CUDA **Atomics**

Marc-Antoine Le Guen

mleguen@ucsp.edu.pe

07/07/2017

Atomics

- Algunas situaciones son muy simple en single-thread
  - pero pueden representar un desafío muy grande implementarlo en un programa masivamente paralelo.

Atomics - Objetivos

- Entender que son las atomics operations y para qué son útiles
- Operaciones aritméticas con operaciones atómicas

Atomics operations

- read-modify-write operation
  - x++
  - Quisiéramos :

Table 9.2 Two threads incrementing the value in x

STEP	EXAMPLE
1. Thread A reads the value in x.	`A` reads 7 from x.
2. Thread A adds 1 to the value it read.	`A` computes 8.
3. Thread A writes the result back to x.	`x <- 8.`
4. Thread B reads the value in x.	`B` reads 8 from x.
5. Thread B adds 1 to the value it read.	`B` computes 9.
6. Thread B writes the result back to x.	`x <- 9.`

Atomics operations

- read-modify-write operation
  - x++
  - Podemos obtener :

Table 9.3 Two threads incrementing the value in x with interleaved operations

STEP	EXAMPLE
Thread A reads the value in x.	A reads 7 from x.
Thread B reads the value in x.	B reads 7 from x.
Thread A adds 1 to the value it read.	A computes 8.
Thread B adds 1 to the value it read.	B computes 8.
Thread A writes the result back to x.	`x <- 8.`
Thread B writes the result back to x.	`x <- 8.`

Atomics operations

- Las operaciones que no descomponen el proceso de read-modify-write son atomics operacións.

Atomics - Ejemplo : Histograma

- Contar la frecuencia de cada dato en una serie de datos
  - colores de píxeles en una imagen
  - letras en un texto

2	2	1	2	1	2	2	1	1	1	2	1	1	1
A	C	D	G	H	I	M	N	O	P	R	T	U	W

Figure 9.1 Letter frequency histogram built from the string *Programming with CUDA C*

Atomics - Ejemplo : Histograma

- Muy simple en CPU
  - Inicialización de datos 100 MB
    - 256 - 8 bits
    - histo 256 inicializado a 0

```
#define SIZE    (100*1024*1024)

int main( void ) {
    unsigned char *buffer = (unsigned char*)big_random_block( SIZE );

    unsigned int    histo[256];
    for (int i=0; i<256; i++)
        histo[i] = 0;
```

Atomics - Ejemplo : Histograma

- Muy simple en CPU
  - generar el histograma
  - 0.31 segundos CPU

```
for (int i=0; i<SIZE; i++)
      histo[buffer[i]]++;
```

- Suma del histograma = cantidad de datos

```
long histoCount = 0;
for (int i=0; i<256; i++) {
    histoCount += histo[i];
}
```

Atomics - Ejemplo : Histograma

- GPU
- Inicialización clásica
  - cudaMemset

```
int main( void ) {
      unsigned char *buffer =(unsigned char*)big_random_block( SIZE );


     // allocate memory on the GPU for the file's data
     unsigned char *dev_buffer;
     unsigned int *dev_histo;
     cudaMalloc( (void**)&dev_buffer, SIZE );
     cudaMemcpy( dev_buffer, buffer, SIZE, cudaMemcpyHostToDevice );

     cudaMalloc( (void**)&dev_histo,256 * sizeof( int ) );
     cudaMemset( dev_histo, 0, 256 * sizeof( int ) );
```

Atomics - Ejemplo : Histograma

- GPU
- Inicialización clásica
  - cudaMemset

```
int main( void ) {
      unsigned char *buffer =(unsigned char*)big_random_block( SIZE );


      // allocate memory on the GPU for the file's data
      unsigned char *dev_buffer;
      unsigned int *dev_histo;
      cudaMalloc( (void**)&dev_buffer, SIZE );
      cudaMemcpy( dev_buffer, buffer, SIZE, cudaMemcpyHostToDevice );
 atomicAdd
```

Atomics - Ejemplo : Histograma

- GPU
- Inicialización clásica
  - cudaMemset

```
int main( void ) {
      unsigned char *buffer =(unsigned char*)big_random_block( SIZE );


      // allocate memory on the GPU for the file's data
      unsigned char *dev_buffer;
      unsigned int *dev_histo;
      cudaMalloc( (void**)&dev_buffer, SIZE );
      cudaMemcpy( dev_buffer, buffer, SIZE, cudaMemcpyHostToDevice );

      cudaMalloc( (void**)&dev_histo,256 * sizeof( int ) );
      cudaMemset( dev_histo, 0, 256 * sizeof( int ) );
```

