Introduction to Cuda

Cuong Bui (Loki IT Solutions)

Tim Wood (ING)

Email: Cuong@iloki.nl

Phone: 0611957777

Loki IT Solutions

Introduction

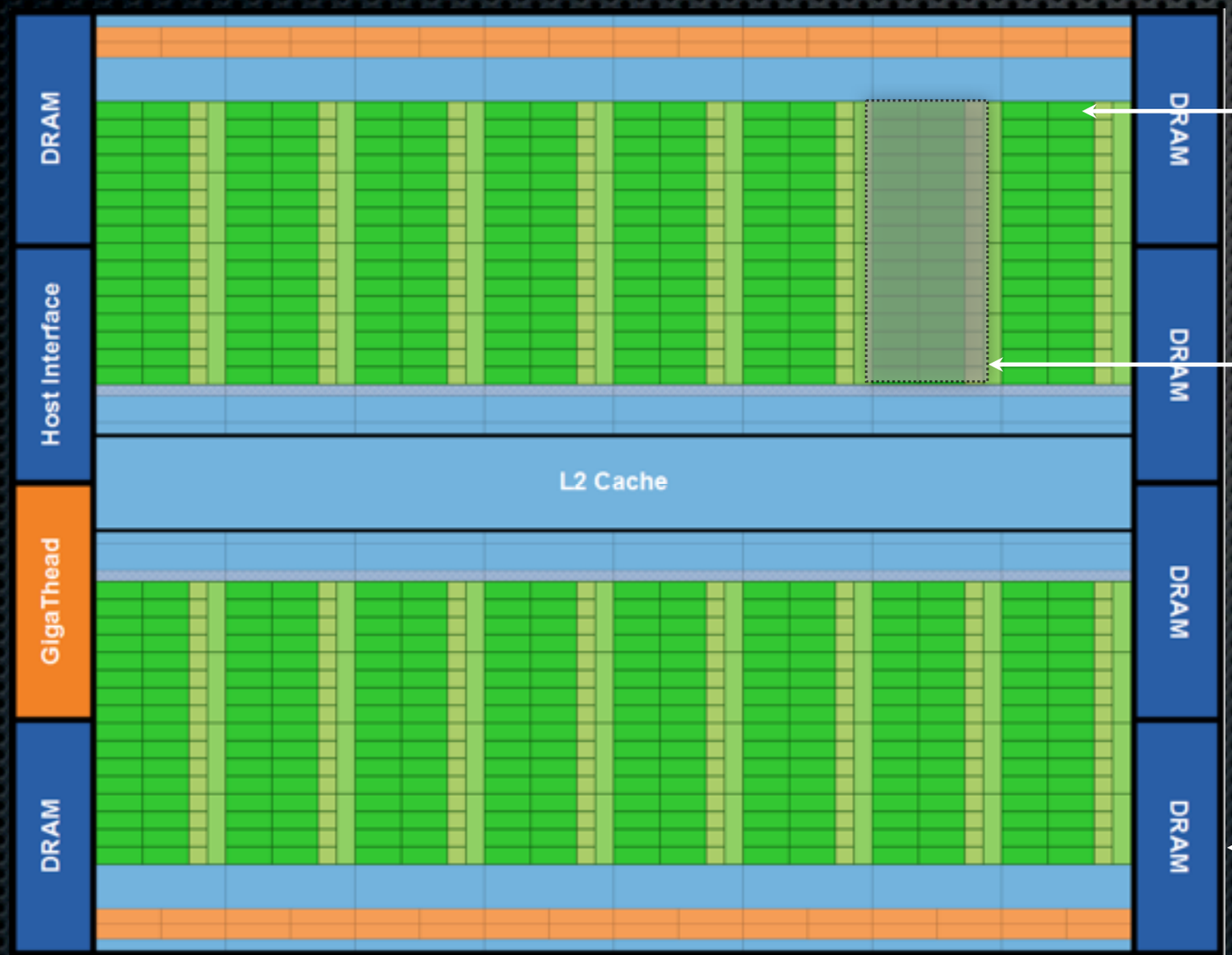
- Architecture and differences
- C++ templates / Functors
- Examples using Thrust
- Questions/Discussions

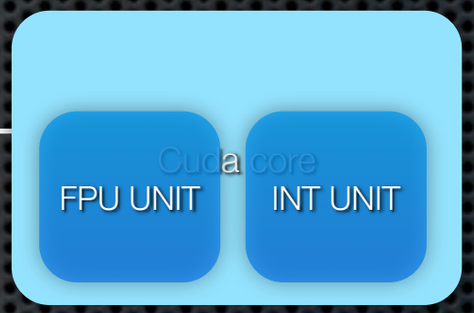
Loki IT Solutions

Cuda Terminology

- Cuda “compute unified device architecture”
  - SDK available for OSX, Windows, Linux
- Host (CPU and its Memory)
- Device (GPU and its Memory)
- Code can run on Host and Device

