TALLER DE PROGRAMACIÓN SOBRE GPUS

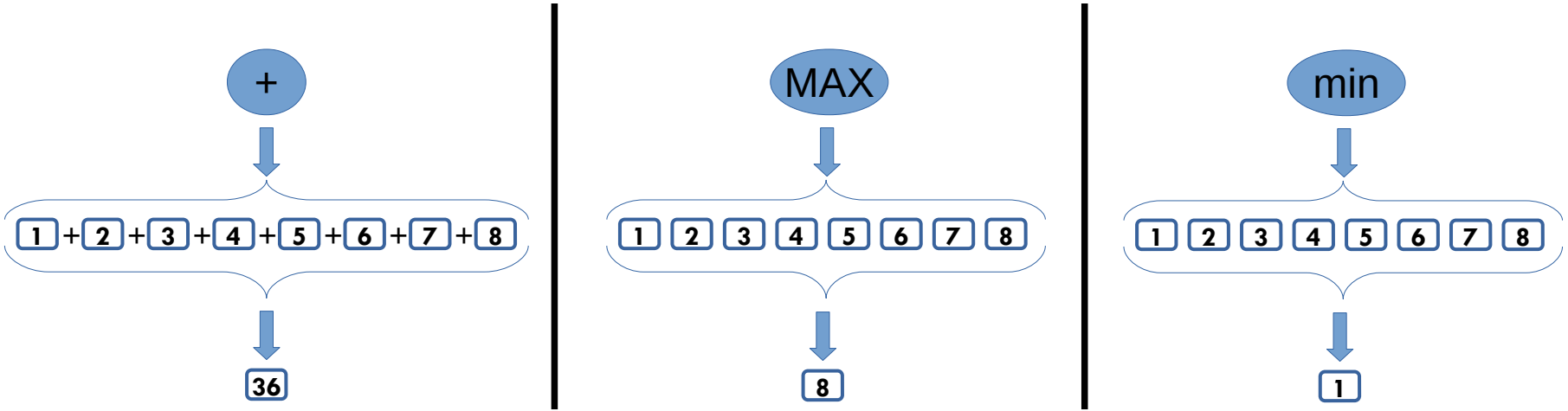
Facultad de Informática – Universidad Nacional de La Plata

Dr. Adrián Pousa

Ejemplo: Optimización de reducción

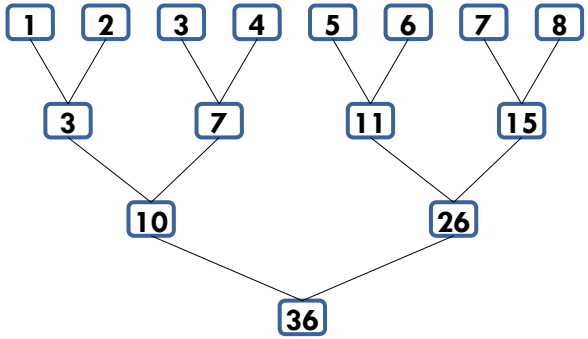
Reducción

- **Reducción:** reducir los valores de un vector a un valor simple.
- Es posible hacerlo aplicando un **operador asociativo** (+, *, MAX, min etc).



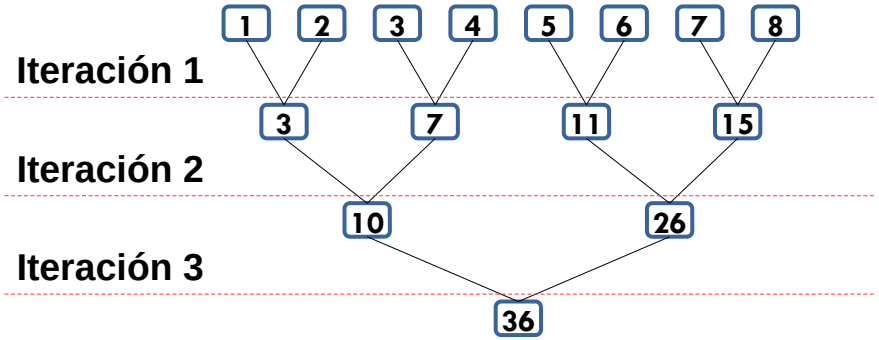
Ejemplo: Suma por Reducción

- En GPU, la solución consiste en una implementación basada en árbol.



Ejemplo: Suma por Reducción

- Suponer que se tiene un vector de 8 elementos.
- La solución requiere varias iteraciones, en cada iteración se realiza una nueva reducción.

