Computación paralela en un entorno heterogéneo CPUs-GPUs

Proxecto Fin de Carreira de Enxeñería Técnica en Informática de Sistemas

Iago López Galeiras

30 de septiembre de 2010

Objetivos

- Implementación de algoritmos de la STL

Objetivos

- Implementación de algoritmos de la STL
- Aprovechar sistemas paralelos multiCPU y multiGPU

Objetivos

- Implementación de algoritmos de la STL
- Aprovechar sistemas paralelos multiCPU y multiGPU
- Para ello, fue necesario aprender:

Objetivos

- Implementación de algoritmos de la STL
- Aprovechar sistemas paralelos multiCPU y multiGPU
- Para ello, fue necesario aprender:
  - C++

Objetivos

- Implementación de algoritmos de la STL
- Aprovechar sistemas paralelos multiCPU y multiGPU
- Para ello, fue necesario aprender:
  - C++
  - STL

Objetivos

- Implementación de algoritmos de la STL
- Aprovechar sistemas paralelos multiCPU y multiGPU
- Para ello, fue necesario aprender:
  - C++
  - STL
  - Pthreads

Objetivos

- Implementación de algoritmos de la STL
- Aprovechar sistemas paralelos multiCPU y multiGPU
- Para ello, fue necesario aprender:
  - C++
  - STL
  - Pthreads
  - CUDA

Procesamiento paralelo

- Realizar varios cálculos de forma simultánea

Procesamiento paralelo

- Realizar varios cálculos de forma simultánea
- Límite físico al aumento de frecuencias

Procesamiento paralelo

- Realizar varios cálculos de forma simultánea
- Límite físico al aumento de frecuencias
- Dificultad

Procesamiento paralelo

- Realizar varios cálculos de forma simultánea
- Límite físico al aumento de frecuencias
- Dificultad
- Aumento de rendimiento. En general:

Procesamiento paralelo

- Realizar varios cálculos de forma simultánea
- Límite físico al aumento de frecuencias
- Dificultad
- Aumento de rendimiento. En general:
- Ley de Amdahl

$$S = \frac{1}{(1 - P) + P/N} \tag{1}$$

Clasificación de Flynn

- SISD: Mononúcleo tradicional

Clasificación de Flynn

- SISD: Mononúcleo tradicional
- SIMD: Unidades vectoriales y GPUs

Clasificación de Flynn

- SISD: Mononúcleo tradicional
- SIMD: Unidades vectoriales y GPUs
- MISD: Tolerancia a fallos

Clasificación de Flynn

- SISD: Mononúcleo tradicional
- SIMD: Unidades vectoriales y GPUs
- MISD: Tolerancia a fallos
- MIMD: Multiprocesador

Tipos de paralelismo

- Nivel de bit: N° de bits de cada arquitectura

Tipos de paralelismo

- Nivel de bit: N° de bits de cada arquitectura
- Nivel de instrucción: Pipeline de un procesador

Tipos de paralelismo

- Nivel de bit: Nº de bits de cada arquitectura
- Nivel de instrucción: Pipeline de un procesador
- Nivel de datos: Bucles

Tipos de paralelismo

- Nivel de bit: N° de bits de cada arquitectura
- Nivel de instrucción: Pipeline de un procesador
- Nivel de datos: Bucles
- Nivel de tareas: Multi-hilo

Tipos de hardware paralelo

- Multinúcleo

Tipos de hardware paralelo

- Multinúcleo
- SMP

Tipos de hardware paralelo

- Multinúcleo
- SMP
- Distribuido

Tipos de hardware paralelo

- Multinúcleo
- SMP
- Distribuido
- Arquitecturas especializadas

Lenguajes de programación paralelos

- Complicados y poco robustos

Lenguajes de programación paralelos

- Complicados y poco robustos
- API con funciones utilizables en lenguajes de propósito general

Lenguajes de programación paralelos

- Complicados y poco robustos
- API con funciones utilizables en lenguajes de propósito general
- Ejemplos:

