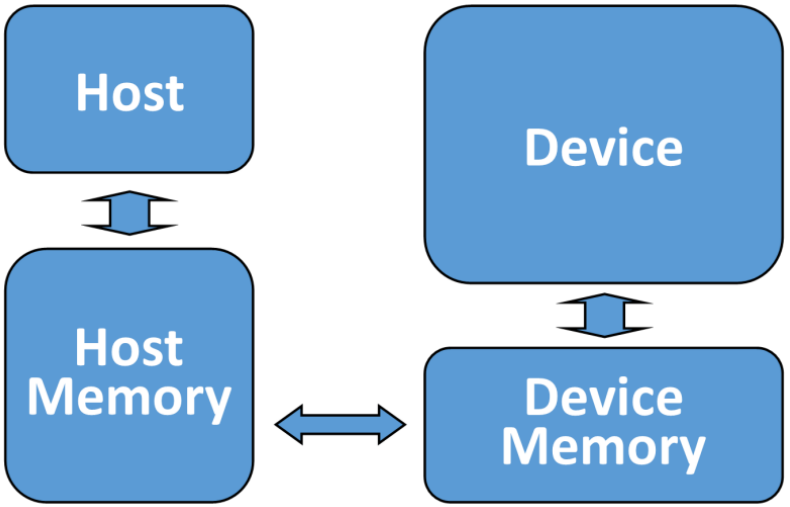
(perspectiva personal)

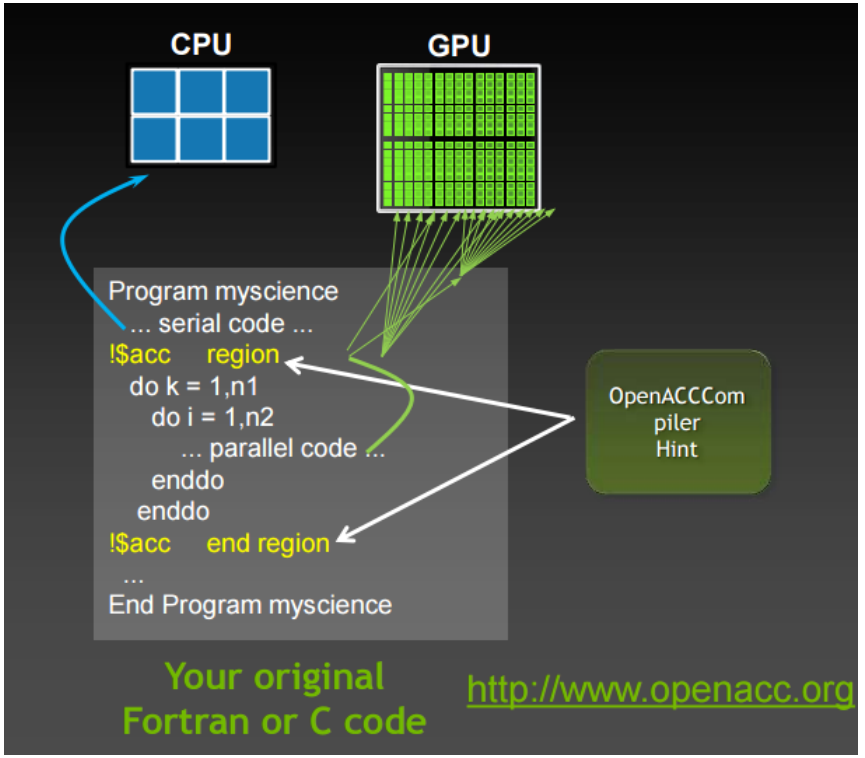
¿ Que es OPENACC ?

- Las *directivas* de OpenACC son “hints al compilador” que permiten acelerar códigos **seriales** en **C/C++ y Fortran** rápidamente en NVIDIA GPUs, x86 o ARM CPUs, Intel Xeon Phi, AMD Radeon, sin requerir un nuevo lenguaje como CUDA o OPENCL.
- *El compilador se encarga de paralelizar el **código serial**, con ayuda de la información que le damos. Mas allá de eso, delegamos completamente la implementación (muy similar a OPENMP).*

http://www.openacc.org/


Application Code
$acc parallel
$acc end parallel
Compute-Intensive Functions
Generate Parallel Code for GPU
GPU
Rest of Sequential
CPU Code
CPU


Host
Host
Memory
Device
Device
Memory


CPU
GPU
Program myscience
... serial code ...
!$acc region
do k = 1,n1
do i = 1,n2
... parallel code ...
enddo
enddo
!$acc end region
...
End Program myscience
OpenACCCom
piler
Hint
Your original
Fortran or C code
http://www.openacc.org

*13x agregando **una sola línea** al programa!*

Serial Code

*Single CPU Core Performance: **1x***

```
.
.
.
.
for(int j=1;j<ny-1;j++) {
 for(int k=i1;k<nz-1;k++) {
  for(int i=1;i<nx-1;i++) {
   Anext[Index3D (nx,ny,i,j,k)] =
    (A0[Index3D (nx,ny,i,j,k+1)] +
       A0[Index3D (nx,ny,i,j,k-1)] +
       A0[Index3D (nx,ny,i,j+1,k)] +
       A0[Index3D (nx,ny,i,j-1,k)] +
       A0[Index3D (nx,ny,i+1,j,k)] +
       A0[Index3D (nx,ny,i-1,j,k)])*c1
       -A0[Index3D (nx,ny,i,j,k)]*c0;
   }
  }
}
```

Parallel Code for GPU

Add One OpenACC Directive

*Tesla K40 Perf: **13.6x***

```
.
.
.
#pragma acc parallel loop collapse(3)
for(int j=1;j<ny-1;j++) {
 for(int k=i1;k<nz-1;k++) {
  for(int i=1;i<nx-1;i++) {
   Anext[Index3D (nx,ny,i,j,k)] =
    (A0[Index3D (nx,ny,i,j,k+1)] +
       A0[Index3D (nx,ny,i,j,k-1)] +
       A0[Index3D (nx,ny,i,j+1,k)] +
       A0[Index3D (nx,ny,i,j-1,k)] +
       A0[Index3D (nx,ny,i+1,j,k)] +
       A0[Index3D (nx,ny,i-1,j,k)])*c1
       -A0[Index3D (nx,ny,i,j,k)]*c0;
   }
  }
}
```

Dual socket E5-2698 v3 @2.3GHz (Haswell), 16 cores per socket, 256 GB memory, 1x Tesla K40
Benchmark: Parboil Stencil from University of Illinois with 1000 iterations

Source code for Parboil: http://impact.crhc.illinois.edu/Parboil/parboil.aspx

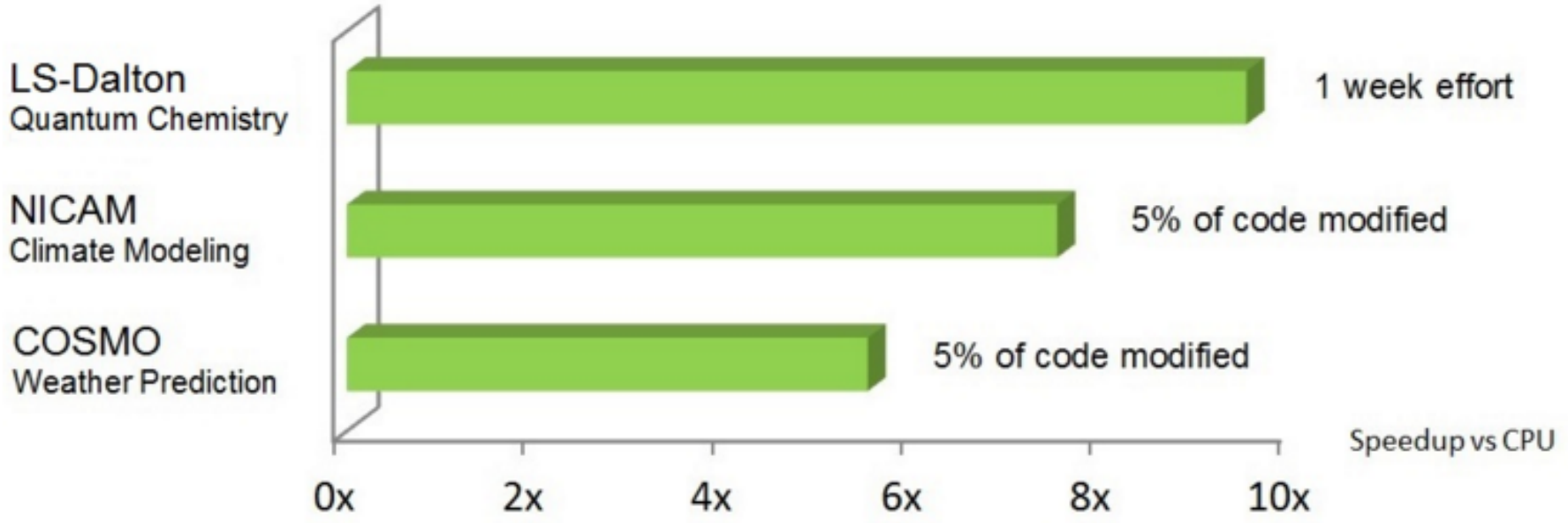
OpenACC Toolkit

Propaganda...

