
# – Advanced C++ Parallel STL by example of Thrust

Dmitry Mikushin

July 6, 2021

# Example: SAXPY – the functor

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

struct saxpy {

  float a;

  saxpy(float a) : a(a) {}

  __host__ __device__ float operator()(float x, float y) {
    return a * x + y;
  }

};
```

# Example: SAXPY – the functor

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

struct saxpy {

  float a;

  saxpy(float a) : a(a) {}

  __host__ __device__ float operator()(float x, float y) {
    return a * x + y;
  }

};
```

There is no predefined saxpy transform in Thrust, so here we implement this custom operation as a functor

# Example: SAXPY – main

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;

  float a = 2.0f;

  thrust::transform(X.begin(), X.end(),
          Y.begin(),
          Z.begin(),
          saxpy(a));

  for (size_t i = 0; i < Z.size(); i++)
    std::cout << "Z[" << i << "] = " << Z[i] << "\n";

  return 0;

}
```

# Example: SAXPY – main

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;

  float a = 2.0f;

  thrust::transform(X.begin(), X.end(),
          Y.begin(),
          Z.begin(),
          saxpy(a));

  for (size_t i = 0; i < Z.size(); i++)
    std::cout << "Z[" << i << "] = " << Z[i] << "\n";

  return 0;

}
```

Perform Thrust transform with user-defined saxpy functor

# Types of transforms

- **Unary:** $X[i] = f(A[i])$
- **Binary:** $X[i] = f(A[i], B[i])$
- **Ternary:** $X[i] = f(A[i], B[i], C[i])$
- **General:** $X[i] = f(A[i], B[i], C[i], \ldots)$

# Zip-iterators


Individual sequences

Sequence of tuples

# Example: ternary transform

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

struct linear_combo {

  __host__ __device__ float operator()(thrust::tuple<float,float,float> t) {

    float x, y, z;

    thrust::tie(x,y,z) = t;

    return 2.0f * x + 3.0f * y + 4.0f * z;

  }

};
```

# Example: ternary transform

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

struct linear_combo {

  __host__ __device__ float operator()(thrust::tuple<float,float,float> t) {

    float x, y, z;

    thrust::tie(x,y,z) = t;

    return 2.0f * x + 3.0f * y + 4.0f * z;

  }

};
```

Functor that takes a tuple

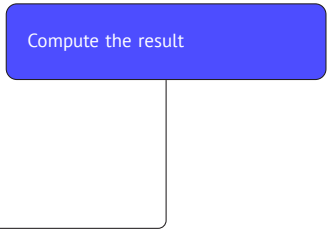
# Example: ternary transform

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

struct linear_combo {

  __host__ __device__ float operator()(thrust::tuple<float,float,float> t) {

    float x, y, z;

    thrust::tie(x,y,z) = t;

    return 2.0f * x + 3.0f * y + 4.0f * z;

  }

};
```

Break up a tuple into components

# Example: ternary transform

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

struct linear_combo {

  __host__ __device__ float operator()(thrust::tuple<float,float,float> t) {

    float x, y, z;

    thrust::tie(x,y,z) = t;

    return 2.0f * x + 3.0f * y + 4.0f * z;

  }

};
```



# Example: ternary transform

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);
  thrust::device_vector<float> U(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;
  Z[0] = 20; Z[1] = 30; Z[2] = 25;

  thrust::transform(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin(), Z.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(),
    Y.end(),
    Z.end())),
    U.begin(),
    linear_combo());

  for (size_t i = 0; i < Z.size(); i++)
  std::cout << "U[" << i << "] = " << U[i] << "\n";

  return 0;

}
```

# Example: ternary transform

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);
  thrust::device_vector<float> U(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;
  Z[0] = 20; Z[1] = 30; Z[2] = 25;

  thrust::transform(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin(), Z.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(),
    Y.end(),
    Z.end())),
    U.begin(),
    linear_combo());

  for (size_t i = 0; i < Z.size(); i++)
  std::cout << "U[" << i << "] = " << U[i] << "\n";

  return 0;

}
```

Input data ranges are expressed in terms of zip iterators

# Example: sum

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float result = thrust::reduce(X.begin(), X.end());

  std::cout << "sum is " << result << "\n";

  return 0;

