Universidad Católica San Pablo

CUDA **Zero-Copy**

Marc-Antoine Le Guen

mleguen@ucsp.edu.pe

Zero-Copy Host Memory

- Page-locked memory (pinned memory)
  - Memoria asignada en el host
  - No se puede efectuar un swap hacia el disco duro
  - cudaHostAlloc
    - Flag : cudaHostAllocDefault
  - Mejora hasta de 50% de tiempo de acceso.

Zero-Copy Host Memory

- Zero-copy memory (pinned memory)
  - Memoria asignada en host
  - No se puede efectuar un swap hacia el disco duro
  - cudaHostAlloc
    - Flag : cudaHostAllocMapped
  - Violamos un regla que vimos anteriormente :
    - No podemos acceder a la memoria del CPU desde el GPU

Zero-Copy Host Memory

- En algunos casos puede ser interesante romper la regla de violación
- Sincronizar el acceso a la memoria entre el CPU y el GPU
- Peligro de comportamiento indefinido

Zero-Copy Host Memory - Producto escalar

- `cudaHostAllocMapped` <- Acceder a la memoria del CPU con el GPU
- `cudaHostAllocWriteCombined` <- Creado en el host pero GPU sera el unico en leer los datos (mejorar el rendimiento)

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;

a = (float*)malloc( size*sizeof(float) );
b = (float*)malloc( size*sizeof(float) );
partial_c = (float*)malloc( blocksPerGrid*sizeof(float) );
```

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );
```

Zero-Copy Host Memory - Producto escalar

- Memoria virtual del GPU es diferente a la del CPU
  - Recuperar un puntero para el GPU `cudaHostGetDevicePointer()`
  - Este puntero será enviado al Kernel

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );

for (int i=0; i<size; i++) {
   a[i] = i;
   b[i] = i*2;
}
cudaHostGetDevicePointer( &dev_a, a, 0 );
cudaHostGetDevicePointer( &dev_b, b, 0 );
cudaHostGetDevicePointer( &dev_partial_c, partial_c, 0 );
```

Zero-Copy Host Memory - Producto escalar

- `cudaHostAllocMapped` <- Acceder a la memoria del CPU con el GPU
- `cudaHostAllocWriteCombined` <- GPU sera el unico en acceder a los datos
- Memoria virtual del GPU es diferente a la del CPU
  - Recuperar un puntero para el GPU `cudaHostGetDevicePointer()`
  - Este puntero será pasado al Kernel

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );

for (int i=0; i<size; i++) {
   a[i] = i;
   b[i] = i*2;
}
cudaHostGetDevicePointer( &dev_a, a, 0 );
cudaHostGetDevicePointer( &dev_b, b, 0 );
cudaHostGetDevicePointer( &dev_partial_c, partial_c, 0 );

dot<<<blocksPerGrid,threadsPerBlock>>>( size, dev_a, dev_b,dev_partial_c );
cudaThreadSynchronize();
```

Zero-Copy Host Memory - Producto escalar

- Necesitamos sincronizar el GPU con el CPU para asegurarnos que el GPU terminó su trabajo antes de terminar la reducción en el CPU.

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );

for (int i=0; i<size; i++) {
   a[i] = i;
   b[i] = i*2;
}
cudaHostGetDevicePointer( &dev_a, a, 0 );
cudaHostGetDevicePointer( &dev_b, b, 0 );
cudaHostGetDevicePointer( &dev_partial_c, partial_c, 0 );

dot<<<blocksPerGrid,threadsPerBlock>>>( size, dev_a, dev_b,dev_partial_c );
cudaThreadSynchronize();
```

Zero-Copy Host Memory - Producto escalar

- Se libera la memoria de la misma manera con cudaFreeHost

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );

. . . . .

cudaHostGetDevicePointer( &dev_a, a, 0 );
cudaHostGetDevicePointer( &dev_b, b, 0 );
cudaHostGetDevicePointer( &dev_partial_c, partial_c, 0 );

dot<<<blocksPerGrid,threadsPerBlock>>>( size, dev_a, dev_b,dev_partial_c );
cudaThreadSynchronize();

 . . . . .

cudaFreeHost( a );
cudaFreeHost( b );
cudaFreeHost( partial_c );
```

Zero-Copy Host Memory - Producto escalar

- Saber si puedo usarlo

```
cudaDeviceProp  prop;
int whichDevice;
cudaGetDevice( &whichDevice );
cudaGetDeviceProperties( &prop, whichDevice );
if (prop.canMapHostMemory != 1) {
    printf( "Device can not map memory.\n" );
    return 0;
}
```

Zero-Copy Host Memory - Producto escalar

- Informar que el **device** es autorizado de mapear la memoria del CPU

```
cudaSetDeviceFlags( cudaDeviceMapHost ) );
```

Zero-Copy - Rendimiento

- Mismas ventajas e inconvenientes del page-locked memory
- GPU - Tarjeta gráfica
  - DRAM dedicada
  - Separado de la placa del CPU
- GPU - Integrado
  - Comparte la memoria con el CPU
  - Zero-Copy incremente siempre el rendimiento