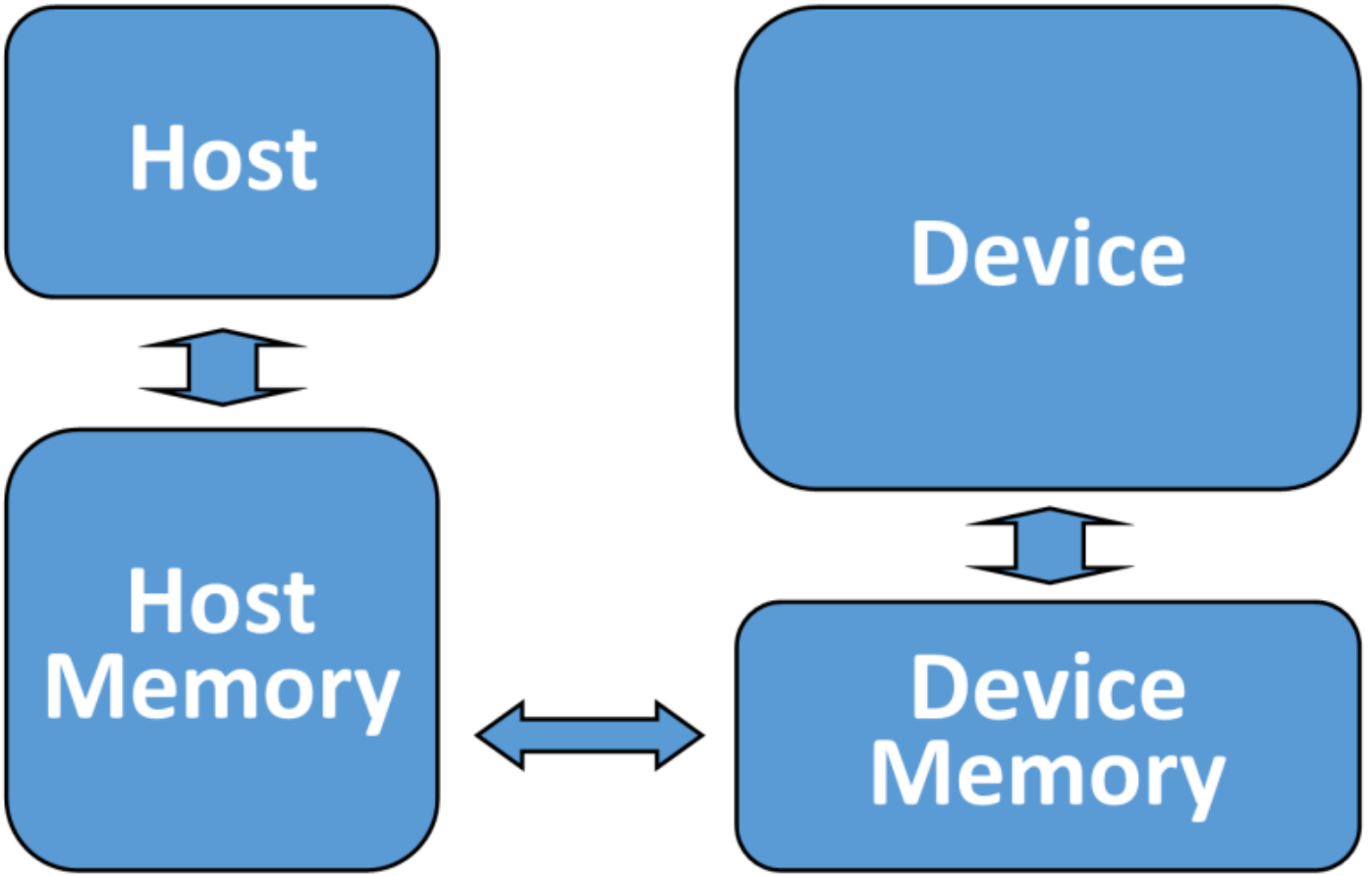
The OpenACC Toolkit from NVIDIA offers scientists and researchers a simple way to accelerated scientific computing without significant programming effort. Simply insert hints (or "directives") in C or Fortran code and the OpenACC compiler runs the code on the GPU.

- **Simple:** Insert compiler hints to instantly tap into thousands of computational cores in the GPU
- **Powerful:** Delivers up to 10x faster application performance
- **Free:** The OpenACC Toolkit with compiler included is available at no charge for academia*

Application Acceleration with OpenACC on GPUs



Modelo OPENACC

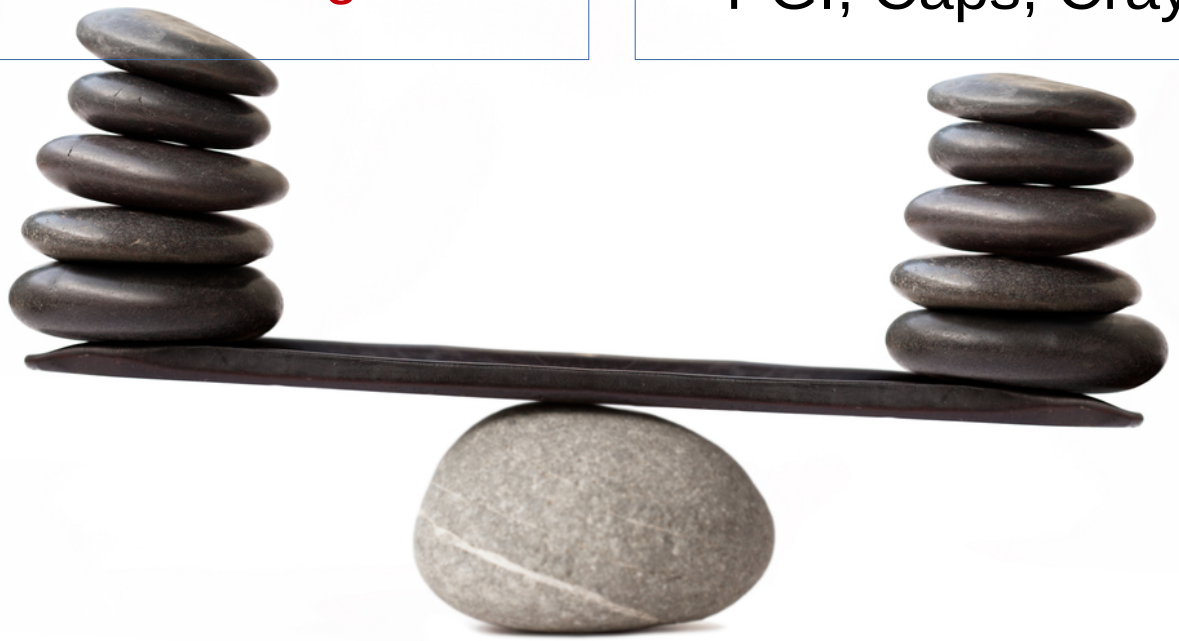


For developers coming to OpenACC from other accelerator programming models, such as CUDA or OpenCL, where host and accelerator memory is frequently represented by two distinct variables (host_A[] and device_A[], for instance), it's important to remember that ***when using OpenACC a variable should be thought of as a single object****, regardless of whether the it's backed by memory in one or more memory spaces.*

Ventajas y desventajas

- Alto nivel
- Muy simple
- Portable
- Fácil de acelerar código serial

- Performance menor a la posible con optimizaciones en CUDA y alguna de sus libs.
- Por ahora, solo compiladores PGI, Caps, Cray (¿gnu...?).



La performance puede ser mejorada gracias a su interoperabilidad con CUDA y sus libs, sacrificando portabilidad...

¡ OPENACC en GCC !

Accelerators

Open Source
GCC Efforts by Samsung & Mentor Graphics

Pervasive Impact
Free to all Linux users

Mainstream
Most Widely Used HPC Compiler

> *Incorporating OpenACC into GCC is an excellent example of open source and open standards working together to make accelerated computing broadly accessible to all Linux developers.*
>
> Oscar Hernandez
> Oak Ridge National Laboratories

OAK RIDGE National Laboratory

OpenACC Directives For Accelerators

Installation of GCC 5.0.1 with OpenACC support

Keep in mind that the GCC 5.0.1 is the first release of GCC that supports OpenACC. According to the `man gcc`

> // Note that this is an experimental feature, incomplete, and subject to change in future versions of GCC. See https://gcc.gnu.org/wiki/OpenACC for more information.

The installation will be more complicated than the standard CPU only installation. To use OpenACC on GPU the compiler needs to generate binary that can execute on CPU and GPU which are two very different architectures. Two compilers will need to be installed together with their dependencies. The first compiler will be for NVIDIA accelerator, the second will be the host compiler for CPU which will be the accelerator compiler aware and linked.

Prepare for the installation

Prerequisites

- working GCC installation (we have used 4.8.1)
- GCC dependencies GMP 4.3.2, MPFR 2.4.2, MPC 0.8.1
- CUDA toolkit (we have used 6.0.37)