Architecture (Fermi)





Multi processor
(up to 16)

Host cpu

Source: Nvidia

Loki IT Solutions

Differences

- Limited memory
- Massive threading
- Massive amount of computation units
- High memory bandwidth
- General purpose vs special purpose
- More challenging to utilize full computational power

Loki IT Solutions

Performance figures

	GTX 580 Consumer	Tesla C2075	Core i7 3.3 ghz Hexa core
Cores CPU and GPU cores are not comparable	512	448	6
Single precision	1581 gflops	1030 gflops	158.4 gflops
Double precision	197.6 gflops	515 gflops	79.2 gflops

Source: Nvidia & Intel

Loki IT Solutions

C++ templates

- Generic programming (write once for all types)
  - template functions / classes
- STL Lib consist of useful containers
  - C++11 adds native threading and unordered sets

Loki IT Solutions

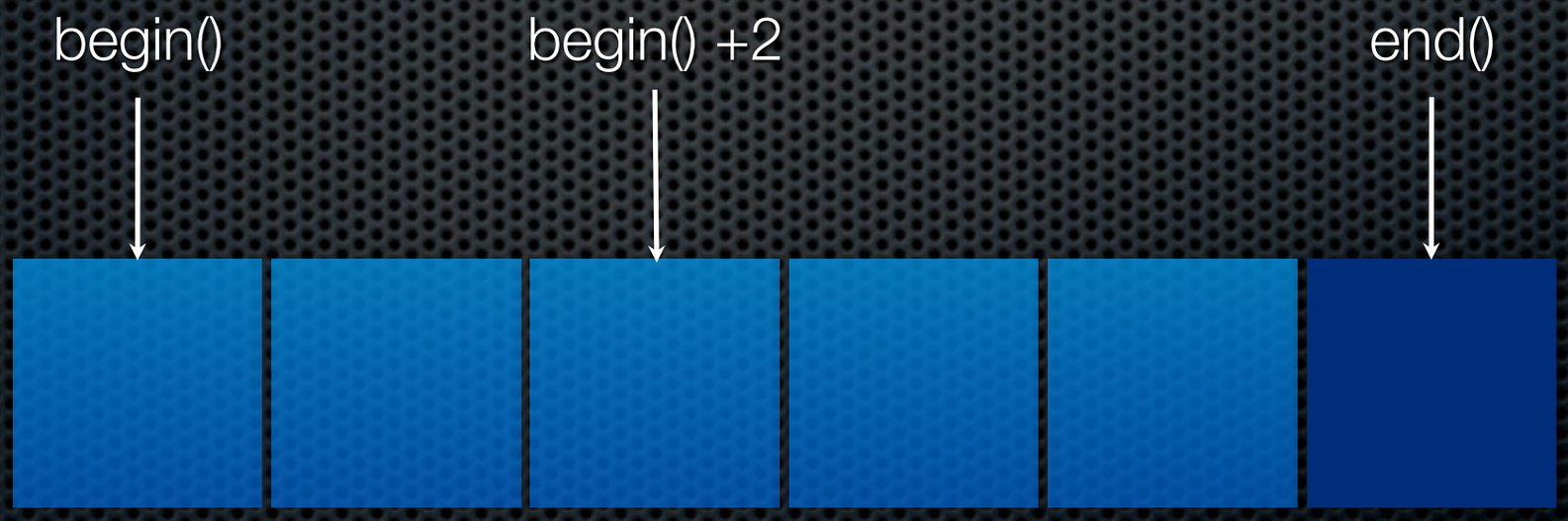
“template” example

```
#include <iostream>
using namespace std;

template <typename T>
T mull(const T &a, const T &b) {
  return a * b;
};

int main() {
  cout << " 2.0 * 5.512 = " << mull<float>(2.0, 5.512) << endl;
  cout << " 3 * 6 = " << mull<float>(3, 6) << endl;
  return 0;
}
```

Loki IT Solutions

Iterators

- Pointer like objects
  - arithmetic operations allowed
  - dereferencing



Loki IT Solutions

“Functor” example

```
template<typename T>
class mull {
public:
    T operator()(const T &a, const T &b) {
        return a*b;
    };
};
template <typename T, typename Func>
void myTransform(int n, T *x, T *y, T *z, Func &f) {
    for (int i=0; i<n; i++)
        z[i] = f(x[i], y[i]);
};

mull<float> mfl;
mull<int> mi;
mfl(1.0, 2.0);
mi(1,2);

mull<float> mf;
myTransform(N, x, y, z, mf);
```

Thrust

- Generic programming like C++ STL
- Template containers on Device
  - hides memory management
  - auto alloc/dealloc
  - hide transfers
- Tested

Loki IT Solutions

Thrust

- 2 containers
  - thrust::host_vector<T>
  - thrust::device_vector<T>
- Iterators/sequences
- Algorithms
  - sort, reduce, transform, ...