Atomics - Ejemplo : Histograma

- GPU
- Inicialización clásica
  - Asignación de memoria en el Host para después recuperar el histograma resultado

```
unsigned int histo[256];
 ….
cudaMemcpy( histo, dev_histo, 256 * sizeof( int ),
cudaMemcpyDeviceToHost );
```

Atomics - Ejemplo : Histograma

- GPU
- **Verificación de los resultados**
    - Calcular en el CPU un histograma al reverse
        - restar / sumar
        - Si existe una elemento diferente de 0 en el histograma, nuestro programa GPGPU fallo.

```
for (int i=0; i<SIZE; i++)
 histo[buffer[i]]--;
 for (int i=0; i<256; i++) {
     if (histo[i] != 0)
            printf( "Failure at %d!  Off by %d\n", i, histo[i] );
 }
```

Atomics - Ejemplo : Histograma

- Kernel
  - atomicAdd(addr , value)
    - el gpu nos asegura la conservación de la unidad del proceso read-modify-write

```
__global__ void histo_kernel( unsigned char *buffer,
                              long size,
                              unsigned int *histo ) {
     // calculate the starting index and the offset to
the next
     // block that each thread will be processing
     int i = threadIdx.x + blockIdx.x * blockDim.x;
     int stride = blockDim.x * gridDim.x;
     while (i < size) {
        atomicAdd( &histo[buffer[i]], 1 );
        i += stride;
     }
}
```

Atomics - Ejemplo : Histograma

- Problema, este algoritmo produce un problema debido a la gran cantidad de threads en competición para acceder a la misma memoria
  - Muchas operaciones serializadas.
  - Verdad en GPU de compute capability ~ 1.3
    - 0.31 segundos CPU
    - 1.7 segundos 285 gtx 1.3 vs 0.035 segundos 670 gtx 3.0

```
__global__ void histo_kernel( unsigned char *buffer,
                              long size,
                              unsigned int *histo ) {
      // calculate the starting index and the offset to the next
      // block that each thread will be processing
      int i = threadIdx.x + blockIdx.x * blockDim.x;
      int stride = blockDim.x * gridDim.x;
      while (i < size) {
         atomicAdd( &histo[buffer[i]], 1 );
         i += stride;
      }
}
```

Atomics - Ejemplo : Histograma

```
int main( void ) {
        unsigned char *buffer =
                        (unsigned char*)big_random_block ( SIZE );

        // capture the start time
        // starting the timer here so that we include the cost of
        // all of the operations on the GPU.
        cudaEvent_t    start, stop;
        cudaEventCreate ( &start );
        cudaEventCreate ( &stop );
        cudaEventRecord ( start, 0 );
        // allocate memory on the GPU for the file's data
        unsigned char *dev_buffer;
        unsigned int *dev_histo;
        cudaMalloc ( (void**)&dev_buffer, SIZE );
        cudaMemcpy ( dev_buffer, buffer, SIZE,
                                cudaMemcpyHostToDevice );

        cudaMalloc ( (void**)&dev_histo,
                                256 * sizeof( int ) );
        cudaMemset ( dev_histo, 0,
                                 256 * sizeof( int ) );

        // kernel launch - 2x the number of mps gave best timing
        cudaDeviceProp  prop;
        cudaGetDeviceProperties ( &prop, 0 );
        int blocks = prop.multiProcessorCount;
        printf(" multiprocesseur :  %d" ,  blocks);
        histo_kernel<<<blocks*2,256>>>( dev_buffer, SIZE, dev_histo
);

        unsigned int   histo[256];
        cudaMemcpy ( histo, dev_histo,
                                256 * sizeof( int ),
                                cudaMemcpyDeviceToHost );
```

```
    // get stop time, and display the timing results
        cudaEventRecord ( stop, 0 );
        cudaEventSynchronize ( stop );
        float   elapsedTime;
        cudaEventElapsedTime ( &elapsedTime,
                                    start, stop );
        printf( "Time to generate:  %3.1f ms\n" , elapsedTime );

        long histoCount = 0;
        for (int i=0; i<256; i++) {
        histoCount += histo[i];
        }
        printf( "Histogram Sum:  %ld\n" , histoCount );

        // verify that we have the same counts via CPU
        for (int i=0; i<SIZE; i++)
        histo[buffer[i]]--;
        for (int i=0; i<256; i++) {
        if (histo[i] != 0)
            printf ( "Failure at %d!  Off by %d\n" , i, histo[i] );
        }

        cudaEventDestroy ( start );
        cudaEventDestroy ( stop );
        cudaFree( dev_histo );
        cudaFree( dev_buffer );
        free( buffer );
        system("PAUSE");
        return 0;
}
```

