Workshop: School of Data Science and Computation Thinking, Stellenbosch University

Introduction to Parallel Computation with GPUs: Programming with JAX, NUMBA, THRUST, and CUDA

Martin BUCHER
(bucher@sun.ac.za)

LECTURE 6 (6 March 2025)

Course dates (4-5pm SAST):

Tu 18 Feb	Th 20 Feb
Tu 25 Feb	Th 27 Feb
Tu 4 March	Th 6 March
Tu 11 March	Th 13 March

Zoom link:

`https://u-paris.zoom.us/j/82740807191?pwd=8yAGlP3R6VKbsjNCIeRjJU3u0AQtU6.1`

Course webpage:

`http://www.sun.ac.za/english/data-science-and-computational-thinking/parallelcomputation`
`https://github.com/martinabucher/cudaCourse`

Lecture 6 Outline

1. Thrust CUDA Library (with review of relevant C++ features)
2. Some More References
3. C++ Templates
4. C++ Standard Library/Standard Template Library
5. C++ STL Containers
6. C++ Algorithms: Toward "Generic" Programming
7. Parallelizing the C++ Standard Libraries
8. cuda-gdb debugger
9. Exercise: Parallel Sorting (Continued)

Some more references:

https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf

From early U. Illinois, Urbanan-Champagne Course on CUDA.
Periodically updated.
Wen-mei W. Hwu, David B. Kirk, Izzat EL Hajj,
Programming Massively Parallel Processors: A Hands-On Approach
Fourth Edition, Morgan-Kauffmann 2023

Online acces to previous edition (3rd Ed 2017) of the above
http://gpu.di.unimi.it/books/PMPP-3rd-Edition.pdf
https://www.cse.iitd.ac.in/~rijurekha/col730_2022/cudabook.pdf

https://www.nvidia.com/en-us/on-demand/session/gtc24-s62162/
?playlistId=playList-c2f59cb2-bce2-49e5-90c7-0cfa91e97eaa

Anthony Williams, C++ Concurrency in Action, Manning, 2023
https://github.com/samuel-24276/Code-of-CPP-Concurrency-In-Action/blob/master/
C%2B%2B%20Concurrency%20in%20Action%20by%20Anthony%20Williams%20(z-lib.org).pdf
[The above is a bit verbose, on C++ concurrency, not CUDA.]

A More Theoretical Reference

Herlihy et al., The Art of Multiprocessor Programming, Morgan-Kaufmann, 2021

https://github.com/amilajack/reading/blob/master/Computer_Science/The%20Art%20of%20Multiprocessor%20Programming.pdf

An Nvidia talk on Thrust

https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/thrust/IntroductionToThrust.pdf

Deep Dive into Math Libraries (Nvidia video with slides.

https://www.nvidia.com/en-us/on-demand/session/gtc24-s62162/?playlistId=playList-c2f59cb2-bce2-49e5-90c7-0cfa91e97eaa

CUDA Thrust Library

- As we has seen, CUDA programming can be tedious and long-winded, and code generally requires many more lines to do the same thing as sequential (single CPU) code for the same task.
- One may ask to what extent it may be possible to simply CUDA programming through the use of libraries where the device code is written behind the scenes hidden from the programmer.
- The Thrust Library is an attempt to generalize part of the C++ standard library so that many of the C++ Standard Library features may run in parallel on a GPU so that the transfer of data back and forth from the CPU (host) to the GPU (device) takes place transparently.