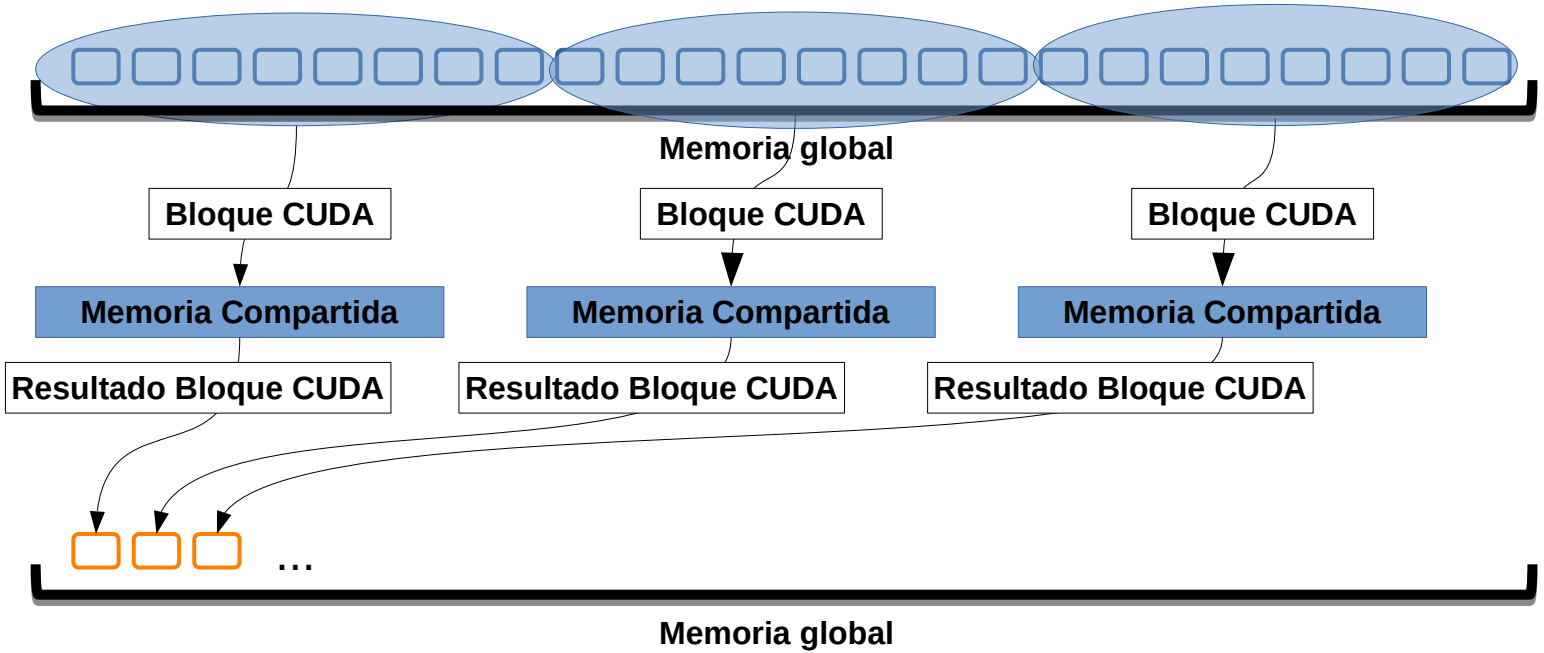


Ejemplo: Suma por Reducción

- Trabajar sobre memoria global resulta costoso.
- Podemos mejorar el tiempo de ejecución utilizando una memoria más rápida como la memoria compartida.
- La estrategia es la siguiente:
  1) Cada bloque trae una porción de los datos, de forma coalescente desde memoria global a la memoria compartida.
  2) Cada bloque realiza la reducción en memoria compartida.
  3) Cada bloque almacena el resultado que reside en memoria compartida en la memoria global.
- Cada bloque dejará un valor. Luego, los resultados de todos los bloques deben reducirse nuevamente.