Loki IT Solutions

“Thrust” example

```cpp
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

#include <iostream>

int main(void)
{
    // H has storage for 4 integers
    thrust::host_vector<int> H(4);

    // initialize individual elements
    H[0] = 14;
    H[1] = 20;
    H[2] = 38;
    H[3] = 46;

    // H.size() returns the size of vector H
    std::cout << "H has size " << H.size() << std::endl;

    // print contents of H
    for(int i = 0; i < H.size(); i++)
        std::cout << "H[" << i << "] = " << H[i] << std::endl;

    // resize H
    H.resize(2);

    std::cout << "H now has size " << H.size() << std::endl;

    // Copy host_vector H to device_vector D
    thrust::device_vector<int> D = H;

    // elements of D can be modified
    D[0] = 99;
    D[1] = 88;

    // print contents of D
    for(int i = 0; i < D.size(); i++)
        std::cout << "D[" << i << "] = " << D[i] << std::endl;

    // H and D are automatically deleted when the function returns
    return 0;
}
```

```
nvcc --versionnvcc: NVIDIA (R) Cuda
compiler driverCopyright (c) 2005-2011
NVIDIA CorporationBuilt on
Fri_May_13_01:54:22_PDT_2011Cuda
compilation tools, release 4.0, V0.2.1221


nvcc vector.cu -o vect
./vectH has size 4
H[0] = 14
H[1] = 20
H[2] = 38
H[3] = 46
H now has size 2
D[0] = 99
D[1] = 88
```

Loki IT Solutions

“Can be used/mixed with STL”

```
std::list<float> sh_list;
sh_list.push_back(1.0);
sh_list.push_back(2.0);

//define vector on the device of floats with sh_list.size() elements
thrust::device_vector<float> d_vect(sh_list.size());
//Use thrust copy to copy the data from sh_list.begin() up to sh_list.end()
//Store the results at d_vect.begin() and so on.
thrust::copy(sh_list.begin(), sh_list.end(), d_vect.begin());

//or use constructor, the easy way:
thrust::device_vector<float> d_vect(sh_list.begin(), sh_list.end());

//init a host_vector and copy it to the device
thrust::host_vector<float> h_vect(sh_list.size());
d_vect = h_vect;
```

Algorithms

- Sort
- Transform
  - for each element in list apply operation to element
  - transform_
- Reduce
  - for each element in list, reduce to one element
    - default is sum

Loki IT Solutions

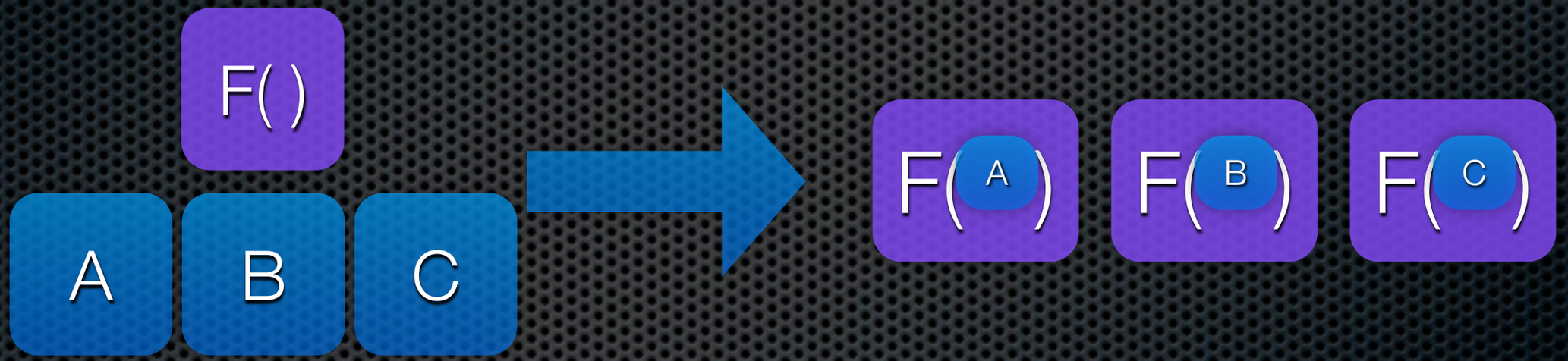
Sorting and reducing

```cpp
#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <thrust/functional.h>
#include <thrust/reduce.h>
#include <iostream>

int main(void)
{
    // D has storage for 4 integers
    thrust::device_vector<int> D(4);

    // initialize individual elements
    D[3] = 14;
    D[2] = 20;
    D[1] = 38;
    D[1] = 46;

    // sort elements on device
    thrust::sort(D.begin(), D.end());

    // print contents of D
    for(int i = 0; i < D.size(); i++)
        std::cout << "D[" << i << "] = " << D[i] << std::endl;
    // Sum all elements of D
    int sum = thrust::reduce(D.begin(), D.end());
    // Get maximum from D
    int max = thrust::reduce(D.begin(), D.end(), -1, thrust::maximum<int>() );

    std::cout << "sum: " << sum << " max: " << max << std::endl;
    return 0;
}
```

```
nvcc sort.cu -o sort
./sort
D[0] = 0
D[1] = 14
D[2] = 20
D[3] = 46
sum: 80 max: 46
```