  `https://docs.nvidia.com/cuda/archive/12.3.0/thrust/index.html#`

Thrust: Parallel Sorting Example

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <thrust/random.h>

int main() {
  // Generate 32M random numbers serially.
  thrust::default_random_engine rng(1337);
  thrust::uniform_int_distribution<int> dist;
  thrust::host_vector<int> h_vec(32 << 20);
  thrust::generate(h_vec.begin(), h_vec.end(), [&] { return dist(rng); });

  // Transfer data to the device.
  thrust::device_vector<int> d_vec = h_vec;

  // Sort data on the device.
  thrust::sort(d_vec.begin(), d_vec.end());

  // Transfer data back to host.
  thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());
}
```

Commentary on previous code snippet

1. Much of the C++ template library (and likewise the Thrust library) is implemented in header files. Classes and functions are defined in terms of templates, which are instantiated (turned into compilable code for the particular template type variable values on an as needed basis). C++ also make extensive use of inline functions (which can be incorporated into library code). This approach differs from C libraries, which are generally take the form as pre-compiled code packaged in the form of shared objects (dynamic libraries). C header files typically contain only function prototypes (and not function definitions). C++ header contain a lot more.
2. C++ makes extensive use of namespaces in order to avoid bugs due to the same name being used in more than one module. Namespaces have a hierarchical structure, where the separator :: takes the place of the / in unix path names or the . in the Python namespace system. Thus `thrust::default_random_engine` is the name `default_random_engine` in the `thrust` namespace and is distinguished from and cannot be confused with a variable `default_random_engine` in the root namespace.

Commentary on previous code snippet (2)

1. Whereas the C++ Standard Library has just one implementation defined in the header file

   ```
   #include <vector>
   ```

   and this implementation is not thread safe (it is guaranteed to work only in a sequential environment. The Thrust library has two types of vectors: one for use on the host, and another for use on the device. These are defined in the header files

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

C++ Containers: Basic Types

`https://en.cppreference.com/w/cpp/container`

Sequence containers

Sequence containers implement data structures which can be accessed sequentially.

array (C++11)	fixed-sized inplace contiguous array (class template)
vector	resizable contiguous array (class template)
inplace_vector (C++26)	resizable, fixed capacity, inplace contiguous array (class template)
hive (C++26)	collection that reuses erased elements' memory (class template)
deque	double-ended queue (class template)
forward_list (C++11)	singly-linked list (class template)
list	doubly-linked list (class template)

C++ Containers: Associative Containers

Associative containers

Associative containers implement sorted data structures that can be quickly searched ($O(\log n)$ complexity).

set	collection of unique keys, sorted by keys (class template)
map	collection of key-value pairs, sorted by keys, keys are unique (class template)
multiset	collection of keys, sorted by keys (class template)
multimap	collection of key-value pairs, sorted by keys (class template)

C++ Containers: Other Important Types

Container adaptors

Container adaptors provide a different interface for sequential containers.

stack	adapts a container to provide stack (LIFO data structure) (class template)
queue	adapts a container to provide queue (FIFO data structure) (class template)
priority_queue	adapts a container to provide priority queue (class template)
flat_set (C++23)	adapts a container to provide a collection of unique keys, sorted by keys (class template)
flat_map (C++23)	adapts two containers to provide a collection of key-value pairs, sorted by unique keys (class template)
flat_multiset (C++23)	adapts a container to provide a collection of keys, sorted by keys (class template)
flat_multimap (C++23)	adapts two containers to provide a collection of key-value pairs, sorted by keys (class template)

What is a C++ Iterator?

Answer: An abstraction of a C pointer.

What are the possible operations on a pointer?

- Essential operations:
  - *iterator; // dereferencing
  - ++iterator; // advancing by one, in a linked list vector
  - iterator1 != iterator2 testing for equality/inequality
- Possible for some iterators:
  - −−iterator; // moving backward in a list, not possible in a singly linked list
  - iterator=iterator+n; // where n is an integer (generally taken to be of size t)

Using C++ Iterators: Toward Generic Programming

Option 1:

```
for(auto i=container.begin(); i!=container.end(); i++)
       actionFunc(*i);
```

rather than

Option 2:

```
for(int i=0; i<vec.len(); ++i)
  action(vec[i]);
  // equivalent to *(vec+i); // Possible only for random access iterators
```

auto makes the C++ use type inference.

For option 1, the same code works for almost all containers:
vectors, arrays, singly linked lists, doubly-linked lists, maps
(aka dictionaries in Python),

Code in Option 2 works only for arrays and vectors. One wants to be able to use the same code for all containers, to the maximum extent possible.

What are C++ templates?

- C++ templates allow C++ types to be compilation variables.
- Multiple versions of functions or classes can be written once using just once template that can be instantiated. Less typing—and more importantly, easier to maintain and debug.
- Python has another maybe maybe simpler but much slower solution. Types of variables are not known until a function is actually run. In C++ by contrast the types of all variable are known at compilation time. (Many) errors are caught during compilation rather than during execution. In C++ there is no extra overhead from type checking when a program is run.
- C++ language development policy is not to make any compromises that slow down execution. On the other hand, compilation is can be very slow. Work is moved from execution to compilation.
- The C++ compiler is really smart. A lot of computation can be done by the compiler. The C++ compiler/preprocessor is Turing complete. There is a famous example of a program that uses (or misuses) the C++ compiler to compute all the primes up to a certain value. Unfortunately, the code is not very portable, because the C++ standard does not prescribe the form of compiler error messages, and these are used to print the output.

C++ Templates: A Trivial Example

https://www.geeksforgeeks.org/templates-cpp/

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

// One function works for all data types. This would work
// even for user defined types if operator '>' is overloaded
template <typename T> T myMax(T x, T y)
{
    return (x > y) ? x : y;
}

int main()
{
    // Call myMax for int
    cout << myMax<int>(3, 7) << endl;
    // call myMax for double
    cout << myMax<double>(3.0, 7.0) << endl;
    // call myMax for char
    cout << myMax<char>('g', 'e') << endl;

    return 0;
}
```

C++ Template Example (II) – Bubble Sort

```
#include <iostream>
using namespace std; // This is a bad practice.

// We can use this for any data type that supports
// comparison operator < and swap works for it.
template <class T> void bubbleSort(T a[], int n)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = n - 1; i < j; j--)
            if (a[j] < a[j - 1])
                swap(a[j], a[j - 1]);
}

int main(){
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);
    // calls template function
    bubbleSort<int>(a, n);
    cout << " Sorted array : ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
Output: Sorted array : 10 20 30 40 50
```

Simplest vector container example

```
#include <iostream>
#include <vector>
//using namespace std; // Bad (dangerous) practice that
// dumps all the variables defined in the standard library
// into the root namespace. May lead to nasty hard to find bugs

int main() {

      // Creating a vector of 5 elements
      std::vector<int> v = {1, 4, 2, 3, 5};

      for (int i = 0; i < v.size(); i++)
        std::cout << v[i] << " ";
      return 0;
}
```

```
#include <cmath>
#include <iomanip>
#include <iostream>
#include <map>
#include <random>
#include <string>

int main(){
    std::random_device r;
    // Choose a random mean between 1 and 6
    std::default_random_engine e1(r());
    std::uniform_int_distribution<int> uniform_dist(1, 6);
    int mean = uniform_dist(e1);
    std::cout << "Randomly-chosen mean: " << mean << '\n';
    // Generate a normal distribution around that mean
    std::seed_seq seed2{r(), r(), r(), r(), r(), r(), r(), r()};
    std::mt19937 e2(seed2);
    std::normal_distribution<> normal_dist(mean, 2);

    std::map<int, int> hist;
    for (int n = 0; n != 10000; ++n)
        ++hist[std::round(normal_dist(e2))];

    std::cout << "Normal distribution around " << mean << ":\n"
              << std::fixed << std::setprecision(1);
    for (auto [x, y] : hist)
        std::cout << std::setw(2) << x << ' ' << std::string(y / 200, '*') << '
}
```

Generic Algorithms in the C++ Standard Library: Sorting

```
https://en.cppreference.com/w/cpp/algorithm/sort

Defined in header <algorithm>

template< class RandomIt >
void sort( RandomIt first, RandomIt last );
 (1)  (constexpr since C++20)

template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy,
           RandomIt first, RandomIt last );
 (2)  (since C++17)

template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );
 (3)  (constexpr since C++20)

template< class ExecutionPolicy, class RandomIt, class Compare >
void sort( ExecutionPolicy&& policy,
           RandomIt first, RandomIt last, Compare comp );
 (4)  (since C++17)
```