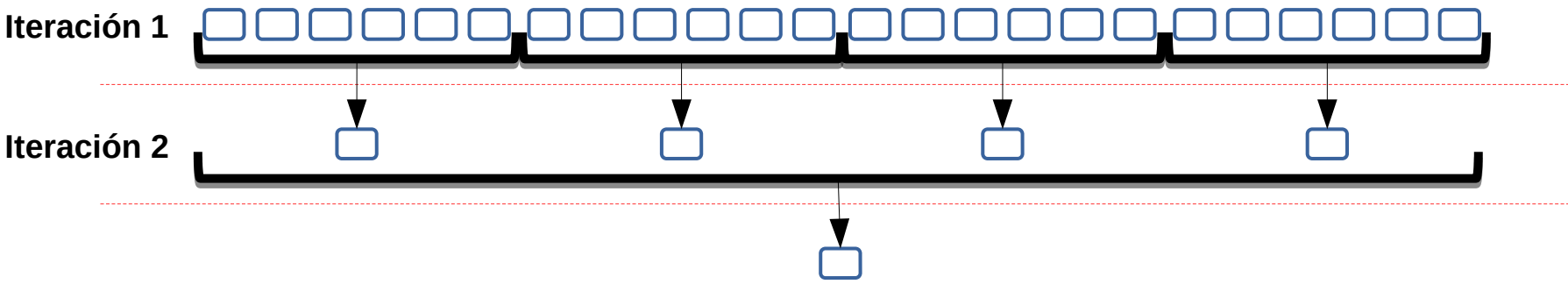
Ejemplo: Suma por Reducción

- Una iteración consiste:



Ejemplo: Suma por Reducción

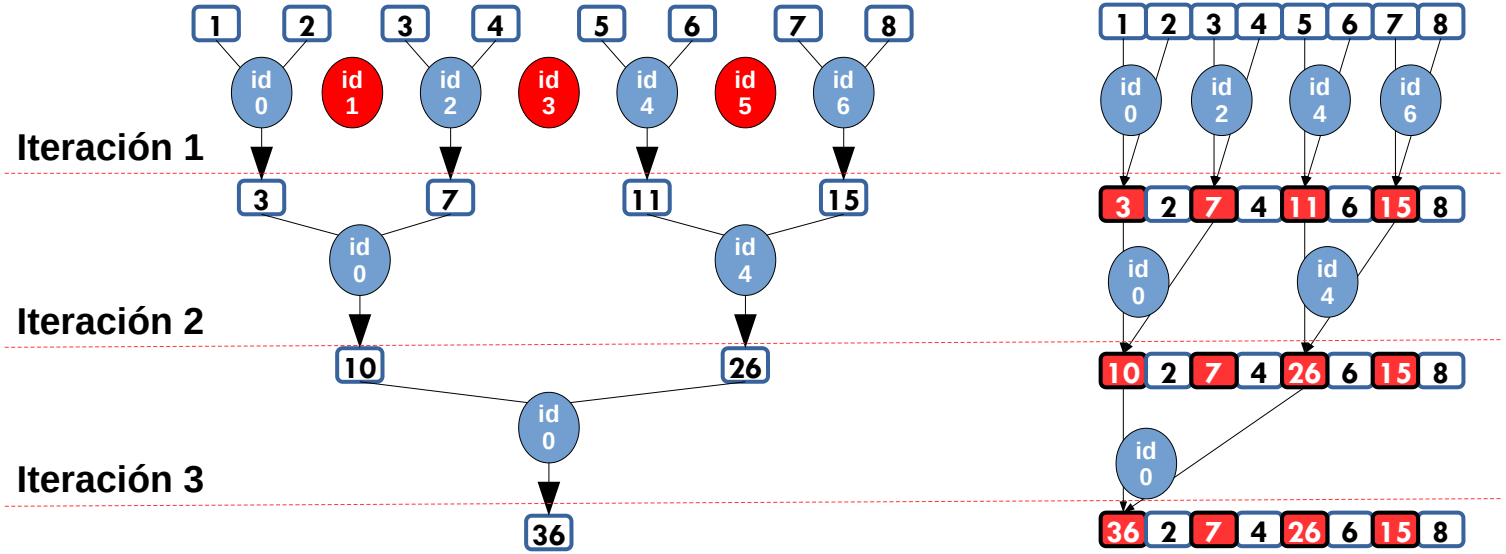
- Las siguientes iteraciones reducen los resultados de los bloques de la iteración anterior.



- A continuación analizaremos distintas soluciones dentro de cada bloque CUDA.

Ejemplo: Suma por Reducción

- **Solución 1:** dentro de cada bloque CUDA los hilos trabajan sobre memoria compartida realizando varia iteraciones. Sólo trabajarán hilos con threadIdx.x a distancia 2*iteración.



Ejemplo: Suma por Reducción


```
__global__ void reduce(int *g_idata, int *g_odata) {
 extern __shared__ int sdata[];
 unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;

      sdata[threadIdx.x] = g_idata[tid];
      __syncthreads();

      for(unsigned int s=1; s < blockDim.x; s *= 2) {
            if (threadIdx.x % (2*s) == 0)
                  sdata[threadIdx.x ] += sdata[threadIdx.x + s];

            __syncthreads();
      }

      if (threadIdx.x == 0)
            g_odata[blockIdx.x] = sdata[0];
}
```

Lectura coalescente desde memoria global a memoria compartida
Cada hilo trae un dato.