Lenguajes de programación paralelos

- Complicados y poco robustos
- API con funciones utilizables en lenguajes de propósito general
- Ejemplos:
  - MPI

Lenguajes de programación paralelos

- Complicados y poco robustos
- API con funciones utilizables en lenguajes de propósito general
- Ejemplos:
  - MPI
  - OpenMP

Lenguajes de programación paralelos

- Complicados y poco robustos
- API con funciones utilizables en lenguajes de propósito general
- Ejemplos:
  - MPI
  - OpenMP
  - Pthreads

Lenguajes de programación paralelos

- Complicados y poco robustos
- API con funciones utilizables en lenguajes de propósito general
- Ejemplos:
  - MPI
  - OpenMP
  - Pthreads
  - Boost.Thread

GPUs

- Procesador dedicado a gráficos: alto paralelismo

GPUs

- Procesador dedicado a gráficos: alto paralelismo
- Existen problemas con alto paralelismo que no son gráficos: GPGPU

GPUs

- Procesador dedicado a gráficos: alto paralelismo
- Existen problemas con alto paralelismo que no son gráficos: GPGPU
- Al principio lenguajes poco adecuados

GPUs

- Procesador dedicado a gráficos: alto paralelismo
- Existen problemas con alto paralelismo que no son gráficos: GPGPU
- Al principio lenguajes poco adecuados
- Ahora, han mejorado

GPUs

- Procesador dedicado a gráficos: alto paralelismo
- Existen problemas con alto paralelismo que no son gráficos: GPGPU
- Al principio lenguajes poco adecuados
- Ahora, han mejorado
- Ejemplos:

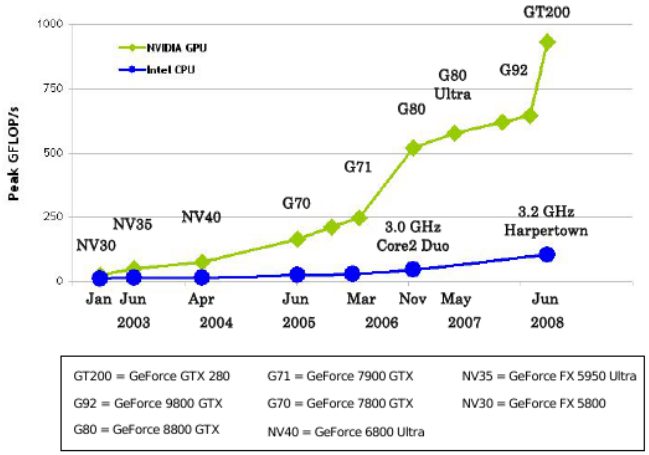
GPUs

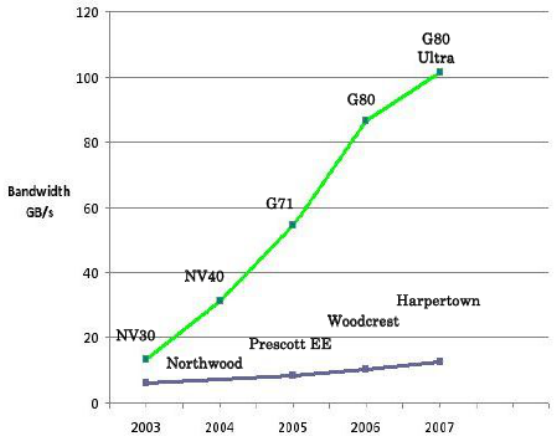
- Procesador dedicado a gráficos: alto paralelismo
- Existen problemas con alto paralelismo que no son gráficos: GPGPU
- Al principio lenguajes poco adecuados
- Ahora, han mejorado
- Ejemplos:
  - OpenCL

GPUs

- Procesador dedicado a gráficos: alto paralelismo
- Existen problemas con alto paralelismo que no son gráficos: GPGPU
- Al principio lenguajes poco adecuados
- Ahora, han mejorado
- Ejemplos:
  - OpenCL
  - CUDA