Generic Programming in C is Ugly and Error Prone

- Some people regard `void` pointers as the root of all evil.
- We need to write a sort routine where the type of the objects to be sorted is unknown and can be anything.
- void* is an address but the pointer must be converted (or "cast") to a pointer type pointing to a specific type of object. Code with void pointers is hard to read and can be confusing, and even worse thwarts the compilers ability to check for type consistency.
- The C library generic sorting function, defined in the header ¡stdlib.h¿, has the function prototype

```
void qsort( void* ptr, size_t count, size_t size,
            int (*comp)(const void*, const void*) );
```

where the last argument is a pointer to a function.

```
int less_than_float(const void *a, const void *b){
  float *a_float=(float*a) a;
  float *b_float=(float*a) b;
  int result=(*a_float<*b_float);
  return(result);
}
```

C++ STL Container Algorithms: `std::transform` (I)

`std::transform` seems like a trivial example. It is a little like the Python map (but without all the overhead)
Exploits Data Parallelism when there is no need for data sharing of intermediate results or synchronization
Defined in header file <`algorithm`>.

```
template<class InputIt, class OutputIt, class UnaryOp >
OutputIt transform( InputIt first1, InputIt last1,
                    OutputIt d_first, UnaryOp unary_op );
 (1)  (constexpr since C++20)

template< class ExecutionPolicy,
    class ForwardIt1, class ForwardIt2, class UnaryOp >
ForwardIt2 transform( ExecutionPolicy&& policy,
                      ForwardIt1 first1, ForwardIt1 last1,
                      ForwardIt2 d_first, UnaryOp unary_op );
 (2)  (since C++17)
```

C++ STL Container Algorithms: `std::transform` (I)

Binary functions

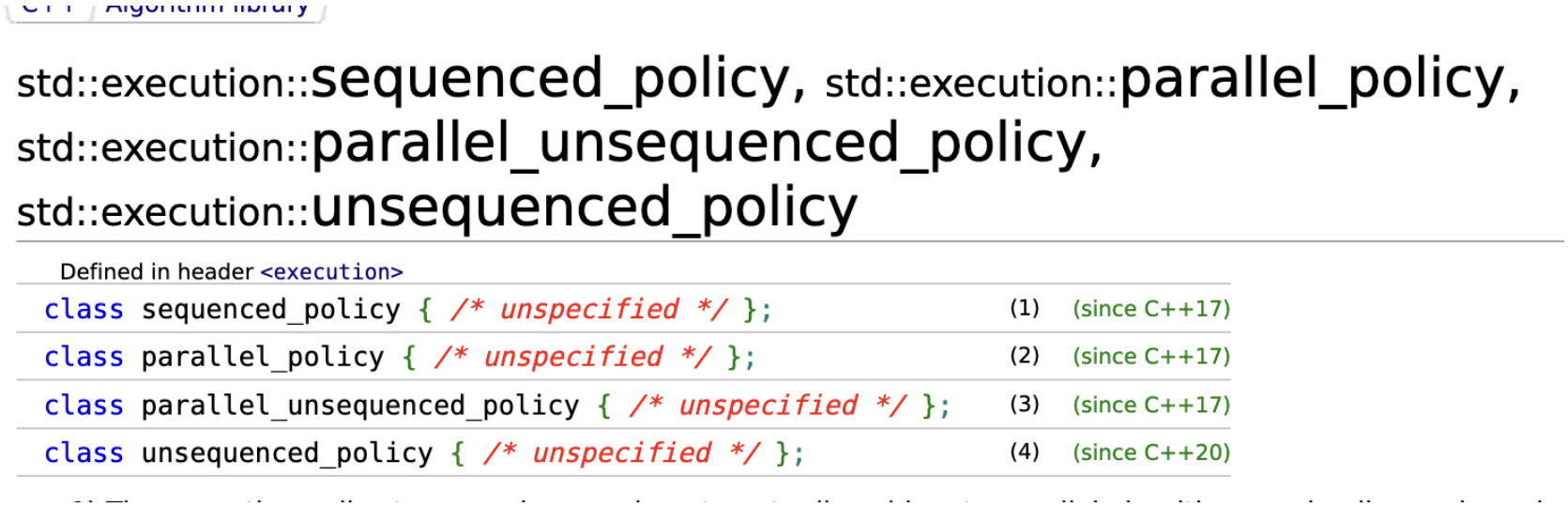
```
template< class InputIt1, class InputIt2,
          class OutputIt, class BinaryOp >
OutputIt transform( InputIt1 first1, InputIt1 last1, InputIt2 first2,
                    OutputIt d_first, BinaryOp binary_op );
 (3)  (constexpr since C++20)

template< class ExecutionPolicy,
          class ForwardIt1, class ForwardIt2,
          class ForwardIt3, class BinaryOp >
ForwardIt3 transform( ExecutionPolicy&& policy,
                      ForwardIt1 first1, ForwardIt1 last1,
                      ForwardIt2 first2,
                      ForwardIt3 d_first, BinaryOp binary_op );
```