Realiza la reducción en memoria compartida

Escritura coalescente del resultado de este bloque desde memoria compartida a memoria global.

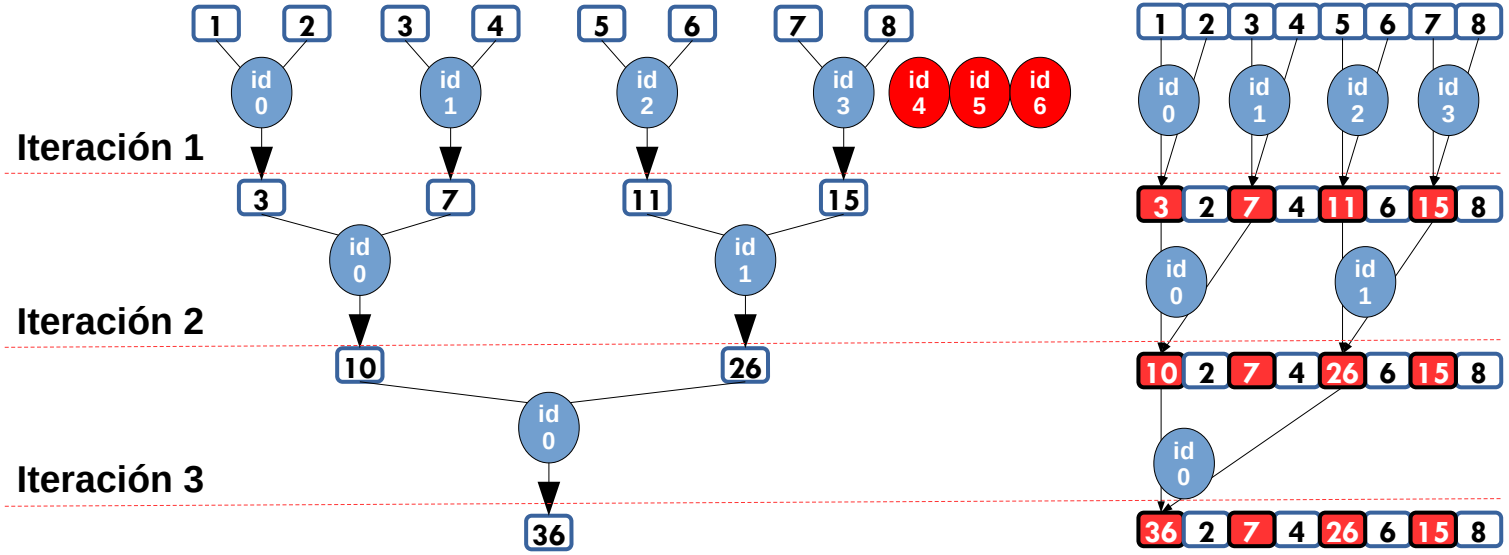
Ejemplo: Suma por Reducción


```
__global__ void reduce(int *g_idata, int *g_odata) {
 extern __shared__ int sdata[];
 unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;

     sdata[threadIdx.x] = g_idata[tid];
     __syncthreads();

     for(unsigned int s=1; s < blockDim.x; s *= 2) {
          if (threadIdx.x % (2*s) == 0)
               sdata[threadIdx.x ] += sdata[threadIdx.x + s];

          __syncthreads();
     }

     if (threadIdx.x == 0)
           g_odata[blockIdx.x] = sdata[0];
}
```

Problema: Divergencia que resulta en un bajo rendimiento. Hilos de un mismo warp no trabajan.

Ejemplo: Suma por Reducción

- **Solución 2:** la misma estrategia reemplazando el if por otro con menor grado de divergencia.