}
```

# Example: sum

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

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float result = thrust::reduce(X.begin(), X.end());

  std::cout << "sum is " << result << "\n";

  return 0;

}
```

Sum is the default operation for reduce

# Example: maximum

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float init = 0.0f;

  float result = thrust::reduce(X.begin(), X.end(),
                 init,
                 thrust::maximum<float>());

  std::cout << "maximum is " << result << "\n";

  return 0;

}
```

# Example: maximum

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float init = 0.0f;

  float result = thrust::reduce(X.begin(), X.end(),
                 init,
                 thrust::maximum<float>());

  std::cout << "maximum is " << result << "\n";

  return 0;

}
```

Reduction with the maximum operation and initial value of 0.0 (works for non-negative numbers only)

# Example: index of maximum

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

typedef thrust::tuple<int,int> Tuple;

struct max_index {

  __host__ __device__ Tuple operator()(Tuple a, Tuple b) {

    if (thrust::get<0>(a) > thrust::get<0>(b))
      return a;
    else
      return b;
  }

};
```

# Example: index of maximum

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

typedef thrust::tuple<int,int> Tuple;

struct max_index {

  __host__ __device__ Tuple operator()(Tuple a, Tuple b) {

    if (thrust::get<0>(a) > thrust::get<0>(b))
      return a;
    else
      return b;
  }

};
```

Functor on (key, value) tuples: compare keys, return pairs

# Example: index of maximum

```
int main(void) {

  thrust::device_vector<int> X(3), Y(3);

  X[0] = 10; X[1] = 30; X[2] = 20; // values
  Y[0] = 0; Y[1] = 1; Y[2] = 2;    // indices‘‘

  Tuple init(X[0],Y[0]);

  Tuple result = thrust::reduce(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y.end())),
    init,
    max_index());

  int value, index;

  thrust::tie(value,index) = result;

  std::cout << "maximum value is " << value << " at index " << index << "\n";

  return 0;

}
```

# Example: index of maximum

```
int main(void) {

  thrust::device_vector<int> X(3), Y(3);

  X[0] = 10; X[1] = 30; X[2] = 20; // values
  Y[0] = 0; Y[1] = 1; Y[2] = 2;    // indices‘

  Tuple init(X[0],Y[0]);

  Tuple result = thrust::reduce(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y.end())),
    init,
    max_index());

  int value, index;

  thrust::tie(value,index) = result;

  std::cout << "maximum value is " << value << " at index " << index << "\n";

  return 0;

}
```

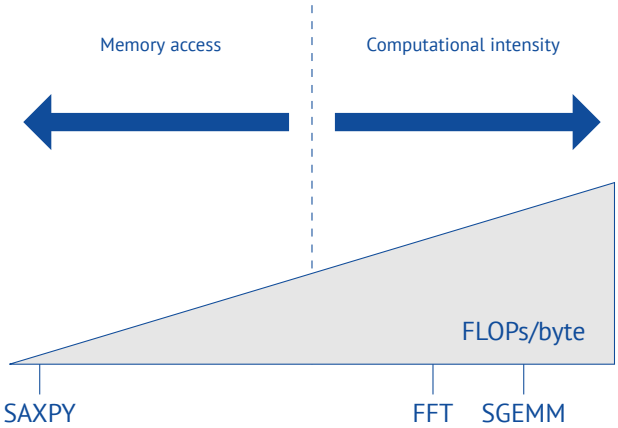
Additional array for indices

# Example: index of maximum

```
int main(void) {

  thrust::device_vector<int> X(3), Y(3);

  X[0] = 10; X[1] = 30; X[2] = 20; // values
  Y[0] = 0; Y[1] = 1; Y[2] = 2;    // indices‘‘

  Tuple init(X[0],Y[0]);

  Tuple result = thrust::reduce(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y.end())),
    init,
    max_index());

  int value, index;

  thrust::tie(value,index) = result;

  std::cout << "maximum value is " << value << " at index " << index << "\n";

  return 0;

}
```

Reduce with max index operation and explicit initial value

# Performance considerations



# Computational intensity

- For the above examples (FLOPS : byte)

| | |
|---|---|
| vector addition | 1 : 12 |
| SAXPY | 2 : 12 |
| ternary transform | 5 : 20 |
| sum | 1 : 4 |
| index of maximum | 1 : 12 |

