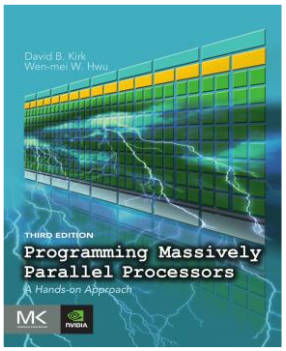
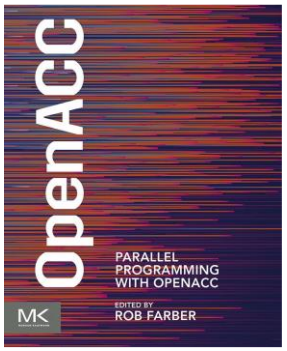
CÓMPUTO PARALELO (OPENACC)

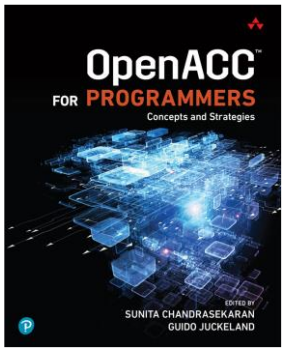
Francisco J. Hernández López

fcoj23@cimat.mx

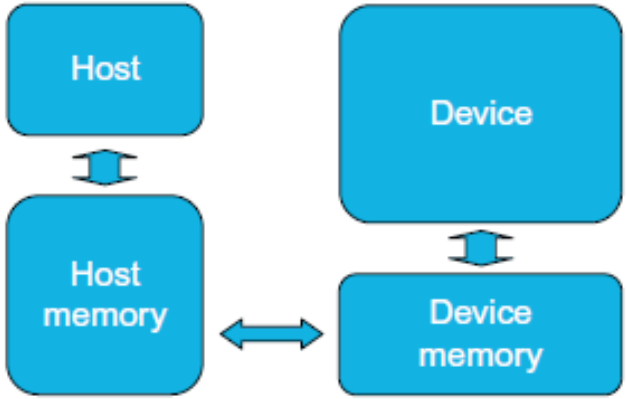
OPENACC

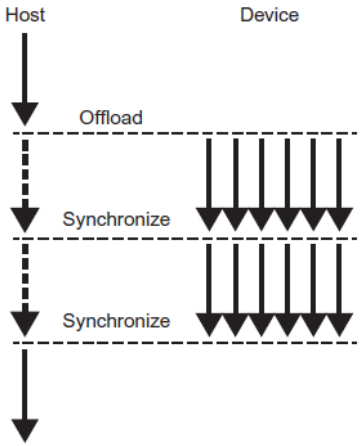






- Es una colección de directivas de compilación, rutinas de la librería y variables de ambiente que pueden utilizarse en C/C++ y Fortran para el cómputo de alto rendimiento (HPC)
- Su modelo de programación asume que la ejecución del programa comenzará en una CPU anfitrión (*host*), la cual puede traspasar (*offloading*) ejecución y datos a un acelerador o coprocesador (Multi-core o GPU)