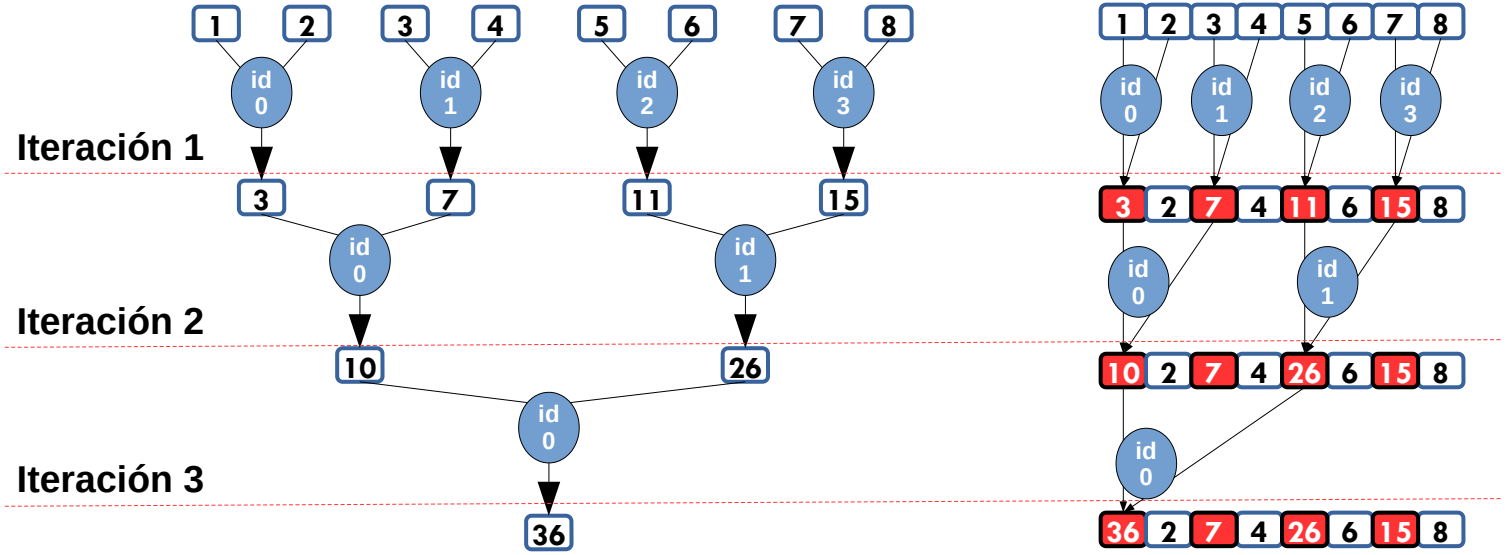
Ejemplo: Suma por Reducción


```
__global__ void reduce(int *g_idata, int *g_odata) {
 extern __shared__ int sdata[];
 unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;

      sdata[threadIdx.x] = g_idata[tid];
      __syncthreads();

      for(unsigned int s=1; s < blockDim.x; s *= 2) {
            int index = 2 * s * threadIdx.x;
            if (index < blockDim.x)
                  sdata[index] += sdata[index + s];

            __syncthreads();
      }

      if (threadIdx.x == 0)
            g_odata[blockIdx.x] = sdata[0];
}
```