Evolución CPU vs GPU





CUDA

- Arquitectura de computación paralela en GPU de NVIDIA

CUDA

- Arquitectura de computación paralela en GPU de NVIDIA
- Extensión de C

CUDA

- Arquitectura de computación paralela en GPU de NVIDIA
- Extensión de C
- Abstracciones:

CUDA

- Arquitectura de computación paralela en GPU de NVIDIA
- Extensión de C
- Abstracciones:
  - Jerarquía de hilos (hasta miles)

CUDA

- Arquitectura de computación paralela en GPU de NVIDIA
- Extensión de C
- Abstracciones:
  - Jerarquía de hilos (hasta miles)
  - Memorias compartidas

CUDA

- Arquitectura de computación paralela en GPU de NVIDIA
- Extensión de C
- Abstracciones:
  - Jerarquía de hilos (hasta miles)
  - Memorias compartidas
  - Sincronización por barreras

Modelo de programación CUDA

- Dos APIs:

Modelo de programación CUDA

- Dos APIs:
  - Driver

Modelo de programación CUDA

- Dos APIs:
  - Driver
  - Runtime

Modelo de programación CUDA

- Dos APIs:
  - Driver
  - Runtime
- Ejecución de kernels. Uno por hilo. Mismo código, distintos datos

Modelo de programación CUDA

- Dos APIs:
  - Driver
  - Runtime
- Ejecución de kernels. Uno por hilo. Mismo código, distintos datos
- Jerarquía de hilos:

Modelo de programación CUDA

- Dos APIs:
  - Driver
  - Runtime
- Ejecución de kernels. Uno por hilo. Mismo código, distintos datos
- Jerarquía de hilos:
  - Los hilos se organizan en bloques de 1, 2 o 3 dimensiones

Modelo de programación CUDA

- Dos APIs:
  - Driver
  - Runtime
- Ejecución de kernels. Uno por hilo. Mismo código, distintos datos
- Jerarquía de hilos:
  - Los hilos se organizan en bloques de 1, 2 o 3 dimensiones
  - Los bloques se organizan en rejillas de 1 o 2 dimensiones

Jerarquía de memoria

- Memoria local, región de memoria global privada de cada hilo pero

Jerarquía de memoria

- Memoria local, región de memoria global privada de cada hilo pero
- Memoria compartida, rápida y compartida entre hilos de un bloque

Jerarquía de memoria

- Memoria local, región de memoria global privada de cada hilo pero
- Memoria compartida, rápida y compartida entre hilos de un bloque
- Memoria global, compartida entre todos los hilos

Jerarquía de memoria

- Memoria local, región de memoria global privada de cada hilo pero
- Memoria compartida, rápida y compartida entre hilos de un bloque
- Memoria global, compartida entre todos los hilos
- Memoria constante, cacheada y de solo lectura

Jerarquía de memoria

- Memoria local, región de memoria global privada de cada hilo pero
- Memoria compartida, rápida y compartida entre hilos de un bloque
- Memoria global, compartida entre todos los hilos
- Memoria constante, cacheada y de solo lectura
- Memoria de texturas, cacheada y de solo lectura. Se beneficia de la localidad espacial

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque
- Scalar processors (SPs). Realizan los cálculos de cada hilo.

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque
- Scalar processors (SPs). Realizan los cálculos de cada hilo.
- Memoria por SM:

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque
- Scalar processors (SPs). Realizan los cálculos de cada hilo.
- Memoria por SM:
  - Registros

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque
- Scalar processors (SPs). Realizan los cálculos de cada hilo.
- Memoria por SM:
  - Registros
  - Memoria compartida por los SPs

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque
- Scalar processors (SPs). Realizan los cálculos de cada hilo.
- Memoria por SM:
  - Registros
  - Memoria compartida por los SPs
  - Caches constante y de texturas