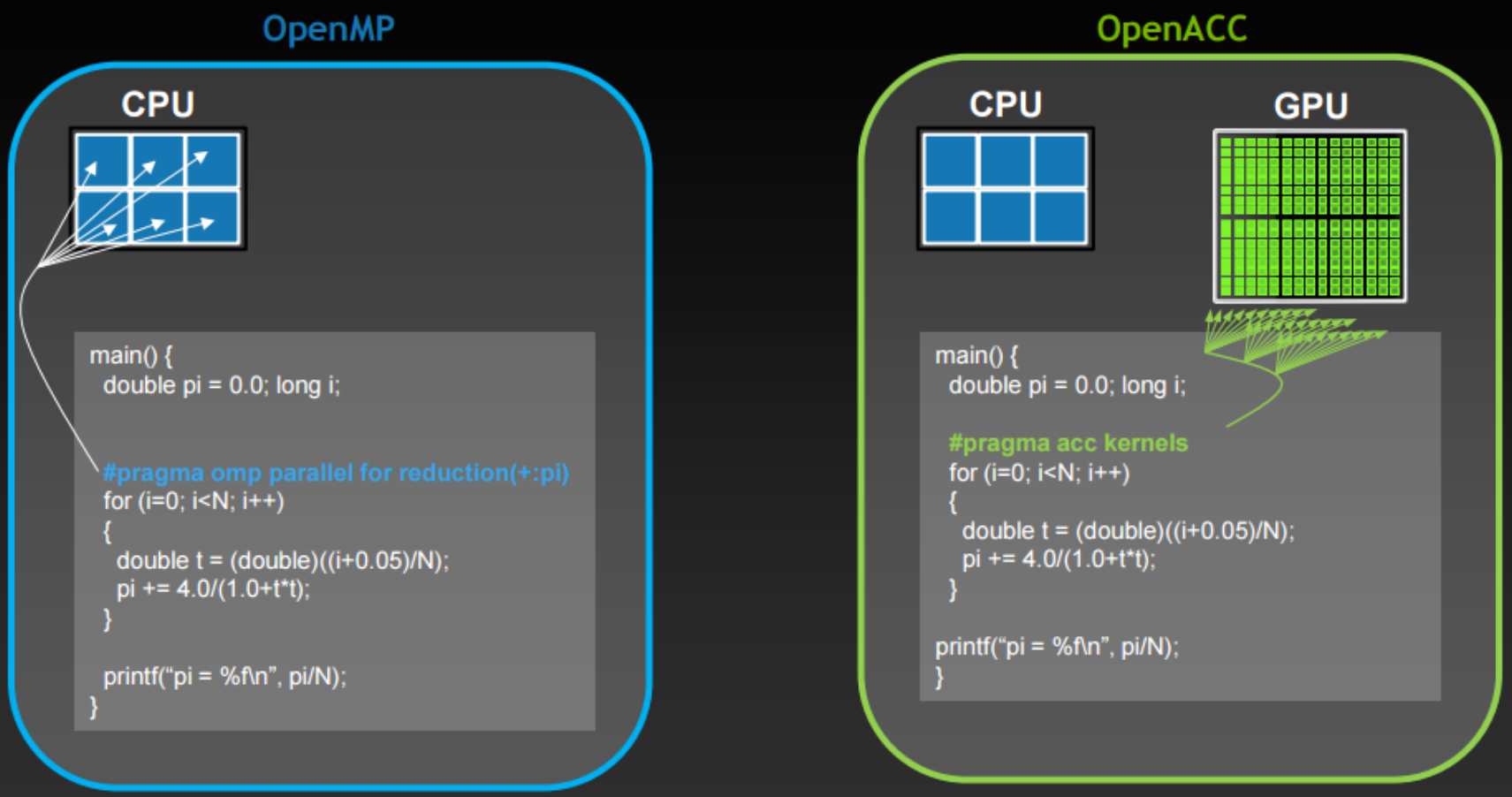
http://scelementary.com/2015/04/25/openacc-in-gcc.html

ACTUALIZACION 2017

https://gcc.gnu.org/wiki/OpenACC

OPENACC y OPENMP

Familiar to OpenMP Programmers



$$\int_0^1 dx \, \frac{4}{1+x^2} = \pi$$

Ecuación de Laplace

- Ecuación de difusión
- Ecuación del calor
- Membrana elástica
- Electrostática...
- Etc.

$$V = f_N(x)$$

$$V = f_O(y)$$

$$(\partial_x^2 + \partial_y^2)V = 0$$

$$V(x,y) = ?$$

$$V = f_E(y)$$

$$V = f_S(x)$$

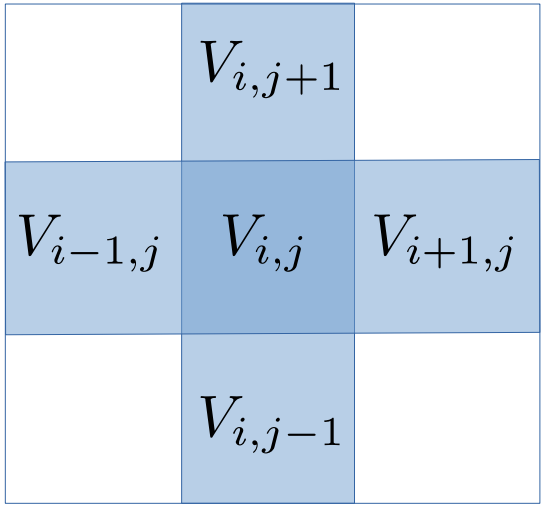
Método de Jacobi

$$(\partial_x^2 + \partial_y^2)V = 0$$

$$V(x = ih, y = jh) = V_{i,j},$$

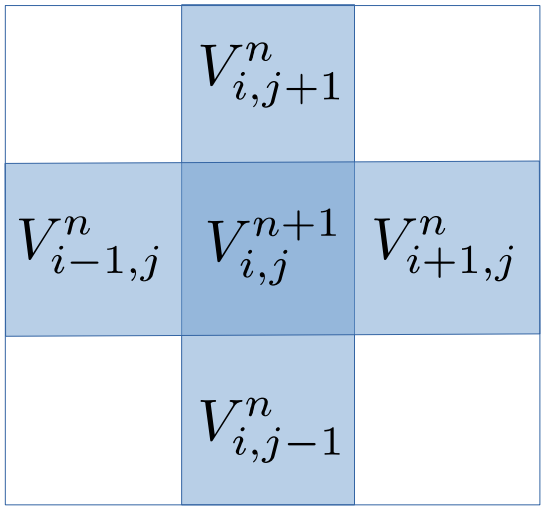
$$V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{i,j} \approx 0$$

$$V_{i,j} \approx \frac{1}{4}(V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1})$$



Método de Jacobi

for(n=0....) $$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^{n} + V_{i-1,j}^{n} + V_{i,j+1}^{n} + V_{i,j-1}^{n})$$

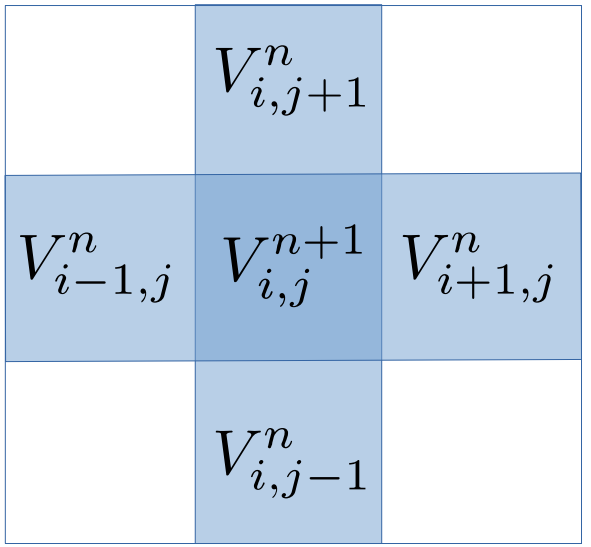


Se puede demostrar que converge a una única solución cuando $n \to \infty$

$$V_{i,j}^{*} = \frac{1}{4}(V_{i+1,j}^{*} + V_{i-1,j}^{*} + V_{i,j+1}^{*} + V_{i,j-1}^{*}) \Rightarrow [\partial_x^2 + \partial_y^2]V^{*}(x,y) = 0$$

Método de Jacobi

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^{n} + V_{i-1,j}^{n} + V_{i,j+1}^{n} + V_{i,j-1}^{n})$$

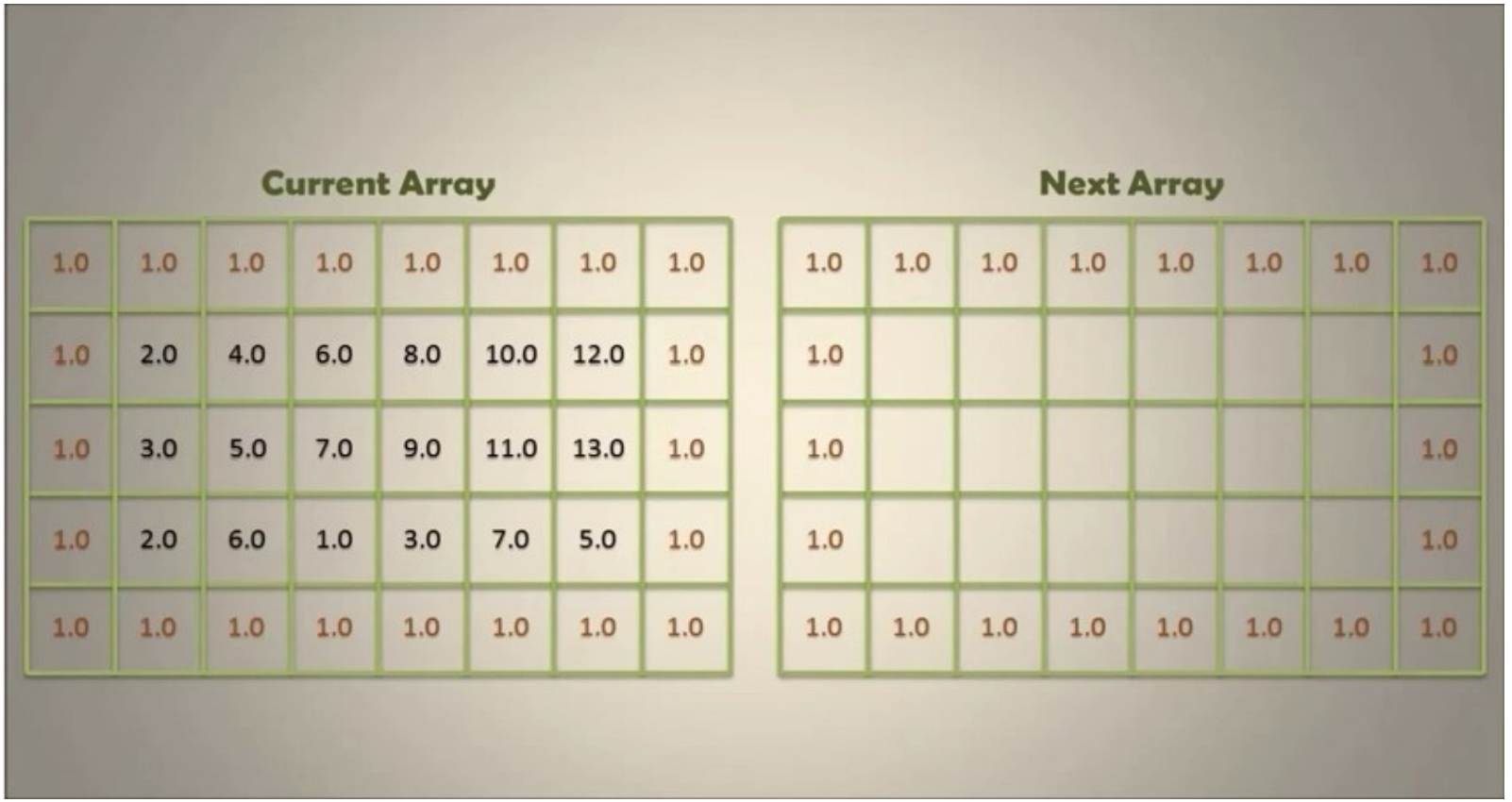


$$\partial_t V = [\partial_x^2 + \partial_y^2]V \Rightarrow V_{i,j}^{n+1} = V_{i,j}^{n} + \frac{\Delta t}{h^2}[V_{i+1,j}^{n} + V_{i-1,j}^{n} + V_{i,j+1}^{n} + V_{i,j+1}^{n} - 4V_{i,j}^{n})]$$

$$\Delta t/h^2 = 1/4 \rightarrow V_{i,j}^{n+1} = V_{i+1,j}^{n} + V_{i-1,j}^{n} + V_{i,j+1}^{n} + V_{i,j-1}^{n}$$