Nuevo if con menor grado de divergencia

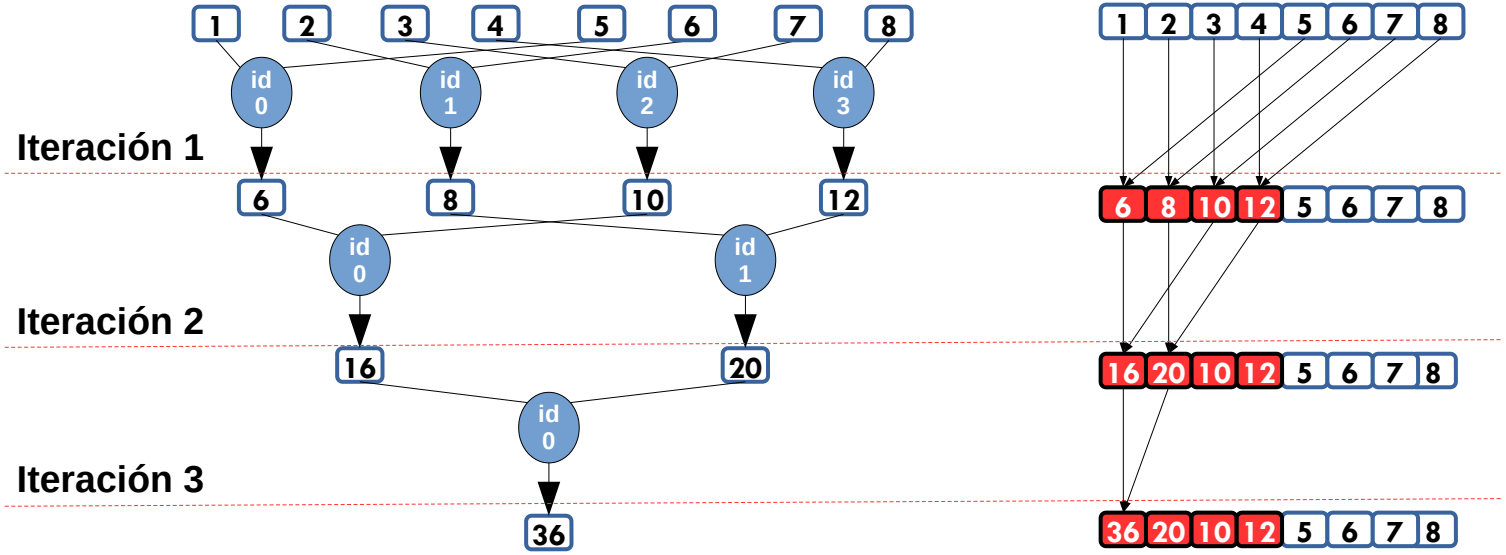
Ejemplo: Suma por Reducción

- **Solución 2:** **tiene conflicto de bancos en memoria compartida!!! si el vector es muy grande a medida que que se itera se accede alejadamente.**



Ejemplo: Suma por Reducción

- **Solución 3:** estrategia con un nuevo patrón de acceso.
- El direccionamiento secuencial está libre de conflictos.



Ejemplo: Suma por Reducción


```
__global__ void reduce(int *g_idata, int *g_odata) {
 extern __shared__ int sdata[];
  unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;

     sdata[threadIdx.x] = g_idata[tid];
     __syncthreads();

     for(unsigned int s=blockDim.x/2; s >0; s >>= 1) {
          if (threadIdx.x < s)
               sdata[threadIdx.x] += sdata[threadIdx.x + s];

          __syncthreads();
     }

     if (threadIdx.x == 0)
           g_odata[blockIdx.x] = sdata[0];
}
```

Nuevo patrón de acceso

Ejemplo: Suma por Reducción


```
__global__ void reduce(int *g_idata, int *g_odata) {
 extern __shared__ int sdata[];
 unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;

     sdata[threadIdx.x] = g_idata[tid];
     __syncthreads();

     for(unsigned int s=blockDim.x/2; s >0; s >>= 1) {
          if (threadIdx.x < s)
               sdata[threadIdx.x] += sdata[threadIdx.x + s];

          __syncthreads();
     }

     if (threadIdx.x == 0)
          g_odata[blockIdx.x] = sdata[0];
}
```

Problema: la mitad de los hilos están inactivos/ociosos en la primera iteración.

Ejemplo: Suma por Reducción

Solución 4:

```
__global__ void reduce(int *g_idata, int *g_odata) {
 extern   shared   int sdata[];
 unsigned int tid = blockIdx.x*(blockDim.x*2) + threadIdx.x;

     sdata[threadIdx.x] = g_idata[tid] + g_idata[tid + blockDim.x];
     __syncthreads();

     for(unsigned int s=blockDim.x/2; s >0; s >>= 1) {
          if (threadIdx.x < s)
               sdata[threadIdx.x] += sdata[threadIdx.x + s];

          __syncthreads();
     }

     if (threadIdx.x == 0)
          g_odata[blockIdx.x] = sdata[0];
}
```

Reducir a la mitad el número de bloques y reemplazar la carga simple con dos cargas de memoria global y la primera suma de la reducción.

Ejemplo: Suma por Reducción

- **Solución 5:**
- La reducción posee una baja intensidad aritmética.
- Un posible cuello de botella es el overhead de instrucciones:
  - Instrucciones auxiliares que no son Loads, Stores o Cómputo sino aritmética de direcciones y overhead del bucle.
- **Estrategia:** desenrollar bucles

Ejemplo: Suma por Reducción

- A medida que avanza la reducción el número de hilos activos disminuye.
- Cuando s <= 32, nos queda sólo un warp.
- Las instrucciones son SIMD sincrónicas dentro de un warp.
- Eso significa que cuando s <= 32:
  - No necesitamos __syncthreads ()
  - No necesitamos "if (tid <s)" porque no se guarda ningún trabajo
- **Estrategia:** Desenrollar las últimas 6 iteraciones del for más interno.

Ejemplo: Suma por Reducción


```
...
     for(unsigned int s=blockDim.x/2; s >32; s >>= 1) {
          if (threadIdx.x < s)
               sdata[threadIdx.x] += sdata[threadIdx.x + s];

          __syncthreads();
     }
     if(threadIdx.x < 32)
          sdata[threadIdx.x] += sdata[threadIdx.x + 32];
          sdata[threadIdx.x] += sdata[threadIdx.x + 16];
          sdata[threadIdx.x] += sdata[threadIdx.x + 8];
          sdata[threadIdx.x] += sdata[threadIdx.x + 4];
          sdata[threadIdx.x] += sdata[threadIdx.x + 2];
          sdata[threadIdx.x] += sdata[threadIdx.x + 1];
     if (threadIdx.x == 0)
           g_odata[blockIdx.x] = sdata[0];
}
```