Loki IT Solutions

Transform iterator



Loki IT Solutions

Transform, sequences

```cpp
// includes ommitted

template<typename T>
class square {
public:
    // tag for nvcc to indicate that this code can run on device and host
    __host__ __device__
    T operator()(const T &x) {
        return x*x;
    };
};


int main(void)
{
    // allocate three device_vectors with 10 elements
    thrust::device_vector<int> X(10);
    thrust::device_vector<int> Y(10);

    // explicit sequence
    // initialize X to 0,1,2,3, ....
    thrust::sequence(X.begin(), X.end());


    // compute Y = X*X
    thrust::transform(X.begin(), X.end(), Y.begin(), square<int>());

    // implicit constant iterator
    thrust::constant_iterator<int> const_2_first(2);
    thrust::constant_iterator<int> const_2_last = const_2_first + X.size();

    // compute Y = X mod 2
    // modulus<> is a predefined template that accepts 2 arguments
    thrust::transform(X.begin(), X.end(), const_2_first, Y.begin(), thrust::modulus<int>());

    // print Y
    thrust::copy(Y.begin(), Y.end(), std::ostream_iterator<int>(std::cout, "\n"));

    return 0;
}
```

```
nvcc transform.cu -o trans
./trans
0
1
0
1
0
1
0
1
0
1
```

Loki IT Solutions

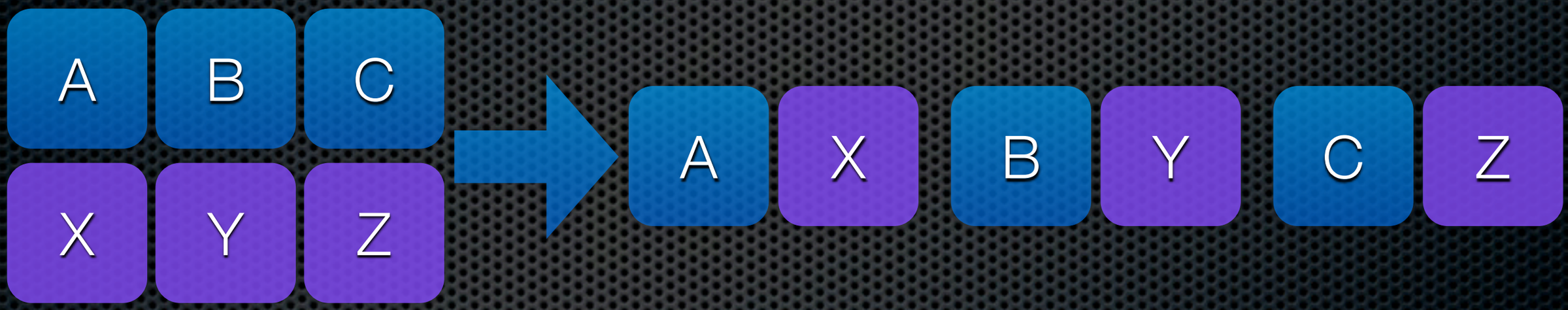
Fusion

```cpp
// includes ommitted

template<typename T>
class square : public thrust::unary_function<T,T>  {
public:
    // special key word for nvcc to indicate that this code can run on device and host
    __host__ __device__
    T operator()(const T &x) {
        return x*x;
    };
};


int main(void)
{
    // allocate three device_vectors with 10 elements
    thrust::device_vector<int> Y(10);

    // implicit sequence
    // initialize X to 0,1,2,3, ....
    thrust::counting_iterator<int> first(0);
    thrust::counting_iterator<int> last = first + Y.size();

    // compute Y = X*X using fusion and implicit sequences
    typedef thrust::counting_iterator<int> int_iterator;
    typedef thrust::transform_iterator< square<int>, thrust::counting_iterator<int> > trans_it;

    trans_it X_first( thrust::make_transform_iterator(first, square<int>()) );
    trans_it X_last( thrust::make_transform_iterator(last, square<int>()) );

    // implicit constant iterator
    thrust::constant_iterator<int> const_2_first(2);
    thrust::constant_iterator<int> const_2_last = const_2_first + Y.size();

    // compute Y = X mod 2 using fusion
    // modulus<> is a predefined template that accepts 2 arguments
    thrust::transform(X_first, X_last, const_2_first, Y.begin(), thrust::modulus<int>());

    // print Y
    thrust::copy(Y.begin(), Y.end(), std::ostream_iterator<int>(std::cout, "\n"));

    return 0;
}
```

```
nvcc transfuse.cu -o
transfuse
./transfuse
0
1
0
1
0
1
0
1
0
1
```