Algoritmo

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^{n} + V_{i-1,j}^{n} + V_{i,j+1}^{n} + V_{i,j-1}^{n})$$



Programa serial

Jacobi Iteration C Code

```
while ( error > tol && iter < iter_max ) {
  error=0.0;

  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {


      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);


      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

- Iterate until converged
- Iterate across matrix elements
- Calculate new value from neighbors
- Compute max error for convergence
- Swap input/output arrays

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Programa serial

Jacobi Iteration Fortran Code

```
do while ( err > tol .and. iter < iter_max )
  err=0._fp_kind

  do j=1,m
    do i=1,n


      Anew(i,j) = .25_fp_kind * (A(i+1, j  ) + A(i-1, j  ) + &
                                 A(i  , j-1) + A(i  , j+1))


      err = max(err, Anew(i,j) - A(i,j))
    end do
  end do

  do j=1,m-2
    do i=1,n-2
      A(i,j) = Anew(i,j)
    end do
  end do

  iter = iter +1
end do
```

- Iterate until converged
- Iterate across matrix elements
- Calculate new value from neighbors
- Compute max error for convergence
- Swap input/output arrays

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Manos a la obra...

Manos a la obra...

- **module load pgi**
- **cp -a /share/apps/codigos/alumnos_icnpg2017/clase_08/* .**
- **cd jacobi/paso0**
- *Ver* las versiones seriales: laplace2d.c y laplace2d.f90
- Compilarlos*:*
  - **pgcc laplace2d.c -o laplace2d -I../common/**
  - **pgf90 laplace2d.f90 -o laplace2d -I../common/**
- Correr:
  - Ver submit_cpu.sh
  - **qsub submit_cpu.sh**
- Analizar output submit_cpu.sh.oxxxx
- *¿Donde se consume casi todo el tiempo de la corrida?*

OPENMP → paralelización CPU multicore

OpenMP C Code

```c
while ( error > tol && iter < iter_max ) {
  error=0.0;

#pragma omp parallel for shared(m, n, Anew, A)
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma omp parallel for shared(m, n, Anew, A)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

OpenMP Fortran Code

```fortran
do while ( err > tol .and. iter < iter_max )
  err=0._fp_kind

!$omp parallel do shared(m,n,Anew,A) reduction(max:err)
  do j=1,m
    do i=1,n

      Anew(i,j) = .25_fp_kind * (A(i+1, j  ) + A(i-1, j  ) + &
                                 A(i  , j-1) + A(i  , j+1))

      err = max(err, Anew(i,j) - A(i,j))
    end do
  end do

!$omp parallel do shared(m,n,Anew,A)
  do j=1,m-2
    do i=1,n-2
      A(i,j) = Anew(i,j)
    end do
  end do

  iter = iter +1
end do
```

→ *Directivas para indicar paralelización de loops en hilos de la CPU*

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

OPENMP vs OPENACC

mismo código, distinta compilación

C

```
...
#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
for( int j = 1; j < n-1; j++)
{
      for( int i = 1; i < m-1; i++ )
      {
            Anew[j][i] = 0.25f * ( A[j][i+1] +
                  A[j][i-1] + A[j-1][i] + A[j+1][i]);
            error = fmaxf( error,
                        fabsf(Anew[j][i]-A[j][i])
                  );
      }
}
...
```

Fortran

```
...
!$omp do reduction( max:error )
!$acc kernels
do j=1,m-2
  do i=1,n-2
      Anew(i,j) = 0.25_fp_kind * ( A(i+1,j  ) + A(i-1,j  ) + &
      A(i  ,j-1) + A(i  ,j+1) )
      error = max( error, abs(Anew(i,j)-A(i,j)) )
  end do
end do
!$acc end kernels
!$omp end do
...
```

cd /share/apps/codigos/alumnos_icnpg2016/clase_08/paso1;

- make
- Openacc → laplace2d_acc
- Openmp → laplace2d_omp
- Serial → laplace2d

- make -f Makefile_f90
- Openacc → laplace2d_f90_acc
- Openmp → laplace2d_f90_omp
- Serial → laplace2d

Openacc → pgcc, pgf90
Openmp → pgcc, gcc, gfortran,...

Serial en C: pgcc -I../common -fast -o laplace2d_omp laplace2d.c
Serial en f90: pgcc -I../common -fast -o laplace2d_omp laplace2d.c
Openmp en C: pgcc -I../common -fast -mp -Minfo -o laplace2d_omp laplace2d.c
Openacc en C: pgcc -I../common -acc -ta=nvidia,time -Minfo=accel -o laplace2d_acc laplace2d.c
Openmp en C: pgf90 -fast -mp -Minfo -o laplace2d_f90_omp laplace2d.f90
Openacc en f90: pgf90 -acc -ta=nvidia -Minfo=accel -o laplace2d_f90_acc laplace2d.f90

Paso 1

- make laplace2d_omp

```
pgcc -I../common -fast -mp -Minfo  -o laplace2d_omp laplace2d.c
main:
    53, Loop not fused: dependence chain to sibling loop
        Generated an alternate version of the loop
        Generated vector sse code for the loop
    59, Loop not fused: function call before adjacent loop
        Generated vector sse code for the loop
    72, Parallel region activated
    80, Parallel loop activated with static block schedule
        Generated an alternate version of the loop
        Generated vector sse code for the loop
    85, Barrier
    87, Parallel region terminated
    88, Parallel region activated
        Parallel loop activated with static block schedule
        Generated vector sse code for the loop
    94, Barrier
        Parallel region terminated
   100, Parallel region activated
        Parallel loop activated with static block schedule
   102, Generated an alternate version of the loop
        Generated vector sse code for the loop
   110, Begin critical section
        End critical section
        Barrier
        Parallel region terminated
   112, Parallel region activated
        Parallel loop activated with static block schedule
   114, Memory copy idiom, loop replaced by call to __c_mcopy4
   120, Barrier
        Parallel region terminated
```

```
#pragma omp parallel shared(Anew)
{
    int tid = omp_get_thread_num();
    if (tid == 0)
    {
            int nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
    }
#pragma omp parallel for shared(Anew)
    for (int j = 1; j < n; j++)
    {
        Anew[#pragma omp parallel for shared(m, n, Anew, A) reduction(max:error)
        Anew[#pragma acc kernels
    }
                for( int j = 1; j < n-1; j++)
                {
                    for( int i = 1; i < m-1; i++ )
                    {
                        Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                             + A[j+1][i] + A[j-1][i]);
                        error = max(error, fabsf(Anew[j][i]-A[j][i]));
#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

- make laplace2d_acc

```
pgcc -I../common -acc -ta=nvidia,time -Minfo=accel -o laplace2d_acc
laplace2d.c
main:
    99, Generating copyout(Anew[1:4094][1:4094])
        Generating copyin(A[:4096][:4096])
   100, Loop is parallelizable
   102, Loop is parallelizable
```

```
y */
) /* blockIdx.x threadIdx.x */
94])
dx.x */
```

Paso 1

- make laplace2d_omp

```
pgcc -I../common -fast -mp -Minfo  -o laplace2d_omp laplace2d.c
main:
     53,
       C
       C
     59,
       C
     72,
     80,
       C
       C
     85, Barrier
     87, Parallel region termina
     88, Parallel region activat
         Parallel loop activated v
         Generated vector sse c
     94, Barrier
         Parallel region terminat
    100, Parallel region activa
         Parallel loop activated with static block schedule
    102, Generated an alternate version of the loop
         Generated vector sse code for the loop
    110, Begin critical section
         End critical section
         Barrier
         Parallel region terminated
    112, Parallel region activated
         Parallel loop activated with static block schedule
    114, Memory copy idiom, loop replaced by call to __c_mcopy4
    120, Barrier
         Parallel region terminated
```

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }
```