Cambio en los límites del for y Desenrrollado del último bucle

Esto ahorra trabajo inútil en todos los warps, ¡no solo en el último!
Sin desenrollar, todos los warps ejecutan cada iteración del bucle for y la instrucción if

Ejemplo: Suma por Reducción

- **Solución 6:**
- Si supiéramos el número de iteraciones en tiempo de compilación, podríamos desenrollar por completo la reducción:
  - El número de hilos por bloque generalmente lo conocemos
  - Además, limitamos el número de hilos por bloque a potencias de 2
- Entonces, podemos desenrollar por completo un bucle fácilmente para un bloque de tamaño fijo, pero tenemos que ser genéricos:

¿Cómo desenrollamos el bloque para tamaños desconocidos en tiempo de compilación?

Ejemplo: Suma por Reducción

- **Podemos hacer uso de CUDA Templates (sólo C++)**
- En el encabezado del kernel se debe indicar el parámetro genérico a recibir como template:

```
...
template <unsigned int blockSize>
__global__ void reduce(int *g_idata, int *g_odata){
…
}
```

- En la llamada al kernel se especifica el tamaño de bloque como un parámetro de template:

```
int main(int argc, char* argv[] ){
...
 reduce<512><<<dimGrid, dimBlock, smemSize >>>(d_idata, d_odata);
…
}
```

Ejemplo: Suma por Reducción


```
template <unsigned int blockSize>
__global__ void reduce(int *g_idata, int *g_odata){
…
if (blockSize >= 512) {    if (tid < 256) { sdata[threadIdx.x] += sdata[threadIdx.x + 256]; }
__syncthreads();           }
if (blockSize >= 256) {    if (tid < 128) { sdata[threadIdx.x] += sdata[threadIdx.x + 128]; }
__syncthreads();      }
if (blockSize >= 128) {    if (tid < 64) { sdata[threadIdx.x] += sdata[threadIdx.x + 64]; }
__syncthreads();      }
if (tid < 32) {
      if (blockSize >= 64) sdata[threadIdx.x] += sdata[threadIdx.x + 32];
      if (blockSize >= 32) sdata[threadIdx.x] += sdata[threadIdx.x + 16];
      if (blockSize >= 16) sdata[threadIdx.x] += sdata[threadIdx.x + 8];
      if (blockSize >= 8) sdata[threadIdx.x] += sdata[threadIdx.x + 4];
      if (blockSize >= 4) sdata[threadIdx.x] += sdata[threadIdx.x + 2];
      if (blockSize >= 2) sdata[threadIdx.x] += sdata[threadIdx.x + 1];
 }
...
}
```

El código relacionado a blockSize se evalúa en tiempo de compilación

Ejemplo: Suma por Reducción

La llamada al kernel debe especificarse para cada opción, los templates no aceptan variables:

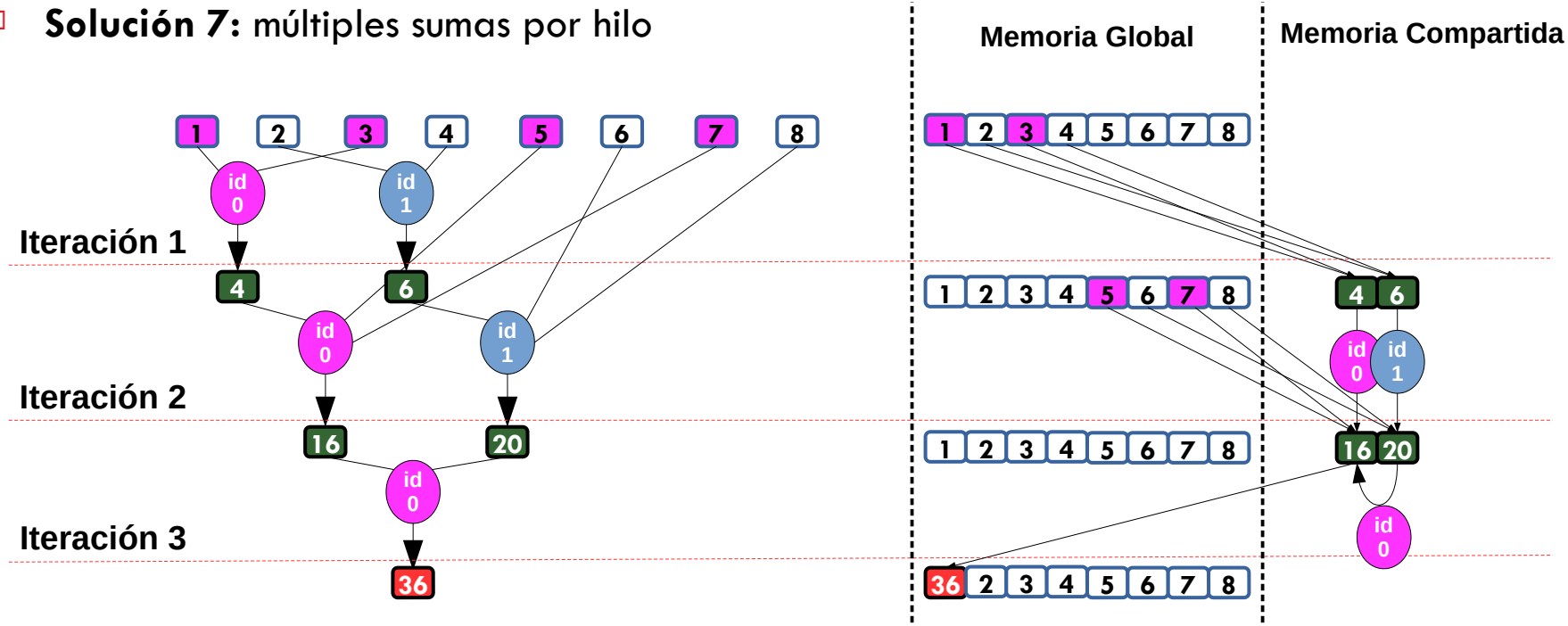
```
int main(int argc, char* argv[]){
...
switch (threads){
 case 512: reduce<512><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 256: reduce<256><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 128: reduce<128><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 64: reduce<64><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 32: reduce<32><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 16: reduce<16><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 8: reduce< 8><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 4: reduce<4><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 2: reduce<2><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
 case 1: reduce<1><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
...
}
```

