This course has already ended.
The latest instance of the course can be found at: Principles of Algorithmic Techniques: 2023 Autumn

« 2.1 Welcome Course materials 2.3 A first exercise »

CS-E3190 / Warm-up to Programming Exercises / 2.2 Hello, world!

Hello, world!

Let us now get started with programming.

We will use the C++ programming language for the exercises, so let us start with a simple C++ program that prints out the string `Hello, world!`.

Here we go ...

```
#include <iostream>

int main(int argc, char **argv) {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

... or what is the same as a downloadable file: hello.cpp.

At this course, we will rely on a preconfigured and appropriately resource-limited C++ configuration at the grading platform. This configuration is accessible via the **scratchpad** below.

If you are experienced with C++, you are welcome to compile and run the program yourself on your local C++ configuration, or on one of the Aalto University Linux workstations. (More on accessing and using the Linux workstations later in this chapter.)

Please go ahead and submit hello.cpp to the scratchpad.

Scratchpad

Here you can submit an arbitrary C++ file for compilation and execution on the grader hardware.

Grading. This exercise awards you 1 points on your first submission. You are welcome to make as many submissions as you like, but only the first submission will award points.

Points 1 / 1 My submissions 64 Deadline Monday, 19 September 2022, 20:00 To be submitted alone

This course has been archived (Friday, 30 December 2022, 16:00).

Scratchpad

- Submit a C++17 file for compilation and execution using the dialog below.
- The compiler is `g++ (Debian 8.3.0-6) 8.3.0` and the compilation command is `g++ -O3 -Werror -Wall --pedantic -std=c++17 -march=native -fopenmp -o test test.cpp`.
- Shared-memory parallel programming is supported via OpenMP.
- Please use this service for working with the CS-E3190 course materials only. Please note that failure to comply with this policy is a violation of the code of conduct at this course. Also note that all submissions are archived.
- The prevailing resource limits are subject to change without notice and can be found in the compilation and execution feedback.

test.cpp

Choose File No file chosen

Submit

The compilation and execution reports

After you submit a file to the scratchpad, you will witness a report of the compilation and the execution of the compiled binary. Let us first look at a compilation report:

```
g++ -O3 -Werror -Wall --pedantic -std=c++17 -march=native -fopenmp -o test test.cpp

--
SUCCESS  [    427 ms,   40428 KiB]

Time limit:     10 seconds wall-clock time
Memory limit:   2 GiB total for control group
Process limit:  32 processes in control group
File limit:     64 MiB capacity in file system
```

We observe that the compilation was a success. The compilation took 427 milliseconds of wall-clock time and used 40428 kilobinary-bytes ($40428 \cdot 2^{10} = 41398272$ bytes) of memory. The compilation was resource-constrained to

- a maximum of 10 seconds of wall-clock time,
- a maximum of 2 gigabinary-bytes ($2 \cdot 2^{30} = 2147483648$ bytes) of memory,
- a maximum of 32 processes, and
- a maximum of 64 megabinary-bytes ($64 \cdot 2^{20} = 67108864$ bytes) of file system capacity.

Resource constraints of this type will be in place at each exercise in this course.

Let us now look at the execution report:

```
./test
Hello, world!

--
SUCCESS  [     55 ms,     368 KiB]

Time limit:     20 seconds wall-clock time
Memory limit:   100 GiB total for control group
Process limit:  32 processes in control group
File limit:     [read-only file system]
```

We observe that the compiled binary `test` was executed, with the expected output `Hello, world!` delivered in 55 milliseconds and using 368 kilobinary-bytes of memory.

Some further examples

Based on the first example, the scratchpad may not appear like a serious programming interface. This, however, is not true.

First, the grading platform is a fairly powerful parallel compute node. In fact, it is one of the nodes at the Triton cluster for high-performance computing at Aalto University.

Remark

In precise terms, the grading platform is a Dell PowerEdge C4130 Rack Server with two 2.5-GHz Intel Xeon E5-2680 v3 CPUs (Haswell microarchitecture, 12 cores/CPU, 24 cores total) and 128 GiB main memory (8 × SK Hynix HMA42GR7MFR4N-TF 16 GiB 2133 MHz DDR4 DIMM).

Currently, 23 of the 24 cores are available for scratchpad and grading use.

Warning

Due to the large number of participants at the course, we may need to restrict the resources available to enable more grading throughput. This in practice means that the grading server will grade multiple submissions in parallel, so your submission may not have access to the entire server. Accordingly, we reserve the right to change the prevailing resource limits without notice.

Second, the C++ compiler gives us low-level programming access to the hardware, all the way to intrinsics and inlined assembly language specifically targeting the processor microarchitecture if we so choose.

Here are two further examples, which you are welcome to submit to the scratchpad to witness the available performance.

- example-mmtest.cpp – A matrix multiplication test using cache-blocking and inlined assembly language for the AVX2-vectorized inner loops.
- example-bandtest.cpp – A memory bandwidth test.

Many of the features in these examples are outside the scope of this course, but if you are familiar with e.g. parallel programming, we welcome you to use the hardware to its fullest extent.

For example, here is an execution report for the matrix multiplication test:

```
./test
n = 384
wtime       =     30 ms,  perf       =    3.74 Gflop/s [   0.16 Gflop/s/core]
wtime_inner =      4 ms,  perf_inner =   27.73 Gflop/s [   1.21 Gflop/s/core]
n = 768
wtime       =     18 ms,  perf       =   49.04 Gflop/s [   2.13 Gflop/s/core]
wtime_inner =      2 ms,  perf_inner =  418.83 Gflop/s [  18.21 Gflop/s/core]
n = 1536
wtime       =     56 ms,  perf       =  128.35 Gflop/s [   5.58 Gflop/s/core]
wtime_inner =     13 ms,  perf_inner =  568.49 Gflop/s [  24.72 Gflop/s/core]
n = 3072
wtime       =    233 ms,  perf       =  248.98 Gflop/s [  10.83 Gflop/s/core]
wtime_inner =     90 ms,  perf_inner =  646.97 Gflop/s [  28.13 Gflop/s/core]
n = 6144
wtime       =   1174 ms,  perf       =  395.00 Gflop/s [  17.17 Gflop/s/core]
wtime_inner =    662 ms,  perf_inner =  700.36 Gflop/s [  30.45 Gflop/s/core]

--
SUCCESS  [   2479 ms, 1775004 KiB]

Time limit:     20 seconds wall-clock time
Memory limit:   100 GiB total for control group
Process limit:  32 processes in control group
File limit:     [read-only file system]
```

Assuming the matrices are stored in cache-blocked representation, we observe a very satisfactory performance—as measured by `wtime_inner` and `perf_inner`—of over 700 billion double-precision floating point operations per second when $n = 6144$.

Remark

A double-precision $n \times n$ matrix for $n = 6144$ consumes $8n^2 = 301989888$ bytes, and the classical algorithm for matrix multiplication executes $2n^3 - n^2 = 463818719232$ double-precision floating point operations.

Remark

The per-core theoretical peak performance of a Xeon E5-2680 v3 CPU is $2.5 \cdot 10^9 \cdot 2 \cdot 2 \cdot 4 = 40 \cdot 10^9$ double-precision floating point operations per second, or 40 Gflop/s/core. Indeed, the clock rate of the CPU is 2.5 GHz ($2.5 \cdot 10^9$ clock cycles per second) and each core contains 2 vectorized fused-multiply-add units (FMAs), each of which executes 2 floating point operations (multiply and add) on vectors of 4 independent inputs in parallel on every clock cycle.

Let us again observe that it is not our intent at this course to make you an expert on specific computer microarchitectures and their peak performance, but we do want to share with you that the tools we use are real-world professional tools that give unrestricted low-level access to the hardware if you know what you are doing. Further resources on software optimization for Intel x86-64 microarchitectures are available, for example, via Intel and at Agner Fog's software optimization resources.

The C++ programming language (C++17)

Let us conclude this section with some basic online resources to the C++ programming language.

This is to say that you need not be an expert in all the details of the C++ programming language to successfully solve the programming exercises at this course, but—as in all programming—it is useful to have available precise references and resources. Indeed, "the C++ programming language" in fact has multiple versions, so it is prudent to choose one standardized version for use at the course.

At this course, **we work with the 2017-standardized version of C++**, precisely the standard ISO/IEC 14882:2017, or, somewhat more colloquially, C++17. A final working draft of the standard is available for free.

There are also a number of interactive online references to C++. For example, one such resource is cppreference.com. Further useful C++ resources can be found here.

If you are completely new to C++, but have programming experience with some other programming language (which we expect you do), a simple way to pick up basic constructs and idioms of C++ is to do a web search for `C++ by example` and study appropriate examples. The questions and answers on C++ in Stack Overflow may also be useful.

Next, we will look at a warmup exercise.