Concurrent Programming With C++

Erik Sevre
Ph.D. Student, SNU
Computational Science and Technology

Code on Github:

https://github.com/esevre/ParallelProgramming

Survey:

What language(s) do you use?

C++	Python	C	Java
Swift	Ruby	C#	Objective-C
Haskel	Fortran	R	JavaScript
Fortran	Perl	Go	Erlang

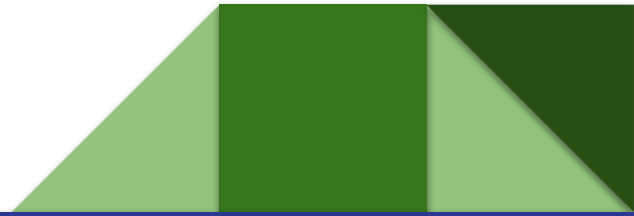
Today’s Focus: C++

I’m a C++ expert, so that is my most comfortable language

C++ allows for you to control all elements of your code from bits to abstract ideas

Remember: all programming is moving bits in memory

The general ideas presented here can be used in any modern language
(even Fortran)



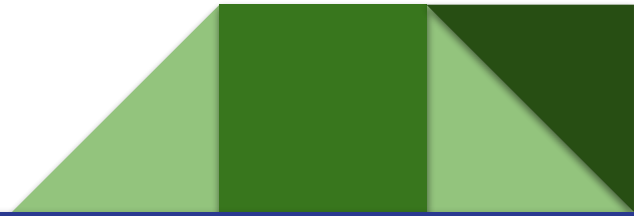
What is Parallel Programming?

Multiple processors working on a single task

Typically: Task divided into smaller tasks over parts of the big task

Small tasks often involve the exact same computations

Tasks that need to be done are related to each other



What is Concurrent Programming?

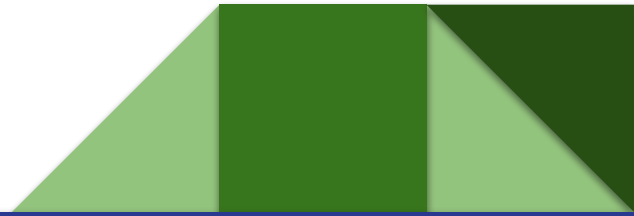
Having two processes work at the same time

A more general word than “parallel programming”

Parallel programming is a subset of concurrent programming

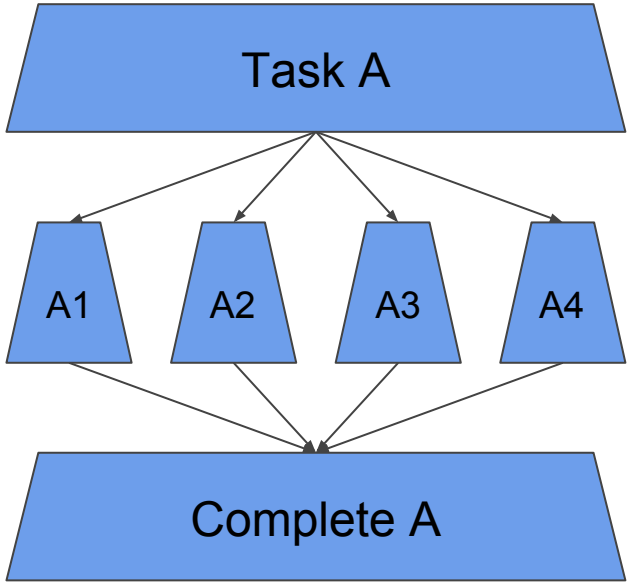
Every task might be for a different computation

“Concurrent” is being used more recently, so add this word to your vocabulary

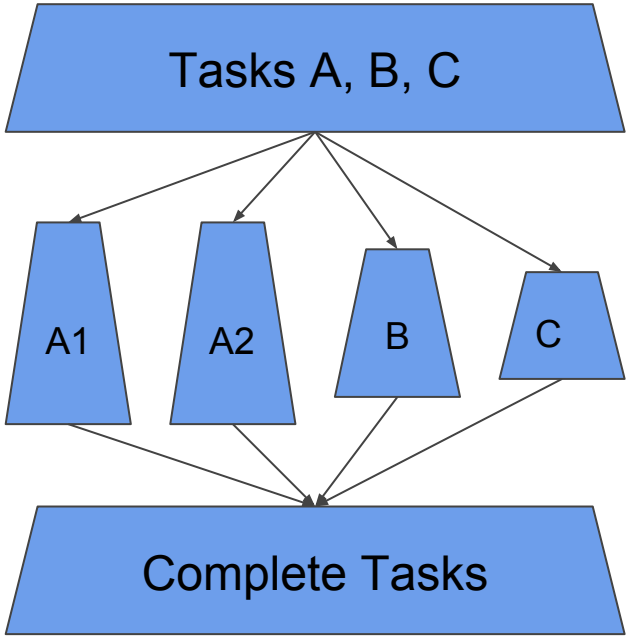


Parallel vs. Concurrent Programming

Parallel Programming



Concurrent Programming



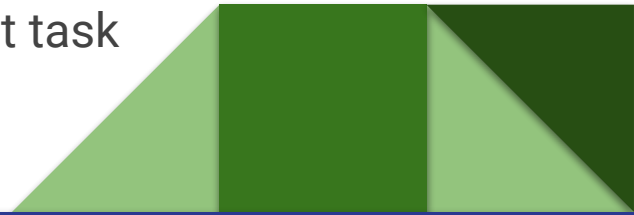
Concurrent Programming Example

Recording an MP3 audio file:

- One process captures an audio buffer
- Another process encodes with MP3
- A third process writes MP3 buffer to file

All tasks run concurrently

Parallel programming can be used for a single concurrent task



Writing Good Code

Concurrent Programming Strategies

Program around the “big” idea

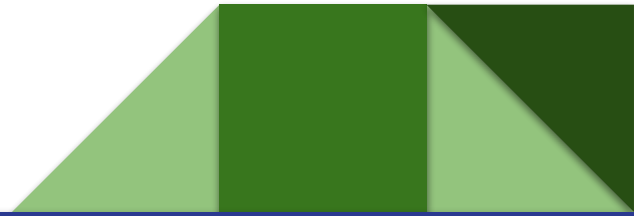
Focus on the algorithms first

Adopt strategies that work for parallel and serial programs

Write readable code

Get to know your compiler

Write C++ (ok, maybe I’m biased)



C++ Algorithms & Numerics

Look at <algorithm> and <numeric> in the C++ STL

Algorithms based on iterators over sets of data

Iterator idea: don't move the data, pass around iterators to data

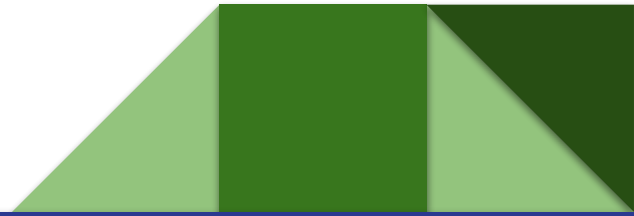
C++17 standards added "parallel execution policies" to algorithms

This means execution policies can be passed to algorithm

allowing the compiler to implement a parallel variant of the algorithm

Even if you don't use these algorithms,

we can learn from the style



Algorithms in the C++ STL

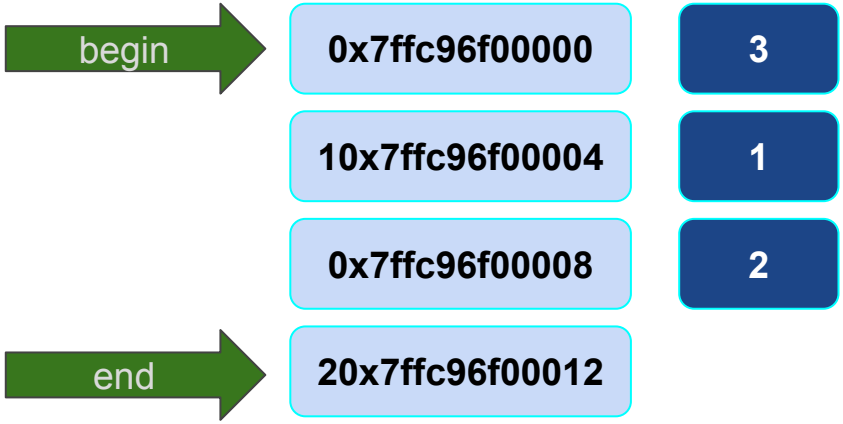
```
std::vector<int> v = {3,1,2};

std::sort(v.begin(), v.end());
```

Iterators point at data

Like pointers, but customized

Work with iterators,

move data when needed



Execution policies (C++17)

	execution::seq	execution::sequenced_policy	
is_execution_policy	execution::par	execution::parallel_policy	
	execution::par_unseq	execution::parallel_unsequenced_policy	

Non-modifying sequence operations

all_of (C++11)	count	find	search
any_of (C++11)	count_if	find_if	search_n
none_of (C++11)	mismatch	find_if_not (C++11)	lexicographical_
for_each	equal	find_end	lexicographical_
for_each_n (C++17)	adjacent_find	find_first_of	

Modifying sequence operations

copy	fill	remove	remove_copy
copy_if (C++11)	fill_n	remove_if	remove_copy_if
copy_n (C++11)	generate	replace	replace_copy
copy_backward	generate_n	replace_if	replace_copy_if
move (C++11)	swap	reverse	reverse_copy
move_backward (C++11)	iter_swap	rotate	rotate_copy
transform	swap_ranges	unique	unique_copy
	sample (C++17)	random_shuffle (until C++17)	shuffle (C++11)

Operations on uninitialized storage

uninitialized_copy	uninitialized_copy_n (C++11)	uninitialized_fill	uninitialized_f
uninitialized_move (C++17)	uninitialized_move_n (C++17)		