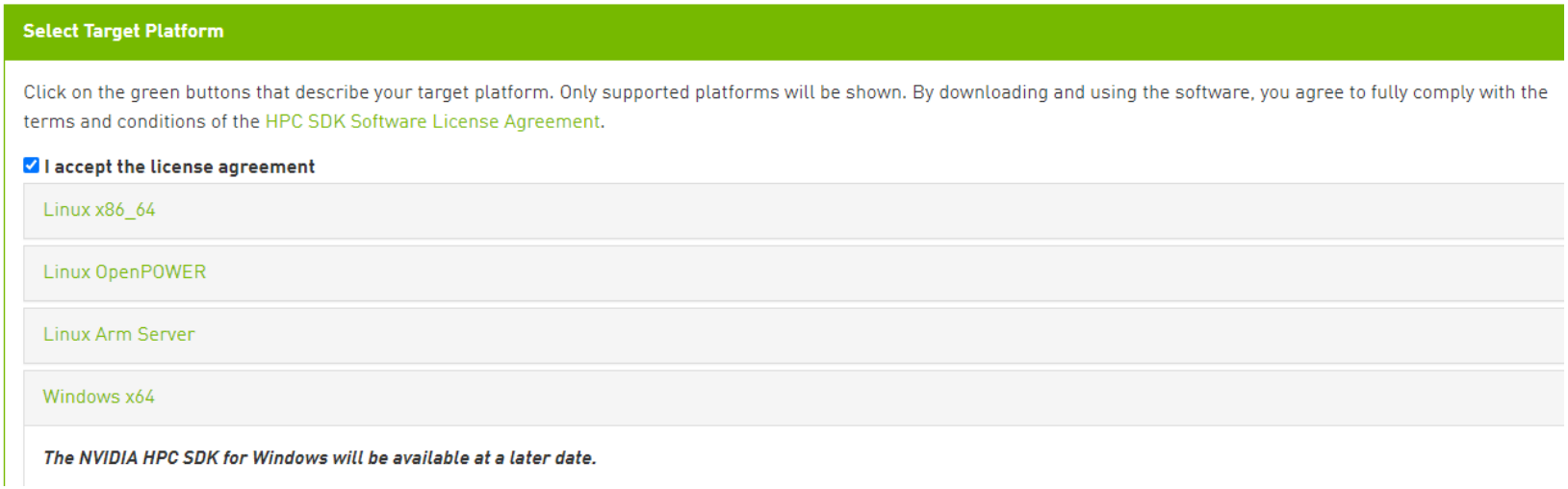


Kirk, D. B., & Wen-Mei, W. H. (2016). *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann.