Implementación hardware

- Streaming mutiprocessors (SMs) escalables. Se corresponden con un bloque
- Scalar processors (SPs). Realizan los cálculos de cada hilo.
- Memoria por SM:
  - Registros
  - Memoria compartida por los SPs
  - Caches constante y de texturas
- Capacidad computacional: 1.0-1.3

Rendimiento

- Convertir código secuencial en paralelo

Rendimiento

- Convertir código secuencial en paralelo
- Minimizar transferencia RAM -> GPU. Es muy lenta

Rendimiento

- Convertir código secuencial en paralelo
- Minimizar transferencia RAM -> GPU. Es muy lenta
- Memoria compartida

Rendimiento

- Convertir código secuencial en paralelo
- Minimizar transferencia RAM -> GPU. Es muy lenta
- Memoria compartida
- Accesos coalesced ("fusionados"):

Rendimiento

- Convertir código secuencial en paralelo
- Minimizar transferencia RAM -> GPU. Es muy lenta
- Memoria compartida
- Accesos coalesced ("fusionados"):
  - 1.0 y 1.1: Hilos consecutivos -> posiciones de memoria consecutivas

Rendimiento

- Convertir código secuencial en paralelo
- Minimizar transferencia RAM -> GPU. Es muy lenta
- Memoria compartida
- Accesos coalesced ("fusionados"):
  - 1.0 y 1.1: Hilos consecutivos -> posiciones de memoria consecutivas
  - >1.2: Patrón de accesos encaja en determinados segmentos

C++

- Lenguaje de alto nivel, multiplataforma y de uso general

C++

- Lenguaje de alto nivel, multiplataforma y de uso general
- En general superconjunto de C

C++

- Lenguaje de alto nivel, multiplataforma y de uso general
- En general superconjunto de C
- Soporta abstracción de datos, POO y programación genérica

C++

- Lenguaje de alto nivel, multiplataforma y de uso general
- En general superconjunto de C
- Soporta abstracción de datos, POO y programación genérica
- Características interesantes para el proyecto:

C++

- Lenguaje de alto nivel, multiplataforma y de uso general
- En general superconjunto de C
- Soporta abstracción de datos, POO y programación genérica
- Características interesantes para el proyecto:
  - Sobrecarga de operadores: tipos iguales que los propios. Operador()

C++

- Lenguaje de alto nivel, multiplataforma y de uso general
- En general superconjunto de C
- Soporta abstracción de datos, POO y programación genérica
- Características interesantes para el proyecto:
  - Sobrecarga de operadores: tipos iguales que los propios. Operador()
  - Plantillas: programación genérica, funciones con argumentos genéricos. Tiempo de compilación

STL

- Librería (o biblioteca) de C++

STL

- Librería (o biblioteca) de C++
- Genérica mediante plantillas

STL

- Librería (o biblioteca) de C++
- Genérica mediante plantillas
- Estructuras y algoritmos de uso frecuente

Algoritmos utilizados en el proyecto

- Transform: operación específica a cada elemento de un contenedor

Algoritmos utilizados en el proyecto

- Transform: operación específica a cada elemento de un contenedor
- Accumulate: Suma (u otra operación binaria) de los elementos de un contenedor

Definición de las operaciones

- Los algoritmos implementados reciben las operaciones a realizar

Definición de las operaciones

- Los algoritmos implementados reciben las operaciones a realizar
- En el caso de nuestra implementación hay que especificar el algoritmo de CPU y el de GPU

Definición de las operaciones: Transform

- Clase de la que derivar para definir la implementación:

```
1   template <typename T> class Transform_op {
2   protected:
3       const char *_name;
4   public:
5       Transform_op(const char* name):_name(name){}
6       virtual int operator()(T const& n) const = 0;
7       const char* getName()
8       {
9           return _name;
10      }
11  };
```

Definición de las operaciones: Accumulate

- Clase de la que derivar para definir la implementación:

```
1  template <typename T> class Accumulate_op {
2  protected:
3      const char *_name;
4  public:
5      Accumulate_op(const char* name):_name(name){}
6      virtual T operator()(T const& acc, T const& n) const = 0;
7      virtual T identity() const = 0;
8      const char* getName()
9      {
10         return _name;
11     }
12 };
```