- Optimum computational intensity for different GPUs (FLOPS : byte)

| | |
|---|---|
| GeForce GTX 280 | 7.0 : 1 |
| GeForce GTX 480 | 7.6 : 1 |
| Tesla C870 | 6.7 : 1 |
| Tesla C1060 | 9.1 : 1 |
| Tesla C2050 | 7.1 : 1 |

# Computational intensity

- For the above examples (FLOPS : byte)

| | |
|---|---|
| vector addition | 1 : 12 |
| SAXPY | 2 : 12 |
| ternary transform | 5 : 20 |
| sum | 1 : 4 |
| index of maximum | 1 : 12 |

- Optimum computational intensity for different GPUs (FLOPS : byte)

| | |
|---|---|
| GeForce GTX 280 | 7.0 : 1 |
| GeForce GTX 480 | 7.6 : 1 |
| Tesla C870 | 6.7 : 1 |
| Tesla C1060 | 9.1 : 1 |
| Tesla C2050 | 7.1 : 1 |

Actual and optimal ratios are opposite!

# Example: index of maximum (optimization)

```
int main(void) {

  thrust::device_vector<int>     X(3);
  thrust::counting_iterator<int> Y(0);

  X[0] = 10; X[1] = 30; X[2] = 20;

  Tuple init(X[0],Y[0]);

  Tuple result = thrust::reduce(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y)),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y + X.size())),
    init,
    max_index());

  int value, index;

  thrust::tie(value,index) = result;

  std::cout << "maximum value is " << value << " at index " << index << "\n";

  return 0;

}
```

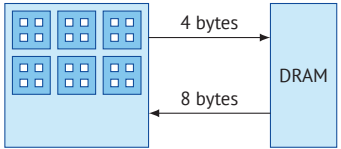
# Example: index of maximum (optimization)

```
int main(void) {

  thrust::device_vector<int>      X(3);
  thrust::counting_iterator<int> Y(0);

  X[0] = 10; X[1] = 30; X[2] = 20;

  Tuple init(X[0],Y[0]);

  Tuple result = thrust::reduce(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y)),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y + X.size())),
    init,
    max_index());

  int value, index;

  thrust::tie(value,index) = result;

  std::cout << "maximum value is " << value << " at index " << index << "\n";

  return 0;

}
```

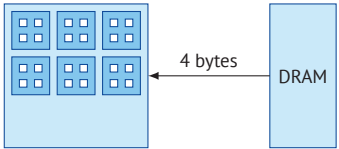
Use special counting iterator instead of explicit array of indices to optimize the data flow

# Example: index of maximum (optimization)

Initial variant




Optimized variant



# Transform loops fusion

Original two loops:

```
for (int i = 0; i < N; ++i)
  U[i] = F(X[i],Y[i],Z[i]);
for (int i = 0; i < N; ++i)
  V[i] = G(X[i],Y[i],Z[i]);
```

Two loops fused into a single loop:

```
for (int i = 0; i < N; ++i) {
  U[i] = F(X[i],Y[i],Z[i]);
  V[i] = G(X[i],Y[i],Z[i]);
}
```

# Example: fused transform

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

typedef thrust::tuple<float,float>       Tuple2;
typedef thrust::tuple<float,float,float> Tuple3;

struct linear_combo {

  __host__ __device__ Tuple2 operator()(Tuple3 t) {

    float x, y, z; thrust::tie(x,y,z) = t;

    float u = 2.0f * x + 3.0f * y + 4.0f * z;
    float v = 1.0f * x + 2.0f * y + 3.0f * z;

    return Tuple2(u,v);

  };

}
```

# Example: fused transform

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

typedef thrust::tuple<float,float>       Tuple2;
typedef thrust::tuple<float,float,float> Tuple3;

struct linear_combo {

  __host__ __device__ Tuple2 operator()(Tuple3 t) {

    float x, y, z; thrust::tie(x,y,z) = t;

    float u = 2.0f * x + 3.0f * y + 4.0f * z;
    float v = 1.0f * x + 2.0f * y + 3.0f * z;

    return Tuple2(u,v);

  };

}
```