Atomics - Ejemplo : Histograma

```
int main( void ) {
        unsigned char *buffer =
                        (unsigned char*)big_random_block ( SIZE );

        // capture the start time
        // starting the timer here so that we include the cost of
        // all of the operations on the GPU.
        cudaEvent_t    start, stop;
        cudaEventCreate ( &start );
        cudaEventCreate ( &stop );
        cudaEventRecord ( start, 0 );
        // allocate memory on the GPU for the file's data
        unsigned char *dev_buffer;
        unsigned int *dev_histo;
        cudaMalloc( (void**)&dev_buffer, SIZE );
        cudaMemcpy( dev_buffer, buffer, SIZE,
                                cudaMemcpyHostToDevice );

        cudaMalloc( (void**)&dev_histo,
                                256 * sizeof( int ) );
        cudaMemset( dev_histo, 0,
                                 256 * sizeof( int ) );

        // kernel launch - 2x the number of mps gave best timing
        cudaDeviceProp  prop;
        cudaGetDeviceProperties ( &prop, 0 );
        int blocks = prop.multiProcessorCount;
        printf(" multiprocesseur :  %d" ,  blocks);
        histo_kernel<<<blocks*2,256>>>( dev_buffer, SIZE, dev_histo
);

        unsigned int   histo[256];
        cudaMemcpy( histo, dev_histo,
                                256 * sizeof( int ),
                                cudaMemcpyDeviceToHost  );
```

```
    // get stop time, and display the timing results
        cudaEventRecord ( stop, 0 );
        cudaEventSynchronize ( stop );
        float   elapsedTime;
        cudaEventElapsedTime ( &elapsedTime,
                                       start, stop );
        printf( "Time to generate:  %3.1f ms\n" , elapsedTime );

        long histoCount = 0;
        for (int i=0; i<256; i++) {
        histoCount += histo[i];
        }
        printf( "Histogram Sum:  %ld\n" , histoCount );

        // verify that we have the same counts via CPU
        for (int i=0; i<SIZE; i++)
        histo[buffer[i]]--;
        for (int i=0; i<256; i++) {
        if (histo[i] != 0)
            printf ( "Failure at %d!  Off by %d\n" , i, histo[i] );
        }

        cudaEventDestroy ( start );
        cudaEventDestroy ( stop );
        cudaFree( dev_histo );
        cudaFree( dev_buffer );
        free( buffer );
        system("PAUSE");
        return 0;
}
```

Atomics - Ejemplo : Histograma

```
int main( void ) {
        unsigned char *buffer =
                        (unsigned char*)big_random_block( SIZE );

        // capture the start time
        // starting the timer here so that we include the cost of
        // all of the operations on the GPU.
        cudaEvent_t     start, stop;
        cudaEventCreate( &start );
        cudaEventCreate( &stop );
        cudaEventRecord( start, 0 );
        // allocate memory on the GPU for the file's data
        unsigned char *dev_buffer;
        unsigned int *dev_histo;
        cudaMalloc( (void**)&dev_buffer, SIZE );
        cudaMemcpy( dev_buffer, buffer, SIZE,
                              cudaMemcpyHostToDevice );

        cudaMalloc( (void**)&dev_histo,
                              256 * sizeof( int ) );
        cudaMemset( dev_histo, 0,
                               256 * sizeof( int ) );

        // kernel launch - 2x the number of mps gave best timing
        cudaDeviceProp  prop;
        cudaGetDeviceProperties( &prop, 0 );
        int blocks = prop.multiProcessorCount;
        histo_kernel<<<blocks*2,256>>>( dev_buffer, SIZE, dev_histo );

        unsigned int   histo[256];
        cudaMemcpy( histo, dev_histo,
                              256 * sizeof( int ),
                              cudaMemcpyDeviceToHost );
```