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```

- make laplace2d_acc

```
pgcc -I../common -acc -ta=nvidia,time -Minfo=accel -o laplace2d_acc
laplace2d.c
main:
     99, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:4096][:4096])
    100, Loop is parallelizable
    102, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
        100, #pragma acc loop gang /* blockIdx.y */
        102, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
        106, Max reduction generated for error
    111, Generating copyin(Anew[1:4094][1:4094])
         Generating copyout(A[1:4094][1:4094])
    112, Loop is parallelizable
    114, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
        112, #pragma acc loop gang /* blockIdx.y */
        114, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

Paso 1

Hola chicos! Ayudenme a completar esta tabla

Ejecución	Tiempo	Aceleración respecto al serial
Serial		
Openmp 2 threads		
Openmp 3 threads		
Openmp 4 threads		
OpenACC		

¿Qué paso?

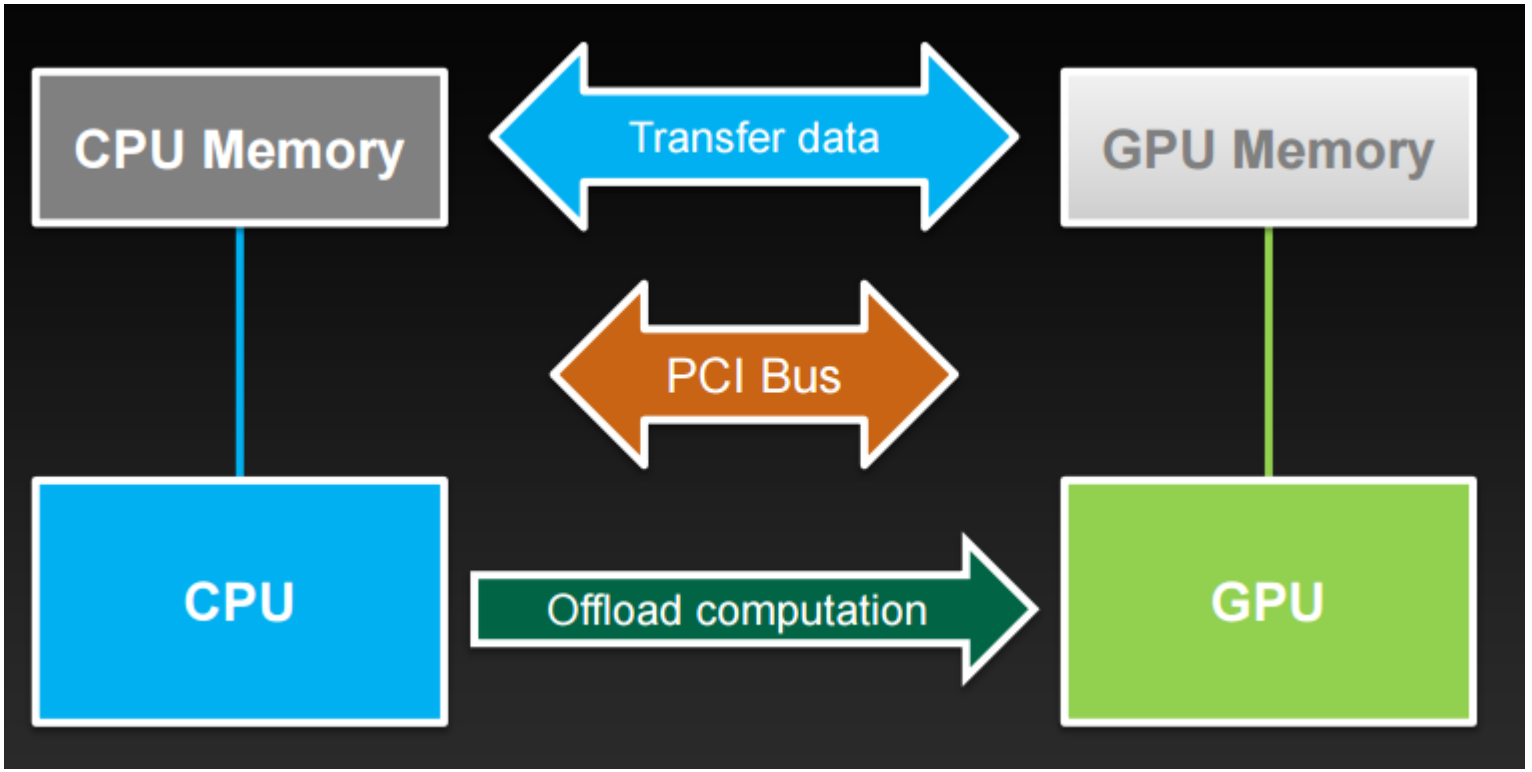
nvprof ./laplace2d_acc

```
==7093== Profiling application: ./laplace2d_acc
==7093== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 42.54%  24.2059s      9000  2.6895ms  3.4870us  3.8239ms  [CUDA memcpy DtoH]
 41.52%  23.6242s      9000  2.6249ms     959ns  3.5642ms  [CUDA memcpy HtoD]
 10.27%  5.84493s      1000  5.8449ms  5.8436ms  5.8544ms  main_102_gpu
  5.20%  2.95895s      1000  2.9590ms  2.9559ms  2.9621ms  main_114_gpu
  0.46%  263.71ms      1000  263.71us  262.65us  265.52us  main_106_gpu_red

==7093== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.05%  57.1905s      44996  1.2710ms     404ns  5.8903ms  cuEventSynchronize
```

¡¡¡¡ Excesivo movimiento de datos !!!!!

El precio de transferir...



http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

CUDA Application Design and Development, R. Farber

THREE RULES OF GPGPU PROGRAMMING

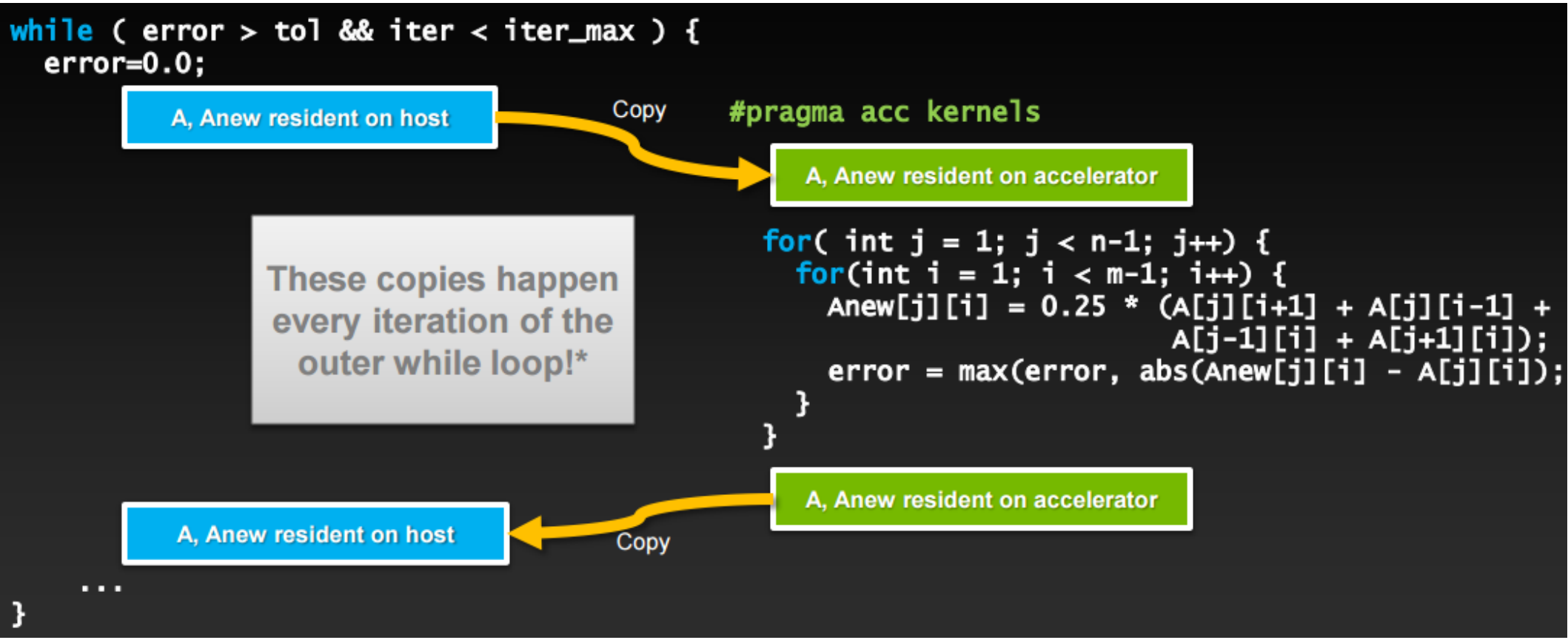
Observation has shown that there are three general rules to creating high-performance GPGPU programs:

1. Get the data on the GPGPU and keep it there.
2. Give the GPGPU enough work to do.
3. Focus on data reuse within the GPGPU to avoid memory bandwidth limitations.

Básicamente es entender las latencias...

¿Compilador precavido o tonto?

*¿Necesitamos **A** y **Anew** en el host cada iteración de Jacobi?*



http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

¿Compilador precavido o tonto?

*¿Necesitamos **A** y **Anew** en el host cada iteración de Jacobi?*

```
while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma omp parallel for shared(m, n, Anew, A) reduction(max:error)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }

#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

¿Que haría Ud aquí si fuera el Compilador?

¿Compilador precavido o tonto?

*¿Necesitamos **A** y **Anew** en el host cada iteración de Jacobi?*

```
do while ( error .gt. tol .and. iter .lt. iter_max )
   error=0.0_fp_kind

!$omp parallel do shared(m, n, Anew, A) reduction( max:error )
!$acc kernels
   do j=1,m-2
     do i=1,n-2
       Anew(i,j) = 0.25_fp_kind * ( A(i+1,j  ) + A(i-1,j  ) + &
                                    A(i  ,j-1) + A(i  ,j+1) )
       error = max( error, abs(Anew(i,j)-A(i,j)) )
     end do
   end do
!$acc end kernels
!$omp end parallel do

   if(mod(iter,100).eq.0 ) write(*,'(i5,f10.6)'), iter, error
   iter = iter +1

!$omp parallel do shared(m, n, Anew, A)
!$acc kernels
   do j=1,m-2
     do i=1,n-2
       A(i,j) = Anew(i,j)
     end do
   end do
!$acc end kernels
!$omp end parallel do

 end do
```

¿Compilador precavido o tonto?