Partitioning operations

is_partitioned (C++11)	partition	stable_partition
partition_point (C++11)	partition_copy (C++11)	

Sorting operations

is_sorted (C++11)	sort	partial_sort	nth_element
is_sorted_until (C++11)	stable_sort	partial_sort_copy	

Binary search operations

lower_bound	upper_bound	binary_search	equal_range

Set operations (on sorted ranges)

merge	set_difference	set_symmetric_difference	includes
inplace_merge	set_intersection	set_union	

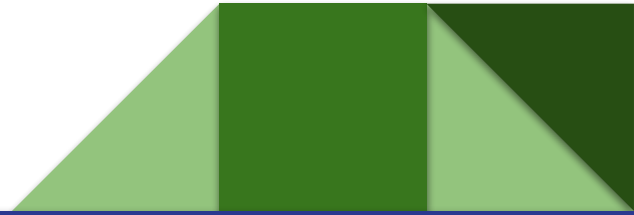
Organize Code With Functions

Only worry about making a function faster if it uses a lot of CPU time

Don’t worry about cost of calling functions, there are ways around this

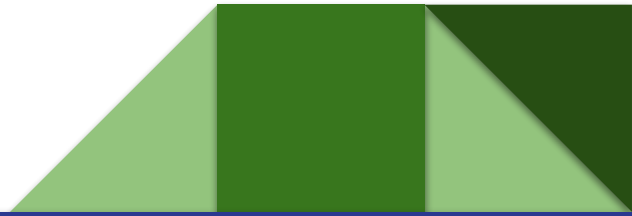
- The compiler will always try to inline a function
- In C++ constexpr will force compile time execution (and inline) if possible

We can look at the assembly to understand if functions have a negative impact on our computations (the compiler will often inline functions for us)



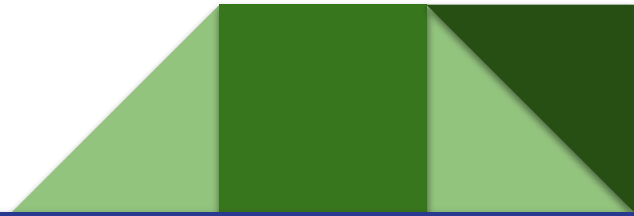
Reasonable Code

```
if (v1.size() < 1000) {
    val = ES::dot_product(v1, v2, ES::ParallelPolicy::SEQ);
} else {
    val = ES::dot_product(v1, v2, ES::ParallelPolicy::PAR);
}
std::cout << "dot product is: " << val << "\n";
```



Creating a General Function

```
template <class VectorType>
auto dot_product(const VectorType &a,
                 const VectorType &b,
                 ParallelPolicy policy)
{
   switch (policy) {
       case ParallelPolicy::SEQ:
           return dot_product_sequential(a, b);
       case ParallelPolicy::PAR:
           return dot_product_parallel(a, b);
   }
}
```



Let Your Compiler Optimize the Code

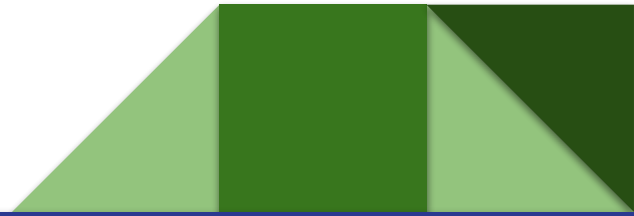
Look at your assembly (From a C++ POV)

Compiler Explorer (C++/Rust/D, no Fortran support)

Learn techniques to do computations at compile time

Let me show you a cool example

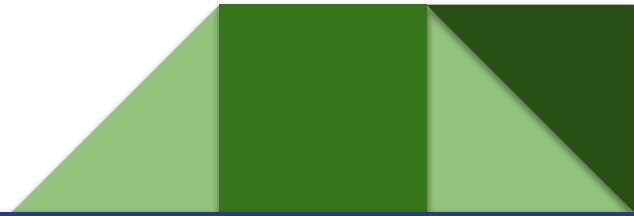
Some compilers use concurrent programming to compile with a cluster



Computations at Compile Time

In C++ we can force computation at compile time with `constexpr`

We can’t do all computations at compile time,
but lookup tables and common computations can be done at compile time



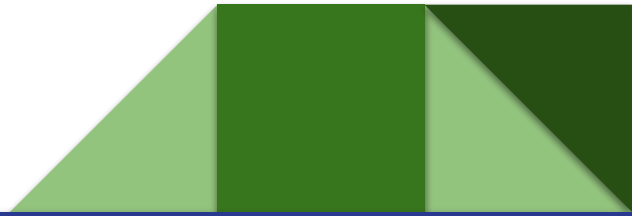
Communicate with Custom Types

You can define a global type used in your functions

This allows you to change the type in one place, and changes all the code

From the compilers perspective it uses the right type, and will optimize as needed

```
using NumberType = float;

using Vector = std::vector<NumberType>;

enum class ParallelPolicy { SEQ, PAR };
```



Let’s Look at Code

Questions?