Loki IT Solutions

Zip iterator



Use to model arrays as semi structures

Loki IT Solutions

Coalescing (structure of arrays)

```cpp
#include <thrust/for_each.h>
#include <thrust/device_vector.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

struct custom_mul
{
    template <typename Tuple>
    __host__ __device__
    void operator()(Tuple t)
    {
        // D[i] = A[i] + B[i] * C[i];
        thrust::get<3>(t) = thrust::get<0>(t) + thrust::get<1>(t) * thrust::get<2>(t);
    }
};


int main(void)
{
    // allocate storage
    thrust::device_vector<float> A(5);
    thrust::device_vector<float> B(5);
    thrust::device_vector<float> C(5);
    thrust::device_vector<float> D(5);

    // initialize input vectors
    A[0] = 3;  B[0] = 6;  C[0] = 2;
    A[1] = 4;  B[1] = 7;  C[1] = 5;
    A[2] = 0;  B[2] = 2;  C[2] = 7;
    A[3] = 8;  B[3] = 1;  C[3] = 4;
    A[4] = 2;  B[4] = 8;  C[4] = 3;

    // apply the transformation
    thrust::for_each(thrust::make_zip_iterator(
                     thrust::make_tuple(A.begin(), B.begin(), C.begin(), D.begin())),
                    thrust::make_zip_iterator(
                              thrust::make_tuple(A.end(),   B.end(),   C.end(),   D.end())),
                    custom_mul());

    // print the output
    for(int i = 0; i < 5; i++)
        std::cout << A[i] << " + " << B[i] << " * " << C[i] << " = " << D[i] << std::endl;
}
```

```
nvcc coalesc.cu -o coalesc
./coalesc
3 + 6 * 2 = 15
4 + 7 * 5 = 39
0 + 2 * 7 = 14
8 + 1 * 4 = 12
2 + 8 * 3 = 26
```

Loki IT Solutions

Random numbers (optional if time permits)

```
// includes ommitted

#define rng_thread 100000

struct myRng : public thrust::unary_function<unsigned long long, float>
{
    unsigned long long seed;
    unsigned long long inside;
   __device__
    myRng(unsigned long long s) : seed(s), inside(0) {}
   __device__
        float operator()(unsigned int threadIdx) {
        curandState_t state;
        curand_init(seed, (unsigned long long)threadIdx, 0, &state);
           for (int i=0; i < rng_thread; i++) {
                    float x = curand_uniform(&state);
                    float y = curand_uniform(&state);
                    if ( x*x + y*y <= 1.0)
                              inside++;
           }
        return inside;
    }
};

int main(int argc, char *argv[]) {
    unsigned long long N = 100;
    if (argc == 2)
          N = atoll( argv[1] );

    unsigned long long tInside = 0;
    // implicit sequence
    thrust::counting_iterator<int> start(0);
    thrust::counting_iterator<int> end = start + N;
    // First apply the rng() followed by plus
    tInside = thrust::transform_reduce( start, end, myRng(1234ULL), 0, thrust::plus<int>() );
    unsigned long long total = N * rng_thread;
    double ratio = tInside/(double)total;
    std::cout <<"# inside: " << tInside << " total: " << total << std::endl;
    std::cout << std::fixed << std::setprecision(15);
    std::cout <<"ratio: " << ratio << " ~ Pi: " << ratio*4.0 << std::endl;
    return 0;
};
```

```
nvcc rand.cu -o rand
-lcurand

./rand 100
~ Pi: 3.141678800000000
./rand 1000
~ Pi: 3.141505600000000
./rand 10000
~ Pi: 3.141585708000000
```

Loki IT Solutions

Summary

- Different mindset/approach
  - Coalescing (prefer structure of arrays)
  - Fusion (group operations to avoid transfers)
    - implicit sequences
- Software will have to be adjusted
- Huge computational potential
- Use Thrust to make your life easier