INSTALACIÓN

https://developer.nvidia.com/hpc-sdk

NVIDIA HPC SDK Version 20.11 Downloads

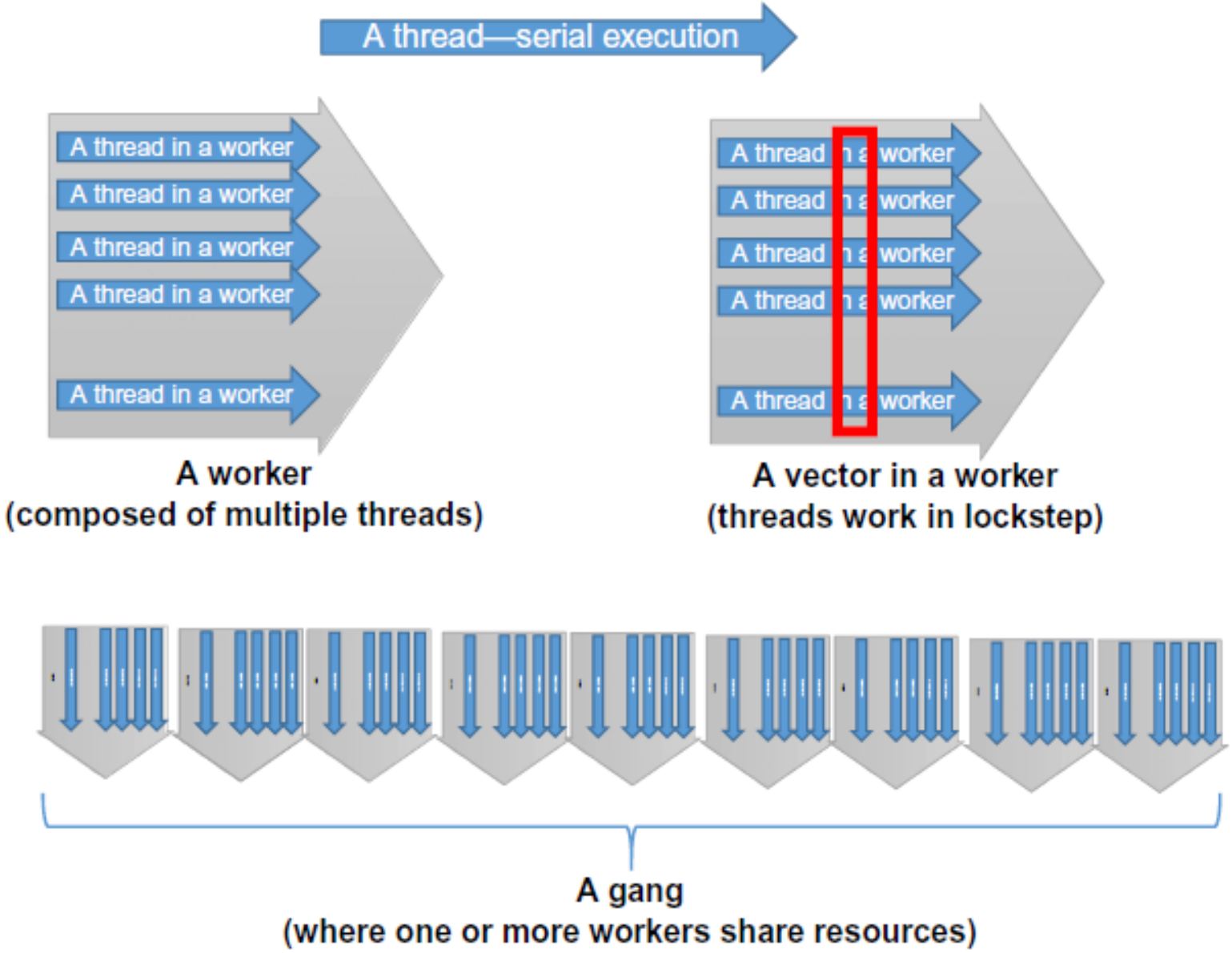


Documentación del HPC SDK:
https://docs.nvidia.com/hpc-sdk/compilers/openacc-gs/index.html

FORMAS DE PARALELISMO EN OPENACC

- **Thread**: Este es un simple hilo corriendo o ejecutándose en algún código C/C++ o Fortran
- **Worker**: Grupos de hilos que pueden operar juntos en un SIMD o vector. En CUDA, esto sería un *warp*
- **Vectors**: Hacen que los grupos de hilos (*workers*) trabajen al mismo tiempo cuando se ejecuta una instrucción vectorial o SIMD. En CUDA, estos serían hilos dentro de un warp
- **Gang**: Son grupos de *workers (pandillas)* los cuales operan de forma independiente uno del otro. En CUDA, una pandilla, sería un bloque de hilos

Farber, R. (2016). *Parallel programming with OpenACC*. Newnes.



Farber, R. (2016). *Parallel programming with OpenACC*. Newnes.

SINTAXIS

- **C/C++**

#pragma acc <directiva> [clause[[,] clause] …] new line

- **Fortran**

!#acc <directive> [clause[[,] clause] …]

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVAS

- **Cómputo**: Para marcar un bloque de código que es posible paralelizar y distribuir el trabajo entre los múltiples hilos

  kernels, parallel, loop, routine

- **Gestión de datos**: Para especificar un determinado tipo de tratamiento de los datos. Evitar el movimiento innecesario de los datos entre diferentes ubicaciones de memoria

  data, update, cache, atomic, declare, enter data, exit data

- **Sincronización**: Para esperar una o varias tareas simultáneas, se utiliza la directiva

  wait

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

CLÁUSULAS

- **Manejo de datos:** Asignan un determinado comportamiento a las variables que se especifican en cada cláusula

  default, private, firstprivate, copy, copyin, copyout, create, delete, deviceptr

- **Distribución de trabajo:** Para distribuir el trabajo entre los hilos generados

  seq, auto, gang, worker, vector, tile, num_gangs, num _workers, vector_length.

- **Flujo de control:** Para la dirección de la ejecución paralela durante el tiempo de ejecución del programa

  if, if_present, independent, reduction, async, wait

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

INSTRUCCIONES DE LA API Y VARIABLES DE AMBIENTE

- OpenACC también ofrece un control más detallado y de bajo nivel para la ejecución del programa
- Hay una API que puede utilizarse dentro del programa incluyendo el archivo de cabecera:
  - **C/C++:** #include "openacc.h"
  - **Fortran:** use openacc
- El uso de las rutinas de la API, nos llevan a una dependencia en un entorno de ejecución con OpenACC
- Podemos usarlas para:
  - Consultar y configurar los tipos y número de dispositivos encontrados para realizar el comp. paralelo. También se pueden usar la variables de entorno: ACC_DEVICE_TYPE, ACC_DEVICE_NUM.
  - Inicialización y apagado del tiempo de ejecución de OpenACC
  - Prueba y espera para trabajos lanzados de forma asíncrona
  - Asignar, mapear y liberar memoria manualmente en el dispositivo de cómputo y transferencias de datos manuales.