*¿Necesitamos **A** y **Anew** en el host cada iteración de Jacobi?*

```
while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma omp parallel for shared(m, n, Anew, A) reduction(max:error)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }

#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

Che, compilador!, **NO** me traigas los datos al host *aquí* porque **no** los voy a necesitar aún!

¿Que haría Ud aquí si fuera el Compilador?

Explotar localidad de Datos

Defining data regions

- The `data` construct defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```
!$acc data
    do i=1,n
       a(i) = 0.0
       b(i) = 1.0
       c(i) = 2.0
    end do

    do i=1,n
       a(i) = b(i) + c(i)
    end do
!$acc end data
```

Data Region

Arrays a, b, and c will remain on the GPU until the end of the data region.

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Localidad de los datos

ANTES COPIABAMOS 4 VECES CADA ITERACION

```
while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma omp parallel for shared(m, n, Anew, A)
reduction(max:error)
```

Aquí el compilador copia A, Anew de host a device

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }
```

Aquí el compilador, por las dudas, actualiza A, Anew en el host

```
#pragma omp parallel for shared(m, n, Anew, A)
```

Aquí el compilador copia A, Anew de host a device

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```

Aquí el compilador, por las dudas, actualiza A, Anew en el host

```
        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

AHORA NOS AHORRAMOS UN MONTON DE COPIAS

Aquí el compilador copia A, Anew al device

```
#pragma acc data copy(A, Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }
#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

Recién aquí el compilador actualiza A, Anew en el host como implica el “copy”

Data Regions

Data Construct

Fortran

```
!$acc data [clause …]
    structured block
!$acc end data
```

C

```
#pragma acc data [clause …]
    { structured block }
```

General Clauses

```
if( condition )
async( expression )
```

Manage data movement. Data regions may be nested.

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Data Regions

Data Clauses

`copy ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.
`copyin ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region.
`copyout ( list )`	Allocates memory on GPU and copies data to the host when exiting region.
`create ( list )`	Allocates memory on GPU but does not copy.
`present ( list )`	Data is already present on GPU from another containing data region.

and `present_or_copy[in|out]`, `present_or_create`, `deviceptr`.

Data Regions

Array Shaping

- Compiler sometimes cannot determine size of arrays
  - Must specify explicitly using data clauses and array “shape”
- C

```
#pragma acc data copyin(a[0:size-1]), copyout(b[s/4:3*s/4])
```

- Fortran

```
!$pragma acc data copyin(a(1:size)), copyout(b(s/4:3*s/4))
```

- Note: data clauses can be used on data, kernels or parallel

Data Regions

Update Construct

Fortran

```
!$acc update [clause …]
```

C

```
#pragma acc update [clause …]
```

Clauses

```
host( list )
device( list )
if( expression )
async( expression )
```

Used to update existing data after it has changed in its corresponding copy (e.g. update device copy after host copy changes)

Move data from GPU to host, or host to GPU.
Data movement can be conditional, and asynchronous.

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

¿Compilador precavido o tonto ?

ANTES

```
while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma omp parallel for shared(m, n, Anew, A)
reduction(max:error)
```

Aquí el compilador copia A, Anew de host a device

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }
```

Aquí el compilador, por las dudas, actualiza A, Anew en el host

```
#pragma omp parallel for shared(m, n, Anew, A)
```

Aquí el compilador copia A, Anew de host a device

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```

Aquí el compilador, por las dudas, actualiza A, Anew en el host

```
        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

Aquí el compilador copia A, Anew al device

```
#pragma acc data copy(A, Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }
#pragma omp parallel for shared(m, n, Anew, A)
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

Recién aquí el compilador actualiza A, Anew en el host como implica el “copy”

Manos a la obra...

- **module load pgi**
- **cp -a /share/apps/codigos/alumnos_icnpg2017/clase_08/* .**
- **cd jacobi/paso1**
- *Ver* (*de nuevo)* laplace2d.c y laplace2d.f90
- *Ver (de nuevo)* Makefile
- Compilar *laplace2d_omp laplace2d_acc laplace2d:*
  - **make**
- Prestar atención al *output de make*!
- *Correr*:
  - *Controlar* submit_cpuNslots.sh, submit_gpu.shsubmit_gpu.sh
  - **qsub submit_cpu.sh; qsub submit_gpu.sh**
- Analizar output submit_xxx.sh.oxxxx

Paso 2

Ejecución	Tiempo	Acelera al serial
Serial		
Openmp 2 threads		
Openmp 3 threads		
Openmp 4 threads		
OpenACC		

~15 X

¿Qué paso?

Antes [~150 s, **0.19** x]

```
==7093== Profiling application: ./laplace2d_acc
==7093== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 42.54%  24.2059s      9000  2.6895ms  3.4870us  3.8239ms  [CUDA memcpy DtoH]
 41.52%  23.6242s      9000  2.6249ms     959ns  3.5642ms  [CUDA memcpy HtoD]
 10.27%  5.84493s      1000  5.8449ms  5.8436ms  5.8544ms  main_102_gpu
  5.20%  2.95895s      1000  2.9590ms  2.9559ms  2.9621ms  main_114_gpu
  0.46%  263.71ms      1000  263.71us  262.65us  265.52us  main_106_gpu_red

==7093== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.05%  57.1905s      44996  1.2710ms     404ns  5.8903ms  cuEventSynchronize
```

Ahora [~4 s, **15** x]

```
==6200== Profiling application: ./laplace2d_acc
==6200== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 61.57%  2.36598s      1000  2.3660ms  2.3647ms  2.3704ms  main_94_gpu
 29.62%  1.13818s      1000  1.1382ms  1.1365ms  1.1402ms  main_106_gpu
  7.54%  289.84ms      1000  289.84us  289.09us  290.98us  main_98_gpu_red
  0.70%  26.767ms      1008  26.554us     896ns  3.9257ms  [CUDA memcpy HtoD]
  0.58%  22.279ms      1010  22.058us  2.0800us  2.5660ms  [CUDA memcpy DtoH]

==6200== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 90.78%  3.86609s      5042  766.78us     950ns  3.9326ms  cuEventSynchronize
```

Expresar paralelismo + Explotar localidad de datos =

~15 X

Consejo: 4 pasos

- **Identificar el Paralelismo:**

Mirar, donde el programa pierde mas tiempo, si hay posibilidad de paralelización (loops, loops anidados, etc)

- **Expresar el Paralelismo:**

Poner directivas en los loops para avisarle al compilador donde puede paralelizar. Generalmente da lugar a un código más lento (paso 2)... pero ¡a no desalentarse!.

- **Expresar localidad de datos:**

El compilador no es tan astuto pero es precavido, no puede tomar ciertas decisiones por uno. Decirle entonces como y cuando los datos son necesarios en la GPU o CPU (paso 2).

- **Optimizar:**

Una ayudita extra para mejorar la performance (paso 3), restructurar código o mejorar los patrones de acceso o usar info del hardware... Si no esta satisfecho con las posibilidades que brinda openacc programe la parte crítica en CUDA!, y sacrifique un poco la portabilidad...

https://devblogs.nvidia.com/parallelforall/getting-started-openacc/

Openacc API

Directive Syntax

- Fortran

```
!$acc directive [clause [,] clause] …]
```

Often paired with a matching end directive surrounding a structured code block

```
!$acc end directive
```

- C

```
#pragma acc directive [clause [,] clause] …]
```

Often followed by a structured code block

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Openacc API

Kernels Construct

Fortran

```
!$acc kernels [clause …]
    structured block
!$acc end kernels
```

C

```
#pragma acc kernels [clause …]
    { structured block }
```

Clauses

```
if( condition )
async( expression )
```

Also any data clause

Kernels Construct

Each loop executed as a separate kernel on the GPU.

```
!$acc kernels
    do i=1,n
       a(i) = 0.0
       b(i) = 1.0
       c(i) = 2.0
    end do

    do i=1,n
       a(i) = b(i) + c(i)
    end do
!$acc end kernels
```

kernel 1

kernel 2

```
#pragma acc data copy(A, Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-
1]
                                     + A[j-1][i] + A[j+1]
[i]);
                error = fmaxf( error, fabsf(Anew[j][i]-
A[j][i]));
            }
        }

#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter,
error);

        iter++;
    }
```

http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

Openacc API

Loop Construct

Fortran

```
!$acc loop [clause …]
    loop
!$acc end loop
```

C

```
#pragma acc loop [clause …]
    { loop }
```

Combined directives

```
!$acc parallel loop [clause …]
!$acc kernels loop [clause …]
```

```
!$acc parallel loop [clause …]
!$acc kernels loop [clause …]
```

Detailed control of the parallel execution of the following loop.

Loop Clauses Inside parallel Region

`gang`	Shares iterations across the gangs of the parallel region.
`worker`	Shares iterations across the workers of the gang.
`vector`	Execute the iterations in SIMD mode.