Input data is combined into tuples

# Example: fused transform

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);
  thrust::device_vector<float> U(3), V(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;
  Z[0] = 20; Z[1] = 30; Z[2] = 25;

  thrust::transform(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin(), Z.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y.end(), Z.end())),
    thrust::make_zip_iterator(thrust::make_tuple(U.begin(), V.begin())),
    linear_combo());

  for (size_t i = 0; i < Z.size(); i++)
    std::cout << "U[" << i << "] = " << U[i] << " V[" << i << "] = " << V[i] << "\n";

  return 0;

}
```

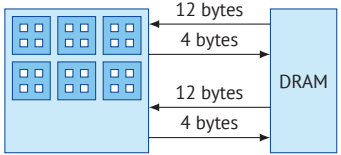
# Example: fused transform

```
int main(void) {

  thrust::device_vector<float> X(3), Y(3), Z(3);
  thrust::device_vector<float> U(3), V(3);

  X[0] = 10; X[1] = 20; X[2] = 30;
  Y[0] = 15; Y[1] = 35; Y[2] = 10;
  Z[0] = 20; Z[1] = 30; Z[2] = 25;

  thrust::transform(
    thrust::make_zip_iterator(thrust::make_tuple(X.begin(), Y.begin(), Z.begin())),
    thrust::make_zip_iterator(thrust::make_tuple(X.end(), Y.end(), Z.end())),
    thrust::make_zip_iterator(thrust::make_tuple(U.begin(), V.begin())),
    linear_combo());

  for (size_t i = 0; i < Z.size(); i++)
    std::cout << "U[" << i << "] = " << U[i] << " V[" << i << "] = " << V[i] << "\n";

  return 0;

}
```

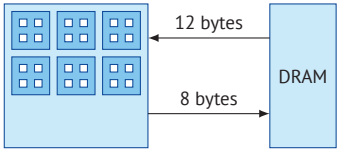
Now, both input and output data are represented as tuple sequences

# Example: fused transform

Initial variant




Optimized variant



# Reduce loops fusion

Original two loops:

```
for (int i = 0; i < N; ++i)
  Y[i] = F(X[i]);
for (int i = 0; i < N; ++i)
  sum += Y[i];
```

Two loops fused into a single loop:

```
for (int i = 0; i < N; ++i)
  sum += F(X[i]);
```

# Example: fused reduction

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/functional.h>
#include <iostream>

using namespace thrust::placeholders;

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float result = thrust::transform_reduce(
    X.begin(), X.end(),
    _1 * _1,
    0.0f,
    thrust::plus<float>());

  std::cout << "sum of squares is " << result << "\n";

  return 0;

}
```

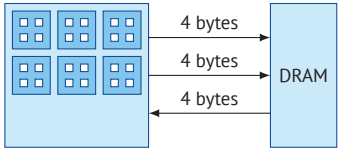
# Example: fused reduction

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

using namespace thrust::placeholders;

int main(void) {

  thrust::device_vector<float> X(3);

  X[0] = 10; X[1] = 30; X[2] = 20;

  float result = thrust::transform_reduce(
    X.begin(), X.end(),
    _1 * _1,
    0.0f,
    thrust::plus<float>());

  std::cout << "sum of squares is " << result << "\n";

  return 0;

}
```

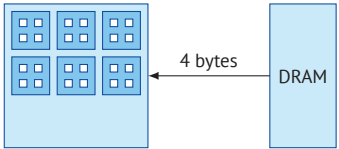
Calculate squares and sum them up in one operation

# Example: fused reduction

Initial variant




Optimized variant



# Interoperability: Thrust CUDA

- Cast the container to the raw pointer:

```
thrust::device_vector<int> d_vec(4);

int *ptr = thrust::raw_pointer_cast(&d_vec[0]);

my_kernel<<< N / 256, 256 >>>(N, ptr);

cudaMemcpyAsync(ptr, ... );
```

# Interoperability: CUDA Thrust

- Wrap the raw pointer into the special Thrust container:

```
int *raw_ptr;

cudaMalloc((void**) &raw_ptr, N * sizeof(int));

thrust::device_ptr<int> dev_ptr(raw_ptr);

thrust::fill(dev_ptr, dev_ptr + N, (int) 0);

dev_ptr[0] = 1;

cudaFree(raw_ptr);
```

# Resources

- Based on original presentation by Nathan Bell:
  Rapid Problem Solving Using Thrust
- Presentation by Ty McKercher:
  Using Thrust to Sort CUDA FORTRAN Arrays