What is "execution policy"?

- It sounds like a rather pompous bureaucratic name!
- In C++11 (new 2011 major update to the language), support for concurrency was included, and one of the goals was to allow parallel execution of algorithms in the C++ Standard Library (which has subsumed the for Standard Template Library, which was first developed independently).

`https://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t`



std::execution::sequenced_policy, std::execution::parallel_policy,
std::execution::parallel_unsequenced_policy,
std::execution::unsequenced_policy

Defined in header <execution>

```
class sequenced_policy { /* unspecified */ };                (1)  (since C++17)
class parallel_policy { /* unspecified */ };                  (2)  (since C++17)
class parallel_unsequenced_policy { /* unspecified */ };      (3)  (since C++17)
class unsequenced_policy { /* unspecified */ };               (4)  (since C++20)
```

Provides compiler cues on whether parallelizing will lead to conflicts. No parallelization is default.

Some more Thrust examples (I)

- **General types and operators**

```
struct negate_float2
{
    __host__ __device__
    float2 operator()(float2 a)
    {
        return make_float2(-a.x, -a.y);
    }
};

// declare storage
device_vector<float2> input  = ...
device_vector<float2> output = ...

// create functor
negate_float2 func;

// negate vectors
transform(input.begin(), input.end(), output.begin(), func);
```

Some more Thrust examples (II)

```
#include <thrust/reduce.h>

// declare storage
device_vector<int>   i_vec = ...
device_vector<float> f_vec = ...

// sum of integers (equivalent calls)
reduce(i_vec.begin(), i_vec.end());
reduce(i_vec.begin(), i_vec.end(),    0, plus<int>());

// sum of floats (equivalent calls)
reduce(f_vec.begin(), f_vec.end());
reduce(f_vec.begin(), f_vec.end(), 0.0f, plus<float>());

// maximum of integers
reduce(i_vec.begin(), i_vec.end(),    0, maximum<int>());
```

GDB and Cuda

- GDB is the GNU debugger, commonly used by unix programmers. It was developed alongside GCC and all that.
- GDB allows one to go through code line by line, seeing the source code, set breakpoints, and more.
- GDB is a graphical extension of the earlier UNIX DBX.
- NVIDIA has enhanced GDB to include functionality for debugging CUDA programs. It is invoked by `cuda-gdb` in the place of `gdb`.