Definición de las operaciones: GPU

- Firma de transform:

```
1  extern "C" __global__ void operacion_tipo(TIPO * const vector, const int
       size)
```

Definición de las operaciones: GPU

- Firma de transform:

```
1  extern "C" __global__ void operacion_tipo(TIPO * const vector, const int
       size)
```

- Firma de accumulate:

```
1  extern "C" __global__ void operacion_tipo(const TIPO * const d_in, TIPO *
       const res, const int size)
```

Interfaz de las funciones

- Interfaz de transform:

```
1  template < typename InputIterator, typename OutputIterator, typename T >
2  OutputIterator transform ( InputIterator first1, InputIterator last1,
       OutputIterator result, Transform_op<T> & op );
```

Interfaz de las funciones

- Interfaz de transform:

```
1  template < typename InputIterator, typename OutputIterator, typename T >
2  OutputIterator transform ( InputIterator first1, InputIterator last1,
       OutputIterator result, Transform_op<T> & op );
```

- Interfaz de accumulate con operación por defecto:

```
1  template <class InputIterator, class T>
2      T accumulate(InputIterator first, InputIterator last, T init)
```

Interfaz de las funciones

- Interfaz de transform:

```
1  template < typename InputIterator, typename OutputIterator, typename T >
2  OutputIterator transform ( InputIterator first1, InputIterator last1,
       OutputIterator result, Transform_op<T> & op );
```

- Interfaz de accumulate con operación por defecto:

```
1  template <class InputIterator, class T>
2      T accumulate(InputIterator first, InputIterator last, T init)
```

- Interfaz de accumulate:

```
1  template <class InputIterator, class T>
2      T accumulate(InputIterator first, InputIterator last, T init,
3                   Accumulate_op<T> & binary_op);
```

RAII

- Patrón de diseño para tratar los recursos

RAII

- Patrón de diseño para tratar los recursos
- Recurso <=> Vida del objeto

RAII

- Patrón de diseño para tratar los recursos
- Recurso <=> Vida del objeto
- Evita olvidos de liberar recursos

RAII

- Patrón de diseño para tratar los recursos
- Recurso <=> Vida del objeto
- Evita olvidos de liberar recursos
- Evita problemas de excepciones.

Implementación de los hilos

- Patrón RAII. Hilo <=> Objeto

Implementación de los hilos

- Patrón RAII. Hilo <=> Objeto
- Problema: excepciones. Solución: copia a variable.

Transform

- Reparto de trabajo: parte CPU, parte GPU

Transform

- Reparto de trabajo: parte CPU, parte GPU
- Tamaño de parte GPU ajustado por factor

Transform

- Reparto de trabajo: parte CPU, parte GPU
- Tamaño de parte GPU ajustado por factor
- División a partes iguales

Transform

- Reparto de trabajo: parte CPU, parte GPU
- Tamaño de parte GPU ajustado por factor
- División a partes iguales
- Implementación CPU: bucle que aplica la operación

Transform

- Reparto de trabajo: parte CPU, parte GPU
- Tamaño de parte GPU ajustado por factor
- División a partes iguales
- Implementación CPU: bucle que aplica la operación
- Implementación GPU: vector en memoria global, aplicación de kernel

Accumulate

- Reparto de trabajo: partes iguales entre GPUs

Accumulate

- Reparto de trabajo: partes iguales entre GPUs
- Resultados -> CPU

Accumulate

- Reparto de trabajo: partes iguales entre GPUs
- Resultados -> CPU
- Implementación CPU: bucle que aplica la operación y acumula el resultado

Accumulate

- Reparto de trabajo: partes iguales entre GPUs
- Resultados -> CPU
- Implementación CPU: bucle que aplica la operación y acumula el resultado
- Implementación GPU: Implementación en árbol, cada bloque se encarga de un fragmento y aplicación recursiva