Zero-Copy - Rendimiento

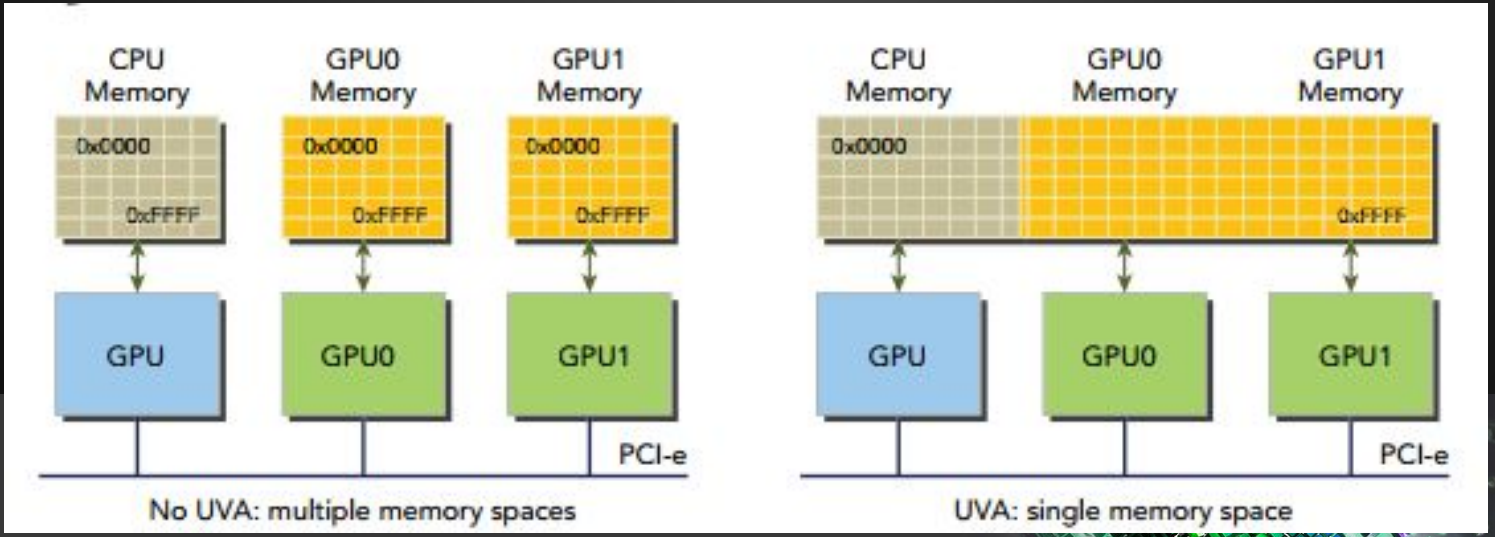
- En caso que los inputs y outputs son usados exactamente una vez veremos una mejora
- Leer y escribir por BUS PCI es acelerado por la tarjeta gráfica y puede representar un mejora de rendimiento
- La memoria no está puesta en el cache en del GPU asi que acceder a los mismos datos varias veces será muy penalizado
  - Mejor seria copiar la memoria al GPU
- Read/Write exactly once
  - 30% - 50%

Zero-Copy - Ventajas

- Utilizar la memoria del host cuando la memoria del device es insuficiente
- Evitar las transferencias explícitas entre CPU y GPU
- Mejorar la transferencia de datos por PCIe

Unified virtual addressing

- CUDA 4.0
- Compute Cap. 2.0
- Linux 64 bits
- Permite simplificar el zero-copy memory
- El puntero de la memoria del host puede ser directamente utilizado por el device
- Inconvenientes y ventajas no cambian



Unified virtual addressing

- El puntero de la memoria del host puede ser directamente utilizado por el

```
float       *a, *b, c, *partial_c;
float       *dev_a, *dev_b, *dev_partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocWriteCombined | cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );

for (int i=0; i<size; i++) {
   a[i] = i;
   b[i] = i*2;
}
cudaHostGetDevicePointer( &dev_a, a, 0 );
cudaHostGetDevicePointer( &dev_b, b, 0 );
cudaHostGetDevicePointer( &dev_partial_c, partial_c, 0 );
```

Unified virtual addressing

- El puntero de la memoria del host puede ser directamente utilizado por el

```
float       *a, *b, c, *partial_c;
// allocate the memory on the CPU
cudaHostAlloc( (void**)&a, size*sizeof(float), cudaHostAllocMapped );
cudaHostAlloc( (void**)&b, size*sizeof(float), cudaHostAllocMapped );
cudaHostAlloc( (void**)&partial_c, blocksPerGrid*sizeof(float), cudaHostAllocMapped );

for (int i=0; i<size; i++) {
   a[i] = i;
   b[i] = i*2;
}

dot<<<blocksPerGrid,threadsPerBlock>>>( size, a, b,partial_c );
```