Atomics - Ejemplo : Histograma

```
int main( void ) {
        unsigned char *buffer =
                        (unsigned char*)big_random_block ( SIZE );

        // capture the start time
        // starting the timer here so that we include the cost of
        // all of the operations on the GPU.
        cudaEvent_t    start, stop;
        cudaEventCreate ( &start );
        cudaEventCreate ( &stop );
        cudaEventRecord ( start, 0 );
        // allocate memory on the GPU for the file's data
        unsigned char *dev_buffer;
        unsigned int *dev_histo;
        cudaMalloc ( (void**)&dev_buffer, SIZE );
        cudaMemcpy ( dev_buffer, buffer, SIZE,
                                cudaMemcpyHostToDevice  );

        cudaMalloc ( (void**)&dev_histo,
                                256 * sizeof( int ) );
        cudaMemset ( dev_histo, 0,
                                 256 * sizeof( int ) );

        // kernel launch - 2x the number of mps gave best timing
        cudaDeviceProp  prop;
        cudaGetDeviceProperties ( &prop, 0 );
        int blocks = prop.multiProcessorCount;
        printf(" multiprocesseur :  %d" ,  blocks);
        histo_kernel<<<blocks*2,256>>>( dev_buffer, SIZE, dev_histo
);

        unsigned int   histo[256];
        cudaMemcpy ( histo, dev_histo,
                                256 * sizeof( int ),
                                cudaMemcpyDeviceToHost  );
```

```
    // get stop time, and display the timing results
        cudaEventRecord ( stop, 0 );
        cudaEventSynchronize ( stop );
        float    elapsedTime;
        cudaEventElapsedTime ( &elapsedTime,
                                        start, stop );
        printf( "Time to generate:  %3.1f ms\n" , elapsedTime );

        long histoCount  = 0;
        for (int i=0; i<256; i++) {
        histoCount  += histo[i];
        }
        printf( "Histogram Sum:  %ld\n" , histoCount );

        // verify that we have the same counts via CPU
        for (int i=0; i<SIZE; i++)
        histo[buffer[i]]--;
        for (int i=0; i<256; i++) {
        if (histo[i]  != 0)
            printf ( "Failure at %d!  Off by %d\n" , i, histo[i] );
        }

        cudaEventDestroy ( start );
        cudaEventDestroy ( stop );
        cudaFree( dev_histo  );
        cudaFree( dev_buffer  );
        free( buffer  );
        system("PAUSE");
        return 0;
}
```

Atomics - Ejemplo : Histograma

```
int main( void ) {
        unsigned char *buffer =
                        (unsigned char*)big_random_block ( SIZE );

        // capture the start time
        // starting the timer here so that we include the cost of
        // all of the operations on the GPU.
        cudaEvent_t     start, stop;
        cudaEventCreate ( &start );
        cudaEventCreate ( &stop );
        cudaEventRecord ( start, 0 );
        // allocate memory on the GPU for the file's data
        unsigned char *dev_buffer;
        unsigned int *dev_histo;
        cudaMalloc ( (void**)&dev_buffer, SIZE );
        cudaMemcpy ( dev_buffer, buffer, SIZE,
                                cudaMemcpyHostToDevice );

        cudaMalloc ( (void**)&dev_histo,
                                256 * sizeof( int ) );
        cudaMemset ( dev_histo, 0,
                                 256 * sizeof( int ) );

        // kernel launch - 2x the number of mps gave best timing
        cudaDeviceProp  prop;
        cudaGetDeviceProperties ( &prop, 0 );
        int blocks = prop.multiProcessorCount;
        printf(" multiprocesseur :  %d" ,  blocks);
        histo_kernel<<<blocks*2,256>>>( dev_buffer, SIZE, dev_histo
);

        unsigned int   histo[256];
        cudaMemcpy ( histo, dev_histo,
                                256 * sizeof( int ),
                                cudaMemcpyDeviceToHost );
```

```
   // get stop time, and display the timing results
        cudaEventRecord ( stop, 0 );
        cudaEventSynchronize ( stop );
        float    elapsedTime;
        cudaEventElapsedTime ( &elapsedTime,
                                        start, stop );
        printf( "Time to generate:  %3.1f ms\n" , elapsedTime );

        long histoCount  = 0;
        for (int i=0; i<256; i++) {
        histoCount  += histo[i];
        }
        printf( "Histogram Sum:  %ld\n" , histoCount );

        // verify that we have the same counts via CPU
        for (int i=0; i<SIZE; i++)
        histo[buffer[i]]--;
        for (int i=0; i<256; i++) {
        if (histo[i] != 0)
             printf ( "Failure at %d!  Off by %d\n" , i, histo[i] );
        }

        cudaEventDestroy ( start );
        cudaEventDestroy ( stop );
        cudaFree( dev_histo );
        cudaFree( dev_buffer );
        free( buffer );
        system("PAUSE");
        return 0;
}
```