Implementación GPU

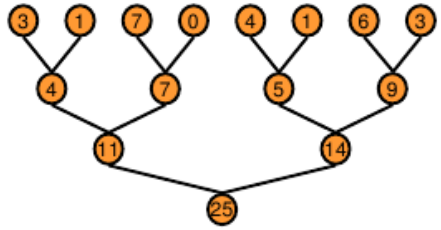


Figura: Árbol de reducción

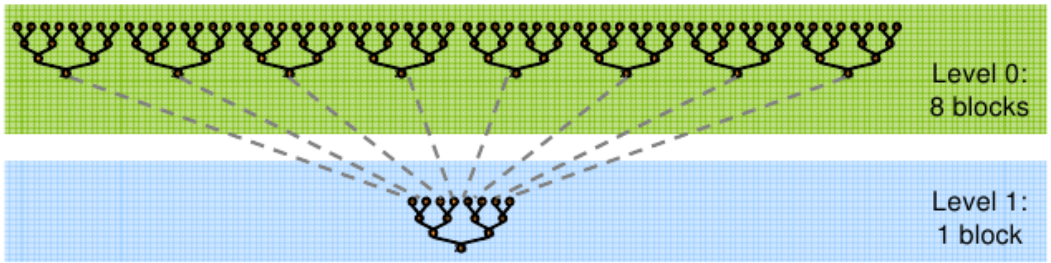


Figura: Aplicación recursiva del kernel

Kernel en un bloque

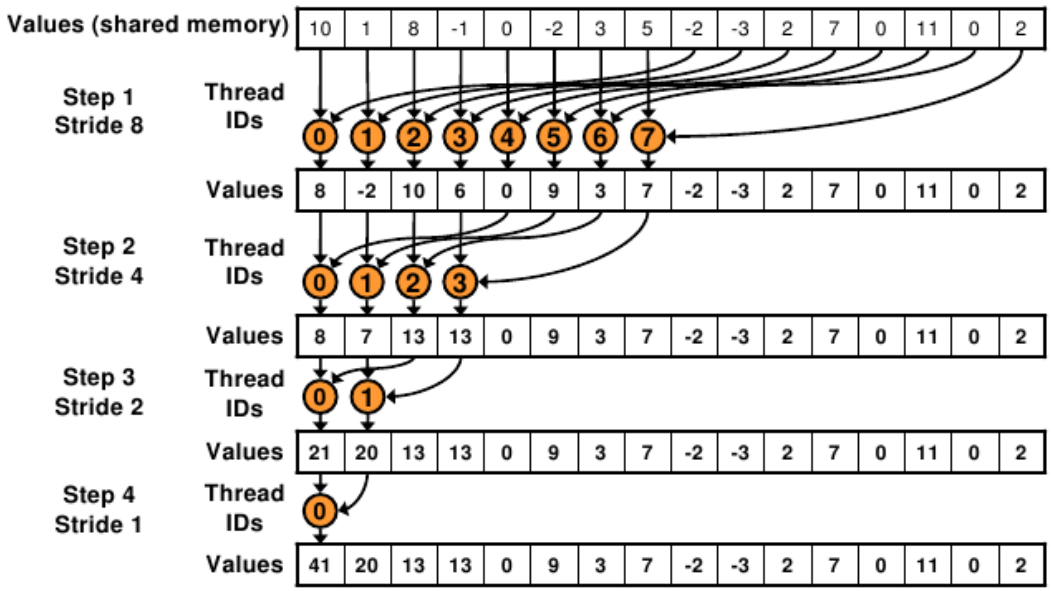


Figura: Direccionamiento secuencial

Configuración de las pruebas

- Hardware:
  - CPU: Intel Core 2 Quad@2.66Ghz
  - RAM: 3GB
  - GPUs: 2 x NVIDIA GeForce 9800GT