For more details, see:

`https://en.wikipedia.org/wiki/GNU_Debugger`

`https://docs.nvidia.com/cuda/cuda-gdb/index.html#walk-through-examples`

Exercise: GPU Sorting

`https://en.wikipedia.org/wiki/Quicksort`
`https://en.wikipedia.org/wiki/Sorting_algorithm`

Name	Best	Average	Worst	Memory	Stable	In-place	Method
In-place merge sort	—	—	$n \log^2 n$	1	Yes	Yes	Merging
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Yes	Selection
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No		Partitioning & Selection
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	No	Merging
Tournament sort	$n \log n$	$n \log n$	$n \log n$	n[7]	No		Selection
Tree sort	$n \log n$	$n \log n$	$n \log n$ (balanced)	n	Yes	No	Insertion

What can be achieved for an ideal parallel machine? On a CUDA GPU? How would you program it? What are the challenges?

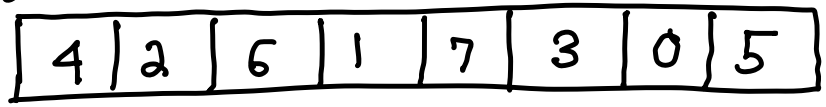
Sorting algorithms: Quicksort

Quicksort is a classic "divide-and-conquer" algorithm of complexity $O(N \log)$ for the typical case on a sequential machine.

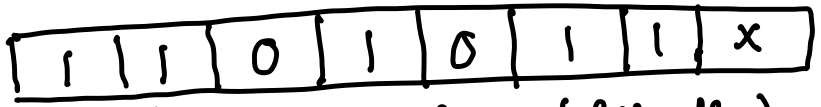
- 1. A pivot element is chosen which ideally is very close to the median of the set.
- 2. The elements (other than the pivot element) are partitioned into two sets: those less than the pivot, and (2) those greater than the pivot.

  This can be done in place.
- 3. This partitioning can be parallelized.
- 4. The algorithm above can be applied recursively, to the elements before and after the pivot until the elements are completely sorted.

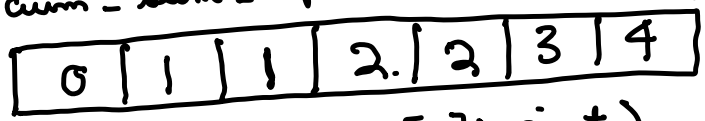
a[n] Pivot



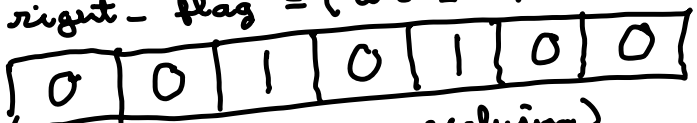
left_flag = (a[n] <= pivot)



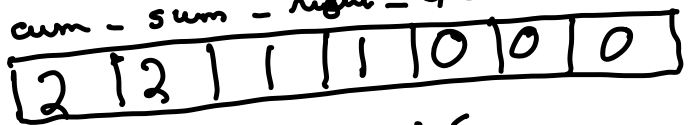
left_sum = sum_left_exclusive(left_flag)



right_flag = (a[n] > pivot)



cum_sum_right_exclusive)




```
if (left_flag[n]){
   dest = dest_left[n];
   b[dest] = a[n];
} else {
   dest = end_index - dest_right[n]
   b[dest] = a[n]
```

More Reduction (Cumulative Sums)