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVA KERNELS

- Especifica una región de código que el compilador debe analizar y decide qué paralelizar y cómo distribuirlo en el dispositivo de cómputo
- El compilador podría ejecutar esa región de código de forma secuencial o en paralelo
- La respons. para paralelizar el código es del compilador

```
int a[n][m], b[n][m], c[n][m];
init(a,b,n,m);
#pragma acc kernels
{
  for(int j = 0; j < n; ++j) {
    for(int k = 0; k < m; ++k) {
      c[j][k] = a[j][k];
      a[j][k] = c[j][k] + b[j][k];
    }
  }
  for(int j = 0; j < n; ++j) {
    for(int k = 0; k < m; ++k) {
      d[j][k] = a[j][k] - 5;
    }
  }
}
```

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

- Del ejemplo anterior, el compilador podría realizar al menos dos cosas:

```
int a[n][m], b[n][m], c[n][m];
init(a,b,n,m);
#pragma acc kernels
for(int j = 0; j < n; ++j) {
  for(int k = 0; k < m; ++k) {
    c[j][k] = a[j][k];
    a[j][k] = c[j][k] + b[j][k];
    d[j][k] = a[j][k] - 5;
  }
}
```

Fusionar las dos partes de ciclos anidados en una sola parte

```
int a[n][m], b[n][m], c[n][m];
init(a,b,n,m);
#pragma acc parallel loop
for(int j = 0; j < n; ++j) {
  for(int k = 0; k < m; ++k) {
    c[j][k] = a[j][k];
    a[j][k] = c[j][k] + b[j][k];
  }
}
#pragma acc parallel loop
for(int j = 0; j < n; ++j) {
  for(int k = 0; k < m; ++k) {
    d[j][k] = a[j][k] - 5;
  }
}
```

Generar dos kernels que podrían codificarse como regiones paralelas

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVA PARALLEL

- Específica qué código en la región paralela es seguro para paralelizar y ejecutará todo el código dentro de esa región paralela en todos los hilos disponibles en el dispositivo de cómputo
- Aquí la responsabilidad para paralelizar el código es del usuario

```
int a[n][m], b[n][m], c[n][m];
init(a,b,n,m);
#pragma acc parallel
for(int j = 0; j < n; ++j) {
  for(int k = 0; k < m; ++k) {
    c[j][k] = a[j][k];
    a[j][k] = c[j][k] + b[j][k];
  }
}
```

¿Qué está mal en este código?

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVA LOOP

- Se puede utilizar dentro de la directiva kernels o parallel
- Dentro de parallel, le dice al compilador que las iteraciones del ciclo son independientes
- Dentro de kernels, le puede indicar al compilador que el ciclo tiene iteraciones indep., cuando el compilador no pueda determinar esto en tiempo de compilación

```
void foo( int *a, int *b, n ) {
#pragma acc parallel
#pragma acc loop
  for(int j = 0; j < n; ++j) {
    a[j] += b[j];
  }
}
```

```
void bar( int *a; int *b, n) {
#pragma acc parallel
#pragma acc loop gang worker vector
  for(int j = 0; j < n; ++j) {
    a[j] += b[j];
  }
}
```

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVA ROUTINE

- Ya que alguna función podría ser llamada dentro de alguna región paralela con algún paralelismo arbitrario, entonces está debe compilarse también para tal situación

```
#pragma acc routine
extern void foo(int *a, int *b, int n);

#pragma acc routine(foo)

#pragma acc routine
void foo(int *a, int *b, int n) {
#pragma acc loop gang
  for(int j = 0; j < n; ++j) {
     a[j] += b[j];
  }
}
```