Atomics - Ejemplo : Histograma

```
int main( void ) {
        unsigned char *buffer =
                        (unsigned char*)big_random_block ( SIZE );

        // capture the start time
        // starting the timer here so that we include the cost of
        // all of the operations on the GPU.
        cudaEvent_t    start, stop;
        cudaEventCreate ( &start );
        cudaEventCreate ( &stop );
        cudaEventRecord ( start, 0 );
        // allocate memory on the GPU for the file's data
        unsigned char *dev_buffer;
        unsigned int *dev_histo;
        cudaMalloc( (void**)&dev_buffer, SIZE );
        cudaMemcpy( dev_buffer, buffer, SIZE,
                                cudaMemcpyHostToDevice );

        cudaMalloc( (void**)&dev_histo,
                                256 * sizeof( int ) );
        cudaMemset( dev_histo, 0,
                                 256 * sizeof( int ) );

        // kernel launch - 2x the number of mps gave best timing
        cudaDeviceProp  prop;
        cudaGetDeviceProperties( &prop, 0 );
        int blocks = prop.multiProcessorCount;
        printf(" multiprocesseur :  %d" ,  blocks);
        histo_kernel<<<blocks*2,256>>>( dev_buffer, SIZE, dev_histo
);

        unsigned int   histo[256];
        cudaMemcpy( histo, dev_histo,
                                256 * sizeof( int ),
                                cudaMemcpyDeviceToHost );
    // get stop time, and display the timing results
        cudaEventRecord( stop, 0 );
        cudaEventSynchronize( stop );
        float   elapsedTime;
        cudaEventElapsedTime( &elapsedTime,
                                          start, stop );
        printf( "Time to generate:  %3.1f ms\n" , elapsedTime );

        long histoCount = 0;
        for (int i=0; i<256; i++) {
        histoCount += histo[i];
        }
        printf( "Histogram Sum:  %ld\n" , histoCount );

        // verify that we have the same counts via CPU
        for (int i=0; i<SIZE; i++)
        histo[buffer[i]]--;
        for (int i=0; i<256; i++) {
        if (histo[i] != 0)
            printf( "Failure at %d!  Off by %d\n" , i, histo[i] );
        }

        cudaEventDestroy( start );
        cudaEventDestroy( stop );
        cudaFree( dev_histo );
        cudaFree( dev_buffer );
        free( buffer );
        system("PAUSE");
        return 0;
}
```

Atomics - shared memory

- Resolución del problema anterior utilizando la shared memory.
- 256 threads/blocks
  - un histograma por blocks

```
__global__ void histo_kernel( unsigned char *buffer, long size, unsigned int *histo ) {

      __shared__  unsigned int temp[256];
      temp[threadIdx.x] = 0;
      __syncthreads();

      . . . . . . . . . .
```

Atomics - shared memory

- 256 threads en competición
  - obtenemos en temp un histograma por block
  - `atomicAdd( &temp[buffer[i]], 1 );`

```
__global__ void histo_kernel( unsigned char *buffer, long size, unsigned int *histo ) {

     __shared__  unsigned int temp[256];
     temp[threadIdx.x] = 0;
     __syncthreads();

     // calculate the starting index and the offset to the next
     // block that each thread will be processing
     int i = threadIdx.x + blockIdx.x * blockDim.x;
     int stride = blockDim.x * gridDim.x;
     while (i < size) {
         atomicAdd( &temp[buffer[i]], 1 );
         i += stride;
     }
      . . . . . .  .
```