Configuración de las pruebas

- Hardware:
  - CPU: Intel Core 2 Quad@2.66Ghz
  - RAM: 3GB
  - GPUs: 2 x NVIDIA GeForce 9800GT
- Software:
  - SO: Debian GNU/Linux con kernel 2.6.32
  - Compilador: gcc 4.3
  - Boost: versión 1.42
  - CUDA: versió 3.0

Configuración de las pruebas

- Hardware:
  - CPU: Intel Core 2 Quad@2.66Ghz
  - RAM: 3GB
  - GPUs: 2 x NVIDIA GeForce 9800GT
- Software:
  - SO: Debian GNU/Linux con kernel 2.6.32
  - Compilador: gcc 4.3
  - Boost: versión 1.42
  - CUDA: versió 3.0
- Algoritmos:
  - Transform: Vector de 450MB de floats y raíz cuadrada
  - Accumulate: Vector de 450MB de enteros y suma

Resultados

Cuadro: Resultados de **transform** (tiempos en ms)

Factor	Cálc. GPU	Transfer. GPU	Total GPU	Total CPU	TOTAL
0	–	–	–	843, 8	844, 3
0,2	2, 54	192, 67	215, 84	749, 33	762, 92
0,4	5, 11	362, 18	405, 06	636, 26	638, 84
0,6	10,39	549, 51	565, 17	475, 15	640, 7
0,8	11, 77	565, 23	580, 5	238, 19	658, 05
1	13, 18	647, 3	659, 14	–	738, 96
					TOTAL
Impl. STL					3758,03

Cuadro: Resultados de **accumulate** (tiempos en ms)

Cálc. por GPU	Transf. por GPU	Total por GPU	Total por CPU	TOTAL
47, 28	321, 37	435, 15	0,002	488, 74
				TOTAL
Impl. STL				1496,96

Aceleración

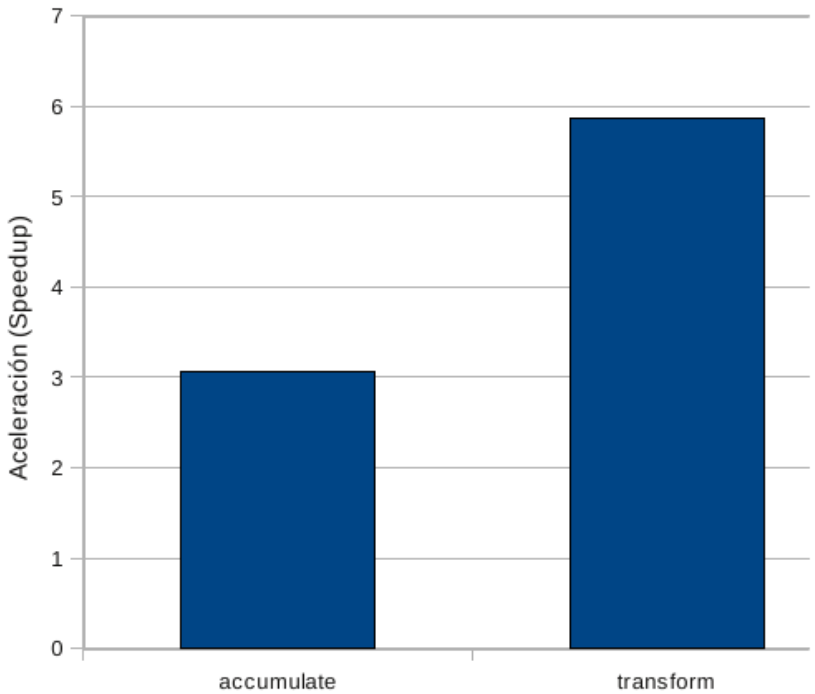


Figura: Aceleración de nuestra implementación con respecto a la de la STL

Conclusiones

- Implementación de transform y accumulate
- Aprendizaje de:
  - C++
  - STL
  - Pthreads
  - CUDA
- ¿Trabajo futuro?

¿Preguntas?

¿Preguntas?