Nota: No se puede solicitar ningún nivel de paralelismo más de una vez, especialmente para rutinas y ciclos anidados

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

MANEJO DE LA MEMORIA

- Variables que serán de tipo private:
  - Variables de control de un ciclo asociadas con la directiva loop
  - Variables índices de ciclos do en Fortran contenidas en una región kernels o parallel
  - Variables declaradas dentro de un bloque
  - Variables declaradas dentro de procedimientos
- Variables escalares utilizadas en regiones paralelas pero no listadas en cláusulas serán de tipo firstprivate o private, dependiendo de si la variable se usa primeramente para leer o escribir dentro de la región

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVA DATA PARA GESTIÓN DE DATOS

- Hay dos tipos de directivas de datos:

```
#pragma acc data copy(a)
{
  <use a>
}

!$acc data copy(a)
  <use a>
!$acc end data
```

```
void foo(int *array, int n) {
#pragma acc enter data copyin(array[0:n])
}

void bar(int *array, int n) {
#pragma acc exit data copyout(array[0:n])
}
```

Estructurado: define un simple ámbito léxico para determinar en donde comienza "{" y termina "}" la vida útil de los datos.

No Estructurado: El tiempo de vida de las variables se determina por el flujo del programa (más flexible)

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

CLÁUSULAS PARA MANEJO DE DATOS

- create: Crea el tiempo de vida de los objetos listados. En un dispositivo de memoria no compartida, la memoria para los objetos debe estar disponible durante la región construida
- present: Asegura que los objetos estén disponibles en el dispositivo para que se pueda realizar algún cálculo en el dispositivo. Si ya están en el dispositivo, no se requiere el movimiento de los datos
- copy: Comienza realizando lo que hace present, si los objetos no están presentes, se realiza lo que hace la cláusula create. Ya que los objetos están en el *device*, entonces se copian los datos desde el *host* al *device* al inicio de la región construida, cuando se termina la región, se realiza una copia del *device* al *host* y se libera la memoria

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

CLÁUSULAS PARA MANEJO DE DATOS (C1)

- copyin: Hace casi todo lo de copy, la diferencia es que al final de la región construida, los datos no se copian del *device* al *host* y estos se liberan solo si el contador de referencia llega a cero
- copyout: Hace casi todo lo de copy, excepto que no realiza la copia del *host* al *device* al inicio de la región construida
- delete: Hace dos cosas. Primero, determina si el objeto está presente, y si no, no toma alguna acción. Segundo, elimina el objeto del entorno de datos del dispositivo forzando que el contador de referencia llegue a cero y liberando entonces la memoria
- deviceptr: Fue agregado para mejorar la compatibilidad con las librerías nativas (como cuFFT) y las llamadas que usan el dispositivo. Esta le dice a OpenACC que el apuntador en la cláusula contiene una dirección que reside en el dispositivo

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

DIRECTIVA CACHE

- Provee un mecanismo para describir los datos que se deben mover a una memoria más rápida, si es posible

```
#pragma acc loop
for(int j = 0; j < m; ++j) {
#pragma acc cache(b[j])
  b[j] = b[j]*c;
}
```

- En este ejemplo, la directiva cache le dice al compilador que cada iteración del ciclo solo va a utilizar un elemento del arreglo b
- En caso de que las iteraciones del ciclo se realicen en paralelo ejecutando N hilos, el compilador debe mover N elementos del arreglo a la memoria rápida. Ya que este movim. tiene un costo, el ciclo debe contener suficientes reutilizaciones de los objetos que se mueven a la memoria rápida

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

TRANSFERENCIAS PARCIALES DE DATOS

- C/C++

#pragma acc data copy(a[0:n])

- Fortran

!#acc data copy(a(1:n))

- Esto se utiliza en dos casos:
  - Cuando al arreglo se le asigna memoria de forma dinámica
  - Cuando el programador solo quiere transferir una parte del arreglo

Chandrasekaran, S., & Juckeland, G. (2017). *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional.

GRACIAS POR SU ATENCIÓN

Francisco J. Hernández-López

fcoj23@cimat.mx

WebPage:

www.cimat.mx/~fcoj23