Loop Clauses

`collapse( n )`	Applies directive to the following `n` nested loops.
`seq`	Executes the loop sequentially on the GPU.
`private( list )`	A copy of each variable in list is created for each iteration of the loop.
`reduction( operator:list )`	`private` variables combined across iterations.

```
#pragma acc data copy(A), create(Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma acc parallel loop collapse(2) reduction(max:error)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }

#pragma acc parallel loop collapse(2)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

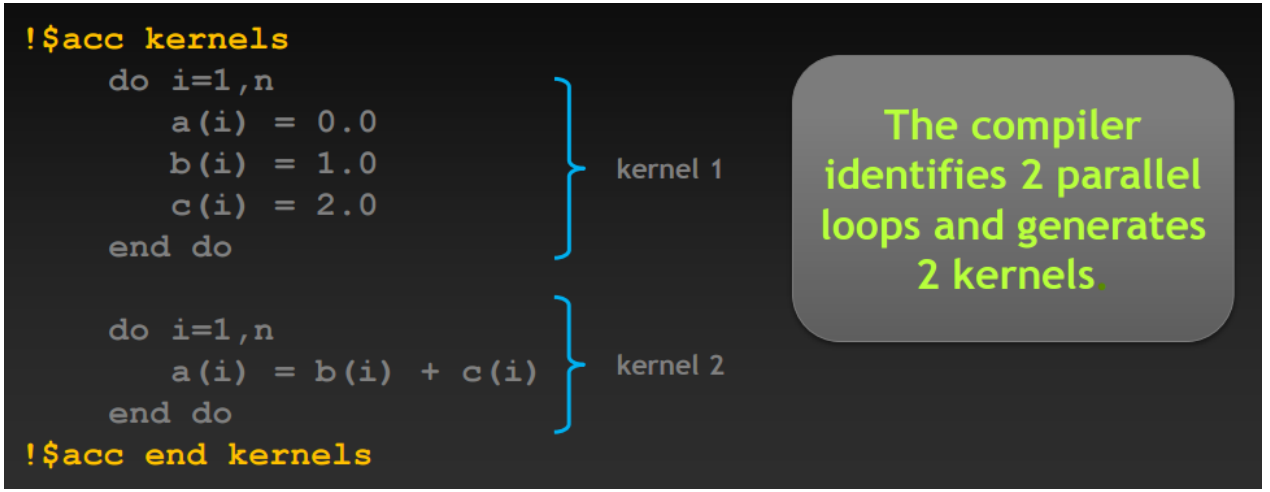
http://lyoncalcul.univ-lyon1.fr/IMG/pdf/00_-_GPU_programming_-_OpenACC_Hands_On.pdf

¿Parallel o Kernel?

- ## Parallel
  - Prescriptivo (como openmp)
  - Usa un kernel en la región
  - El compilador va a acelerar la región *aun si da resultados incorrectos*
  - *Mayor control*

- ## Kernels
  - Descriptivo
  - Usa uno o mas kernels en la región
  - *Más seguro*: podría no acelerar nada si hay ambiguedad
  - *Mejor para empezar* a explorar paralelismo

Ambos definen regiones a ser aceleradas, pero la obligación al compilador es diferente



Se trata de agregar #pragma ...

!$acc kernels	!$acc parallel	!$acc data	!$acc loop	!$acc wait
#pragma acc kernels	#pragma acc parallel	#pragma acc data	#pragma acc loop	#pragma acc wait
Clauses	**Clauses**	**Clauses**	**Clauses**	
if()	if()	if()	collapse()	
async()	async()	async()	within kernels region	
copy()	num_gangs()		gang()	
copyin()	num_workers()		worker()	
copyout()	vector_length()		vector()	
create()	reduction()		seq()	
present()	copyin()	copyin()	private()	
present_or_copy()	copyout()	copyout()	reduction()	
present_or_copyin()	create()	create()		
present_or_copyout()	present()	present()		
present_or_create()	present_or_copy()	deviceptr() in .c		
deviceptr()	present_or_copyin()	deviceptr() in .f		
	present_or_copyout()			
	present_or_create()			
	deviceptr()			
	private()			
	firstprivate()			

```
#pragma acc data copy(A), create(Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma acc parallel loop collapse(2) reduction(max:error)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }

#pragma acc parallel loop collapse(2)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

http://www.openacc.org/sites/default/files/OpenACC_2pt5.pdf

Referencias

http://www.openacc.org/content/education

OpenACC Directives for Accelerators

- OpenACC: Directives for GPUs
- FREE OpenACC Online Course
- Six Ways to SAXPY
- An OpenACC Example (Part I, Part II)

OpenACC using the PGI Compilers

- PGI Accelerator Compilers with OpenACC Getting Started Guide (PDF)
- OpenACC Kernels and Parallel Constructs
- The PGI Accelerator Compilers with OpenACC
- 5X in 5 Hours: Porting a 3D Elastic Wave Simulator using OpenACC
- Understanding the CUDA Data Parallel Threading Model

OpenACC using the Cray Compilers

- A Quick Tour of OpenACC
- OpenACC and the Cray Compilation Environment

Developer Support Forums

- Stackoverflow
- PGI

Seminars & Classes

Europe

- CINECA SCAI
- HLRS
- NVIDIA Europe
- PRACE

North America

- Univ. of Houston
- XSEDE

To add a seminar, email training@openacc.org

Videos

- Introduction to OpenACC
- Unstructured Data Lifetimes in OpenACC 2.0 (CUDAcast)
- C++ Class Management with OpenACC 2.0
- Enabling OpenACC Performance Analysis

Learn More

- Debuggers and Other Tools
- Interoperability with MPI

Consulting & Training Services

- Acceleware
- Applied Parallel Computing
- SagivTech

OpenACC Programming and Best Practices Guide

June 2015

Ejemplo de datos con tiempo de vida estructurado.

```
#pragma acc data copy(A), create(Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.f;

#pragma acc parallel loop collapse(2) reduction(max:error)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                     + A[j-1][i] + A[j+1][i]);
                error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
            }
        }

#pragma acc parallel loop collapse(2)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
```

A, Anew viven aqui arriba

Datos con “tiempo de vida desestructurado”

```
void jacobi(float A[][4096],float Anew[][4096])
{
       int n = 4096;
       int m = 4096;

       int iter = 0;
       const float tol = 1.0e-5f;
       float error     = 1.0f;
       int iter_max = 1000;

       while ( error > tol && iter < iter_max )
       {
           error = 0.f;
#pragma acc parallel loop collapse(2) reduction(max:error)
present(A, Anew)
           for( int j = 1; j < n-1; j++)
           {
               for( int i = 1; i < m-1; i++ )
               {
                   Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                        + A[j-1][i] + A[j+1][i]);
                   error = fmaxf( error, fabsf(Anew[j][i]-A[j]
[i]));
               }
           }
#pragma acc parallel loop collapse(2) present(A, Anew)
           for( int j = 1; j < n-1; j++)
           {
               for( int i = 1; i < m-1; i++ )
               {
                   A[j][i] = Anew[j][i];
               }
           }

           if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

           iter++;
       }
}
```

- enter data
- exit data

```
int main(int argc, char** argv)
{
     // declaracion, inicializacion de A, Anew en el host
     // etc
#pragma acc enter data copyin(A[0:n][0:m],Anew[0:n][0:m])
     jacobi(A,Anew);
     otra_funcion_acelerada(A,Anew);
#pragma acc exit data delete(A[0:n][0:m],Anew[0:n][0:m])
      // declaracion, inicializacion de A, Anew en el
host
       // etc
   }
```

https://devblogs.nvidia.com/parallelforall/cudacasts-episode-17-unstructured-data-lifetimes-openacc-2-0/

OpenACC Programming and Best Practices Guide

June 2015

UPDATE DIRECTIVES:
Sincronizan los contenidos de las memorias que uno quiera, cuando uno quiera

```
int main(int argc, char** argv)
 {
      // declaracion, inicializacion de A, Anew en el host
      // etc
#pragma acc enter data copyin(A,Anew)

      jacobi(A,Anew);

      funcion_acelerada(A,Anew);

#pragma acc update self(A)
      funcion_no_acelerada(A);
#pragma acc update device(A)

      otra_funcion_acelerada(A,Anew);

#pragma acc exit data delete(A[0:n]
[0:m],Anew[0:n][0:m])
    // declaracion, inicializacion de A, Anew en el host
       // etc
 }
```

Best Practice: variables in an OpenACC code should always be thought of as a singular object, rather than a host copy and a device copy ….

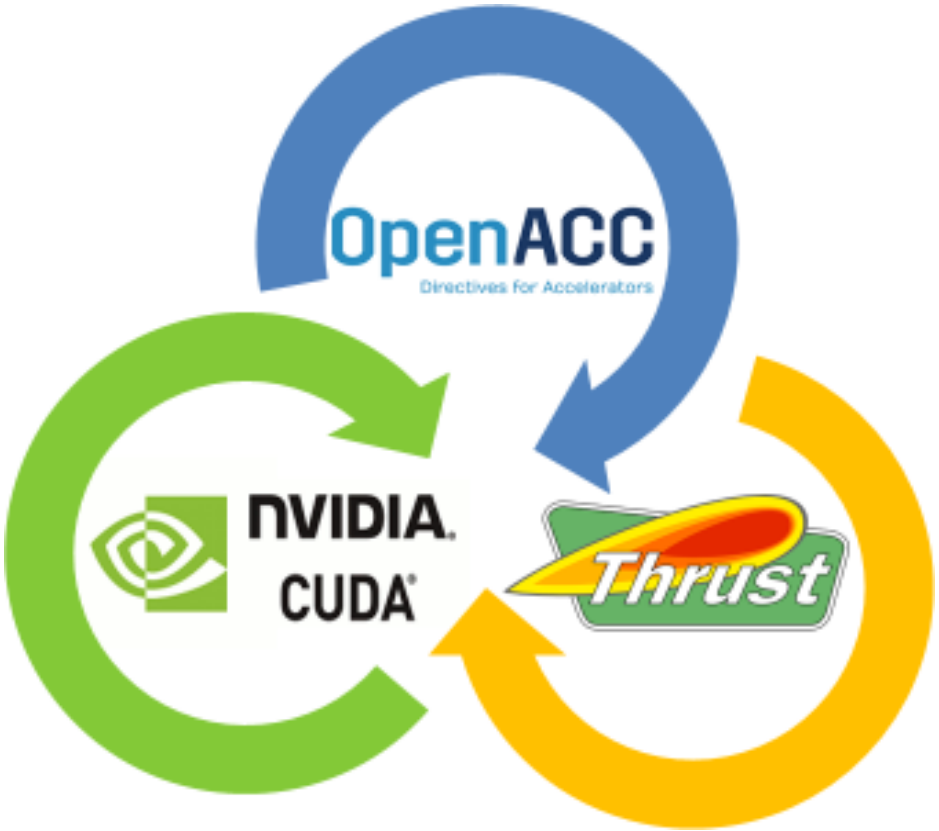
OPENACC y Template Classes C++

```
template<typename vtype> class myvector
{
          vtype* mydata;
          size_t mysize;
  public:
          inline vtype & operator[]( int i ) const {return mydata[i];}
          inline size_t size(){ return mysize; }
          myvector( int size_ ){
              mysize = size_;
              mydata = new vtype[mysize];
              #pragma acc enter data copyin(this)
              #pragma acc enter data create(mydata[0:mysize])
          }
          ~myvector(){
               #pragma acc exit data delete(mydata[0:mysize],this)
              delete [] mydata;
          }
          void updatedev(){
                 #pragma acc update device(mydata[0:mysize])
          }
          void updatehost(){
                 #pragma acc update host(mydata[0:mysize])
          }
 };
```

```
template<typename vtype> void
    axpy( myvector<vtype>& y, myvector<vtype>& x,
vtype a ){
        #pragma acc parallel loop present(x,y)
        for( int i = 0; i < y.size; ++i )
            y[i] += a*x[i];
    }

template<typename vtype> void
    test( myvector<vtype>& a, myvector<vtype>& b ){
        a.updatedev();
        b.updatedev();
        axpy<vtype>( a, b, 2.0 );
        modify( b );
        b.updatedev();
        axpy<vtype>( a, b, 1.0 );
        a.updatehost();
    }
```