Ejemplo: Suma por Reducción

- **Solución 7:**
- La idea es combinar una ejecución secuencial con una paralela donde cada hilo carga y suma múltiples elementos en memoria compartida.
- Cada hilo realiza una iteración que ejecuta tantas sumas como sea necesario.
- Cada hilo suma secuencialmente una porción de la memoria compartida.

Ejemplo: Suma por Reducción

- **Solución 7:** múltiples sumas por hilo



Ejemplo: Suma por Reducción


```
__global__ void reduce(int *g_idata, int *g_odata, unsigned int n) {
 extern __shared__ int sdata[];
 unsigned int tid = blockIdx.x*(blockSize*2) + threadIdx.x;
 unsigned int gridSize = blockSize*2*gridDim.x;
 sdata[threadIdx.x] =0;

     while(tid<n){
          sdata[threadIdx.x] += g_idata[tid] + g_idata[tid+blockSize];
          i += gridSize;
     }
     __syncthreads();
...

}
```

El loop while se mueve de a gridSize manteniendo la coalescencia.

Ejemplo: Suma por Reducción

Algoritmo Final

```
template <unsigned int blockSize>
__global__ void reduce(int *g_idata, int *g_odata, unsigned int n){
 extern __shared__ int sdata[];
 unsigned int i = blockIdx.x*(blockSize*2) + threadIdx.x;
 unsigned int gridSize = blockSize*2*gridDim.x;
   sdata[threadIdx.x] = 0;
   while (i < n) { sdata[threadIdx.x] += g_idata[i] + g_idata[i+blockSize]; i += gridSize; }
   __syncthreads();
   if (blockSize >= 512) { if (threadIdx.x<256) { sdata[threadIdx.x]+=sdata[threadIdx.x + 256]; } __syncthreads(); }
   if (blockSize >= 256) { if (threadIdx.x<128) { sdata[threadIdx.x]+=sdata[threadIdx.x + 128]; } __syncthreads(); }
   if (blockSize >= 128) { if (threadIdx.x<64) { sdata[threadIdx.x]+=sdata[threadIdx.x + 64]; } __syncthreads(); }
   if (threadIdx.x < 32) {
      if (blockSize >= 64) sdata[threadIdx.x] += sdata[threadIdx.x + 32];
      if (blockSize >= 32) sdata[threadIdx.x] += sdata[threadIdx.x + 16];
      if (blockSize >= 16) sdata[threadIdx.x] += sdata[threadIdx.x + 8];
      if (blockSize >= 8) sdata[threadIdx.x] += sdata[threadIdx.x + 4];
      if (blockSize >= 4) sdata[threadIdx.x] += sdata[threadIdx.x + 2];
      if (blockSize >= 2) sdata[threadIdx.x] += sdata[threadIdx.x + 1];
   }
   if (threadIdx.x == 0) g_odata[blockIdx.x] = sdata[0];
}
```