Loki IT Solutions

Questions & Discussions

Loki IT Solutions

GPUs Applied

- Analytical Pricers
- Monte-Carlo Based Pricers
- PDE Solvers

Loki IT Solutions

Black Scholes Analytic

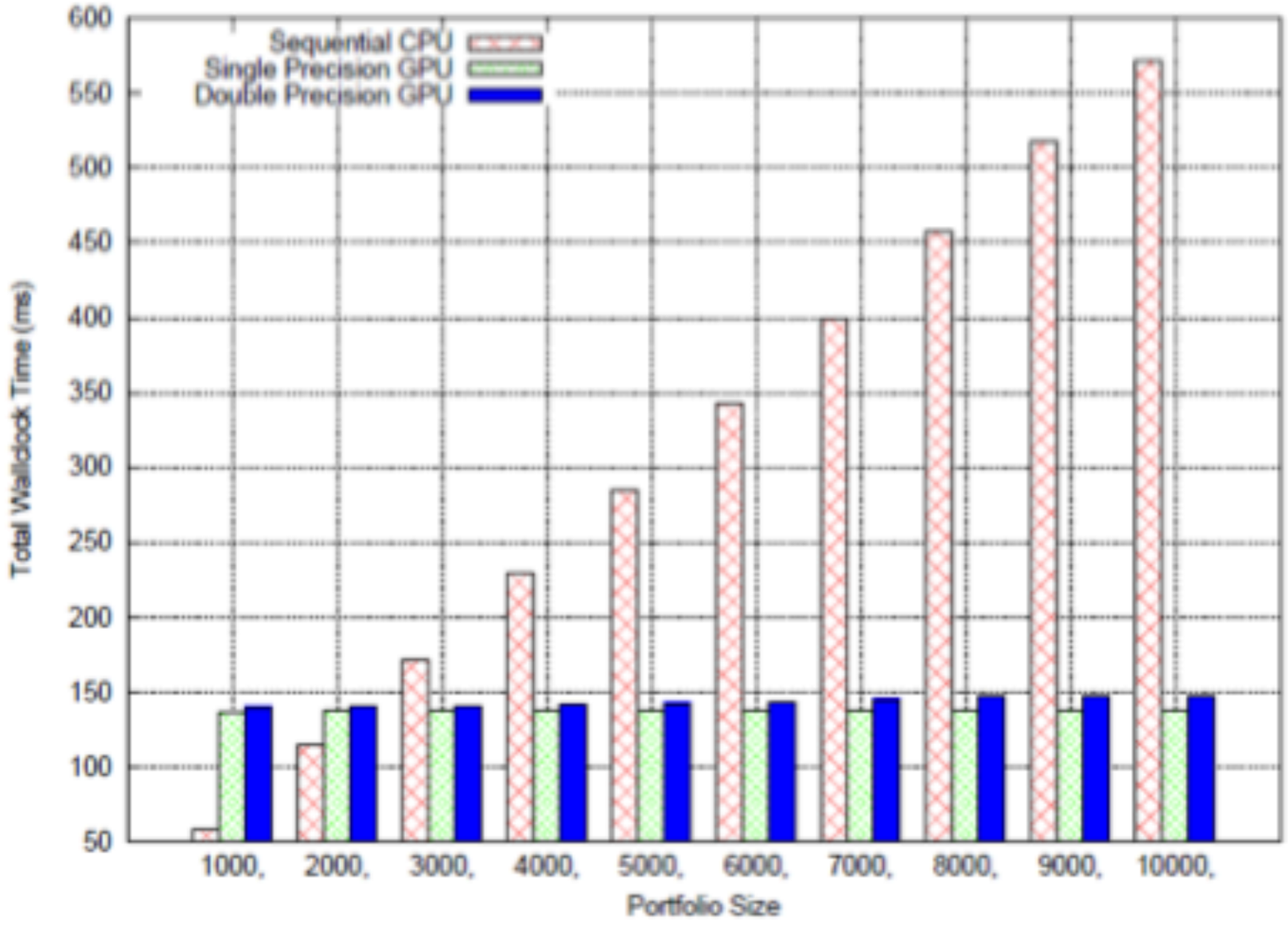


Figure 4.1: Black-Scholes HVaR Execution Time for portfolio sizes in range one to ten thousand instruments.