Atomics - shared memory

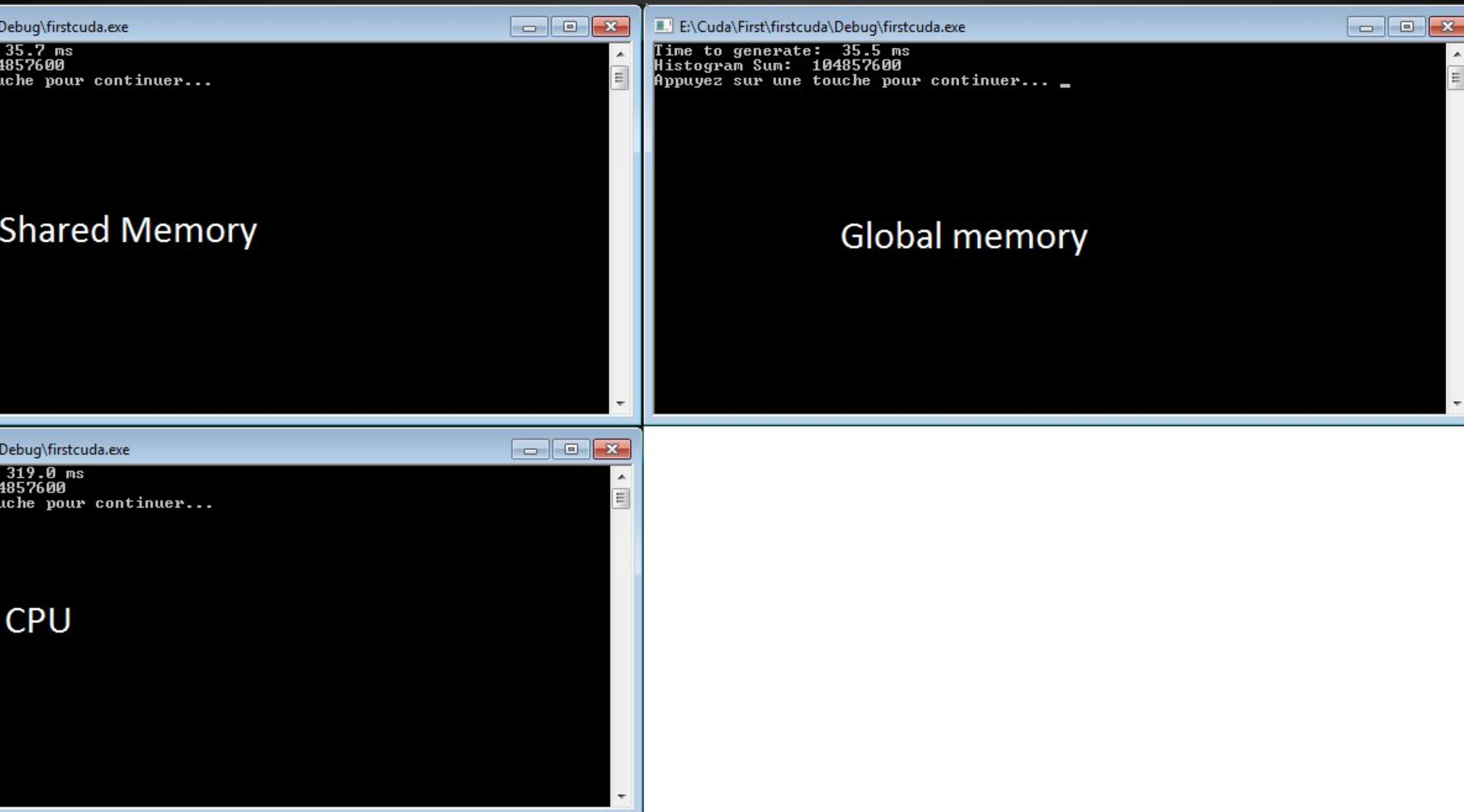
- cada thread tiene que sumar su resultado en histo global
  - `atomicAdd( &(histo[threadIdx.x]), temp[threadIdx.x] );`

```
__global__ void histo_kernel( unsigned char *buffer, long size, unsigned int *histo ) {

      __shared__  unsigned int temp[256];
      temp[threadIdx.x] = 0;
      __syncthreads();

      // calculate the starting index and the offset to the next
      // block that each thread will be processing
      int i = threadIdx.x + blockIdx.x * blockDim.x;
      int stride = blockDim.x * gridDim.x;
      while (i < size) {
      atomicAdd( &temp[buffer[i]], 1 );
      i += stride;
      }
      //updating the
      // global histogram is just one write per thread!
      __syncthreads();
      atomicAdd( &(histo[threadIdx.x]), temp[threadIdx.x] );
}
```

Atomics - Ejemplo : Histograma



Atomics

- Resuelven problemas de competición entre thread para operación de read-modify-write
- Tener cuidado al rendimiento del programa ejecutado en GPU ya que podemos enfrentarnos con problemas de serialización de operaciones

• functions

- atomicAdd()
- atomicSub()
- atomicMin()
- atomicMax()
- ...

http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-functions