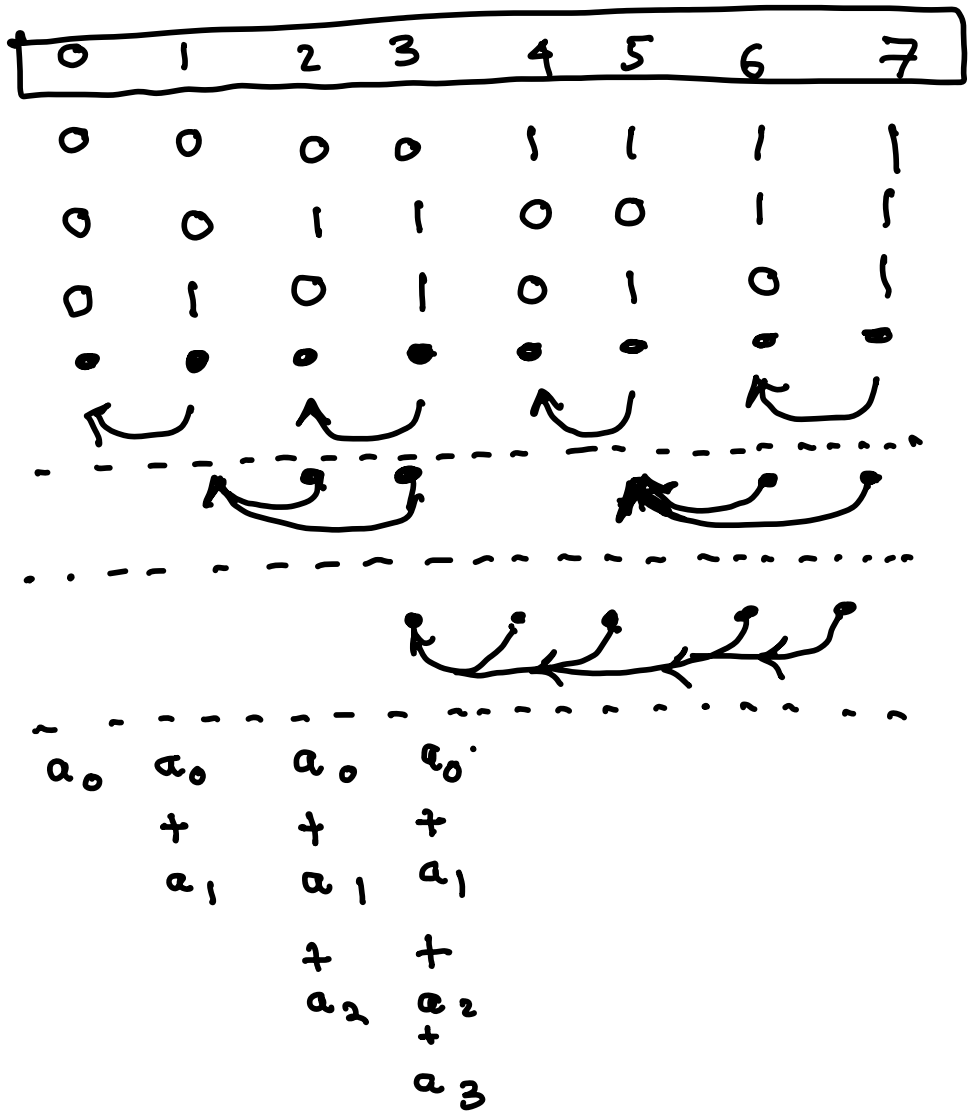
$a[n]$ is an array where $n = 0..N$
Inclusive left cumulative sum

$$cs[n] = \sum_{i=0}^{n} a[n]$$

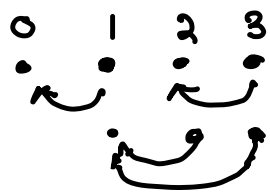
Exclusive left cumulative sum

$$cs[n] = \sum_{i=0}^{n-1} a[n]$$

We may also define the same from the right.
These are operations that would be $O(N)$ on a sequential machine, but like the full sums we saw earlier are $O(\ln N)$ on a parallel machine.


0 1 2 3 4 5 6 7
a_0 a_0 a_0 a_0
+ + +
a_1 a_1 a_1
+ +
a_2 a_2
+
a_3

```
if((n%2) > 0 ) 1-1
   a[m] += a[m-1];
if((n%4) > 1 ) 2-1
   a[n] += a[m]
if((n%8) > 3 ) 4-1
```



$m = n - (n\%2) + 1 - 1$

$m = n - (n\%4) + 2 - 1$

$m = n - (n\%8) + 4 - 1$

$m = n - (n\%16) + 8 - 1$

$\cdots$

$m = n - (n\%2**k) + 2**(k-1) - 1$