Loki IT Solutions

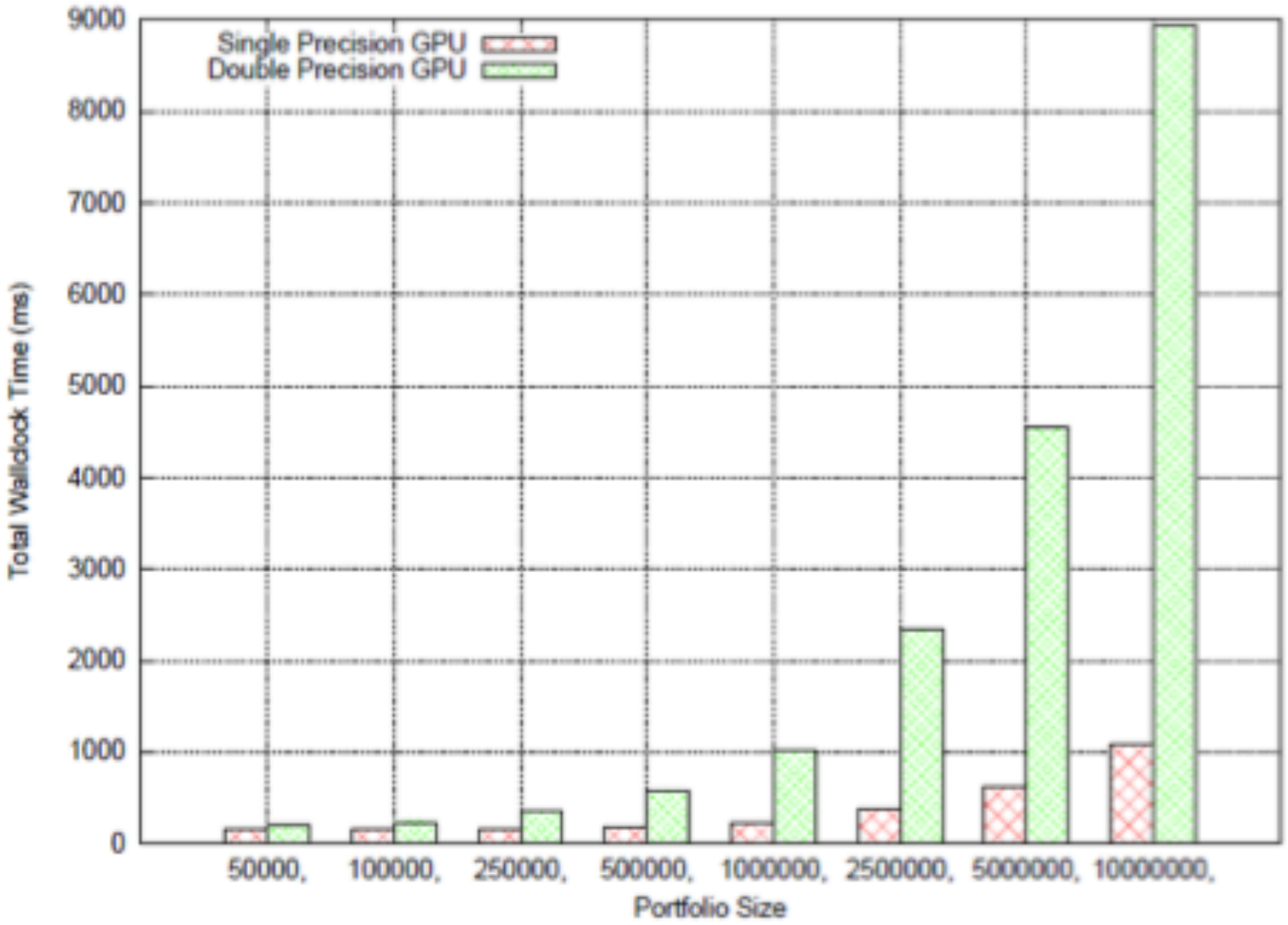


Figure 4.2: Black-Scholes Scaling Execution time for unrealistically large portfolios in range fifty thousand to ten million instruments.

Loki IT Solutions

Black Scholes Monte-Carlo

Portfolio Size	Sequential C++ CPU	Single Precision CUDA GPU	Double Precision CUDA GPU
1000	1046.4	10.2	72.6
2000	2092.2	10.2	72.6
3000	3137.4	10.2	72.6
4000	4185.0	10.8	130.8
5000	5232.6	10.8	130.8
6000	6285.0	10.8	130.8
7000	7327.8	11.4	130.8
8000	8370.0	12.6	203.4
9000	9417.6	12.6	203.4
10000	10465.2	12.6	203.4

Table 4.6: Execution times (s) of sequential C++ and CUDA GPU enabled implementations using parallelisation at instrument level.