Interoperabilidad



- host_data construct
- deviceptr clause
- acc_map_data() API function.

https://devblogs.nvidia.com/parallelforall/3-versatile-openacc-interoperability-techniques/

Usar CUDA (+libs) en un programa OpenACC con *host_data region*

```
…
#pragma acc data create(x[0:n]) copyout(y[0:n])
{
    #pragma acc kernels
    {
        for( i = 0; i < n; i++){
            x[i] = 1.0f;
            y[i] = 0.0f;
        }
    }
    #pragma acc host_data use_device(x,y)
    {
        cublasSaxpy(n, 2.0, x, 1, y, 1);
    // kernel<<<..,..>>>(x,y,n);
    //
thrust::transform(thrust::device,x,x+n,y,y);
    // etc...
    }
}
...
```

```
program main
  use cublas
  integer, parameter :: N = 2**20
  real, dimension(N) :: X, Y

  !$acc data create(x) copyout(y)

  !$acc kernels
  X(:) = 1.0
  Y(:) = 0.0
  !$acc end kernels

  !$acc host_data use_device(x,y)
  call cublassaxpy(N, 2.0, x, 1, y, 1)
  !$acc end host_data

  !$acc end data

  print *, y(1)
end program
```

The host_data region gives the programmer a way to *expose the device address* of a given array to the host for passing into a function. This data **must have already been moved** to the device previously. The host_data region accepts only the use_device clause, which specifies which device variables should be exposed to the host.

http://www.openacc.org/sites/default/files/OpenACC_Programming_Guide_0.pdf

Usar OPENACC en un programa de CUDA con *deviceptr*

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

El array ya esta en device, tomo su puntero...

```
int main(int argc, char **argv)
{
  float *x, *y, tmp;
  int n = 1<<20;

  cudaMalloc((void**)&x,(size_t)n*sizeof(float));
  cudaMalloc((void**)&y,(size_t)n*sizeof(float));

  set(n,1.0f,x);
  set(n,0.0f,y);

  saxpy(n, 2.0, x, y);
  cudaMemcpy(&tmp,y,
(size_t)sizeof(float),cudaMemcpyDeviceToHost);
  printf("%f\n",tmp);
  return 0;
}
```

Usa CUDA API
(podria usar openACC API
o thrust)

In this case OpenACC provides the deviceptr data clause, which may be used where any data clause may appear. This clause informs the compiler that the variables specified are already device on the device and no other action needs to be taken on them.

http://www.openacc.org/sites/default/files/OpenACC_Programming_Guide_0.pdf

Usar OPENACC en un programa de CUDA con *deviceptr*

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

El array ya esta en device, tomo su puntero...

```
int main(int argc, char **argv)
{
      float *x, *y, tmp;
      int n = 1<<20;

      x = acc_malloc((size_t)n*sizeof(float));
      y = acc_malloc((size_t)n*sizeof(float));

      set(n,1.0f,x);
      set(n,0.0f,y);

      saxpy(n, 2.0, x, y);
      acc_memcpy_from_device(&tmp,y,(size_t)sizeof(float));
      printf("%f\n",tmp);
      acc_free(x);
      acc_free(y);
      return 0;
}
```

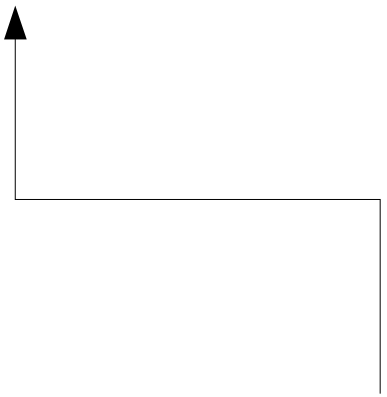
Usa Openacc API
(podria usar CUDA API
o thrust ...)

In this case OpenACC provides the deviceptr data clause, which may be used where any data clause may appear. This clause informs the compiler that the variables specified are already device on the device and no other action needs to be taken on them.

http://www.openacc.org/sites/default/files/OpenACC_Programming_Guide_0.pdf

Usar OPENACC en un programa de Thrust con *deviceptr*

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



```
int main(int argc, char **argv)
{
  int N = 1<<20;
  thrust::host_vector y(N);

  thrust::device_vector d_x(N);
  thrust::device_vector d_y(N);

  thrust::fill(d_x.begin(),d_x.end(), 1.0f);
  thrust::fill(d_y.begin(),d_y.end(), 0.0f);

  saxpy(N, 2.0,
thrust::raw_pointer_cast(d_x.data()),
        thrust::raw_pointer_cast(d_y.data()));

  y = d_y;
  printf("%f\n",y[0]);
  return 0;
}
```

Usar OPENACC en un programa de CUDA con *acc_map_data()*

```
void map(float * restrict harr, float * restrict darr, int size)
{
  acc_map_data(harr, darr, size);
}

void saxpy(int n, float a, float * restrict x, float * restrict y)
{
  // x , y estan presentes en device
  // porque llamamos a acc_map_data()
  #pragma acc kernels present(x,y)
  {
      for(int i=0; i<n; i++)
      {
            y[i] += a*x[i];
      }
  }
}
void set(int n, float val, float * restrict arr)
{
      #pragma acc kernels present(x,y)
      {
            for(int i=0; i<n; i++)
            {
                  arr[i] = val;
            }
      }
}
```

```
int main(int argc, char **argv)
{
  float *x, *y, *dx, *dy, tmp;
  int n = 1<<20;

  x = (float*) malloc(n*sizeof(float));
  y = (float*) malloc(n*sizeof(float));
  cudaMalloc((void**)&dx,(size_t)n*sizeof(float));
  cudaMalloc((void**)&dy,(size_t)n*sizeof(float));

  map(x, dx, n*sizeof(float));
  map(y, dy, n*sizeof(float));

  set(n,1.0f,x);
  set(n,0.0f,y);

  saxpy(n, 2.0, x, y);
  cudaMemcpy(&tmp,dy,
(size_t)sizeof(float),cudaMemcpyDeviceToHost);
  printf("%f\n",tmp);
  return 0;
}
```

The acc_map_data routine is a great way to “set it and forget it” when it comes to sharing memory between CUDA and OpenACC ...

http://www.openacc.org/sites/default/files/OpenACC_Programming_Guide_0.pdf

Manos a la obra...

Disney (por las dudas...)

Manos a la obra...

- **cp -a /share/apps/codigos/alumnos_icnpg2016/clases_openacc . ;**
- **cd openacc-interoperability**
- **make *nombre_ejemplo***
- **qsub submit_gpu.sh *nombre_ejemplo***

  - *nombre_ejemplo* es alguno de estos:
  - * **cuda_main** - calling OpenACC from CUDA C
  - * **openacc_c_main** - Calling CUDA from OpenACC in C
  - * **openacc_c_cublas** - Calling CUBLAS from OpenACC in C
  - * **thrust** - Mixing OpenACC and Thrust in C++
  - * **cuda_map** - Using OpenACC acc_map_data with CUDA in C
  - * **cuf_main** - Calling OpenACC from CUDA Fortran
  - * **cuf_openacc_main** - Calling CUDA Fortran from OpenACC
  - * **openacc_cublas** - Calling CUBLAS from OpenACC in CUDA Fortran
  - * **acc_malloc** - Same as cuda_main, but using the OpenACC API
  - * **openacc_streams** - Mixes OpenACC async queues and CUDA streams
  - * **openacc_cuda_device** - Calls a CUDA __device__ kernel within an OpenACC
  - Author: Jeff Larkin <jlarkin@nvidia.com>

Referencias

http://www.openacc.org/content/education

OpenACC Directives for Accelerators

- OpenACC: Directives for GPUs
- FREE OpenACC Online Course
- Six Ways to SAXPY
- An OpenACC Example (Part I, Part II)

OpenACC using the PGI Compilers

- PGI Accelerator Compilers with OpenACC Getting Started Guide (PDF)
- OpenACC Kernels and Parallel Constructs
- The PGI Accelerator Compilers with OpenACC
- 5X in 5 Hours: Porting a 3D Elastic Wave Simulator using OpenACC
- Understanding the CUDA Data Parallel Threading Model

OpenACC using the Cray Compilers

- A Quick Tour of OpenACC
- OpenACC and the Cray Compilation Environment

Developer Support Forums

- Stackoverflow
- PGI

Seminars & Classes

Europe

- CINECA SCAI
- HLRS
- NVIDIA Europe
- PRACE

North America

- Univ. of Houston
- XSEDE

To add a seminar, email training@openacc.org

Videos

- Introduction to OpenACC
- Unstructured Data Lifetimes in OpenACC 2.0 (CUDAcast)
- C++ Class Management with OpenACC 2.0
- Enabling OpenACC Performance Analysis

Learn More

- Debuggers and Other Tools
- Interoperability with MPI

Consulting & Training Services

- Acceleware
- Applied Parallel Computing
- SagivTech

Openacc + avanzado

- Optimización de loops
- Asincronismo
- Multi GPU

Más adelante....