Loki IT Solutions

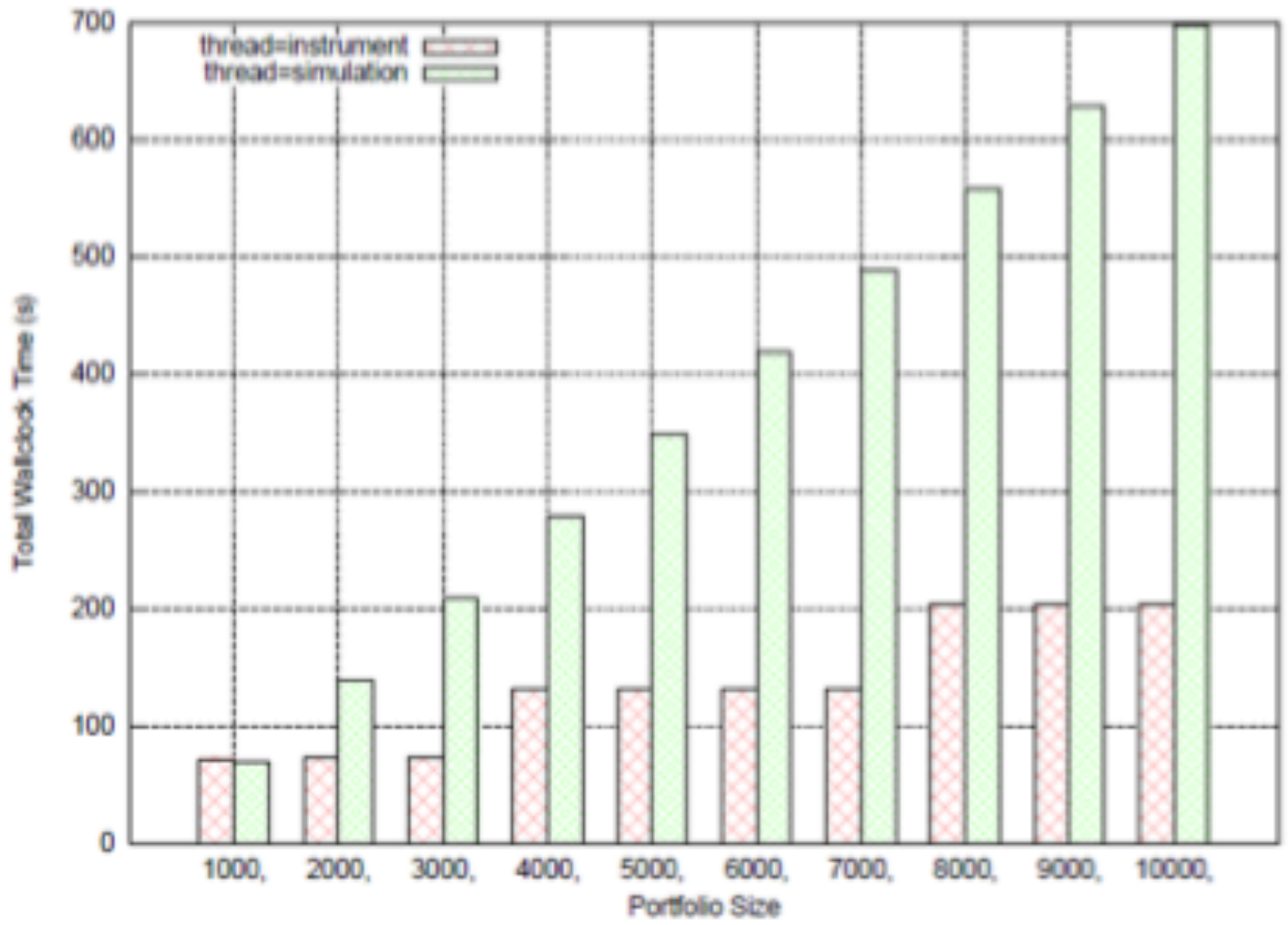


Figure 4.3: Alternate Parallelisation Strategies Effect of alternative parallelisation strategies on execution time of double precision GPU implementation.

Pricing Barrier Options With Heston

	CPU	GPU			
Portfolio	C++	Single Precision		Double Precision	
Size	Ex. Time	Ex. Time	Speed-Up	Ex. Time	Speed-Up
10	2677.8	58.8	46	66.6	40
20	5356.2	106.2	50	132.6	40
30	8034.6	157.8	51	199.2	40
40	10711.8	205.2	52	265.2	40
50	13389.6	258.6	52	331.2	40
60	16068.0	312.0	52	397.2	40
70	18745.8	363.0	52	464.4	40
80	21423.6	414.0	52	529.2	40
90	24101.0	463.2	52	596.4	40
100	26779.8	511.8	52	662.4	40

Table 4.11: Execution times (s) and improvements over sequential implementation for original CUDA implementations in double and single precision.

Loki IT Solutions

PDE Solvers

- Typically less performance improvement
- Very difficult to implement efficiently
- Growing number of libraries available

Loki IT Solutions

Acknowledgement

- Some sample code have been modified from the thrust documentation to illustrate concepts