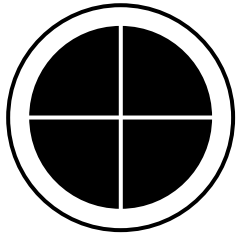
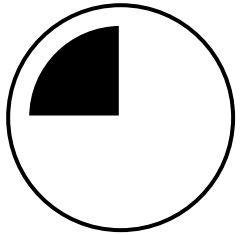
Code Optimization

Thomas Lienbacher and Frederick Knauder

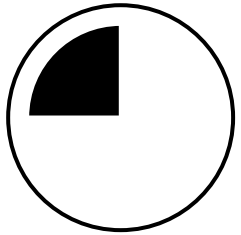


Overview

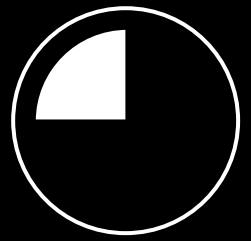
- Branchless Code
- Loop unrolling
- OpenMP
- SIMD
- Practical Demo
- Quiz



What is branchless code?



What are the benefits?



Theory: Branchless Code cheat sheet

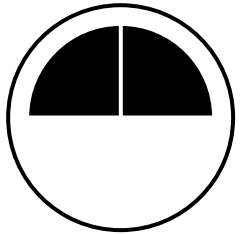
What is Branchless Code?
Branchless programming is **a programming technique that eliminates the branches.**

What is branching?
Branching is when we split the flow of the program into two parts based on a runtime condition check.

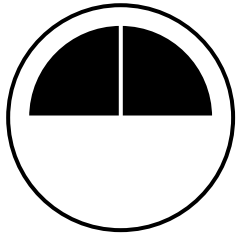
Branches are created by conditional statements like if, else, and loops.

How to avoid Branches?
The most common way to avoid branching is to replace branches with mathematical operations or conditional moves. This reduces the total jumps in your code.

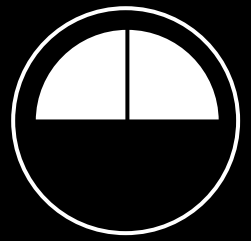
Note:
Modern compilers can recognize branching patterns and replace them with branch-less counterparts.



What is loop unrolling?



What are the benefits?

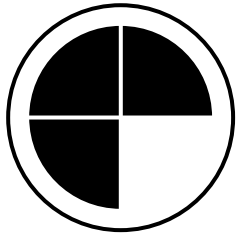


Theory: Loop unrolling cheat sheet

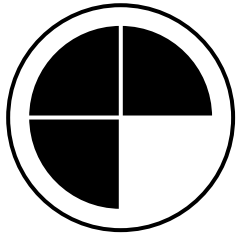
What is loop unrolling?
Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program.
It increases the number of instructions per iteration of the loop, thus reducing the number of times the loop branch logic is executed.

Is unrolling a loop always more efficient?
Unrolled loops are not always faster. They generate larger binaries. They require more instruction decoding. They use more memory and instruction cache.

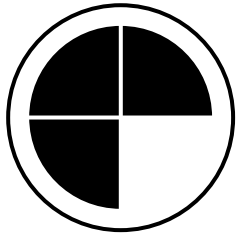
Note:
Modern compilers can already unroll loops effectively.



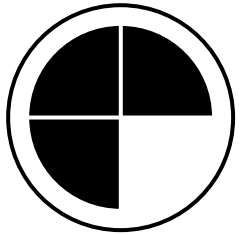
What is OpenMP?



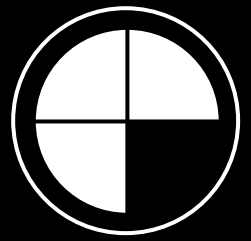
How can I use it?



When should I use it?



How is data handled?

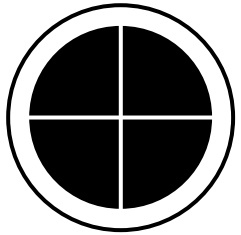


Theory: OpenMP cheat sheet

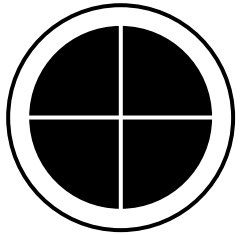
What is OpenMP?
OpenMP is an API to easily add multiprocessing to a program. Typically used for loop-level parallelism, but it also supports function-level parallelism.

Private variables:
Each thread will have its own local copy. A private variable is not initialized and the value is not maintained for use outside the parallel region.

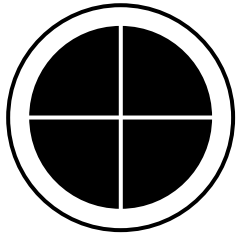
Shared variables:
It is visible to and accessible by all threads simultaneously. Shared variables must be used with care because they could cause race conditions.



What is SIMD?

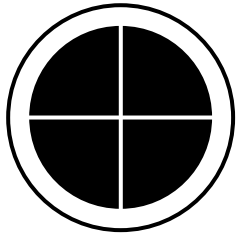


Which extensions exist?

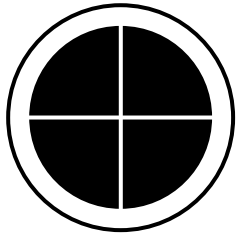


Theory: SIMD

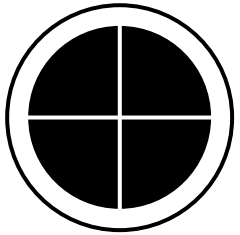
Extensions	
MMX	1997
SSE	1999
SSE2	2000
SSE3	2004
SSSE3	2006
SSE4	2006
AVX	2011
AVX2	2013
AVX512	2017



How can I use it?

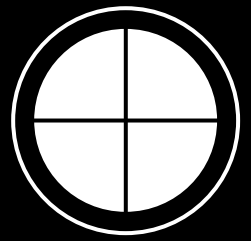


Which flags should I use?



Theory: SIMD

GCC flags	
-O3	Turn on all optimizations
-march=native	Optimize for the host architecture
-msse	Enable SSE instructions
-msse4	Enable SSE4 instructions
-mavx	Enable AVX instructions
-mavx512f	Enable AVX512 foundation instructions



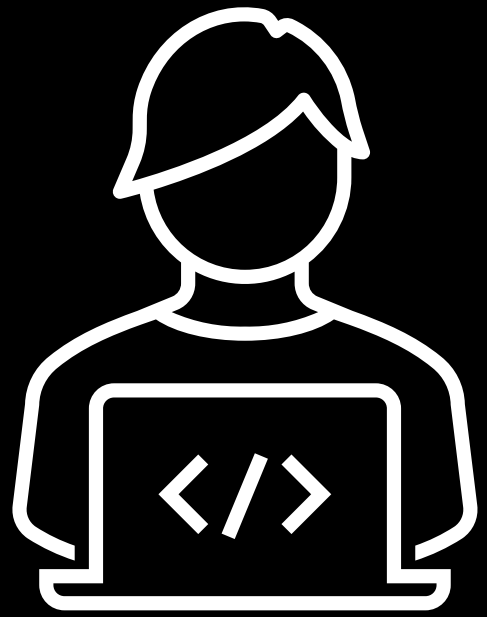
Theory: SIMD cheat sheet

What is SIMD?
SIMD, or Single Instruction Multiple Data, is a technique used in computer architecture and programming to process multiple data elements in parallel using a single instruction. SIMD is implemented in modern processors with special processor instructions and registers that can operate on multiple data elements at once.

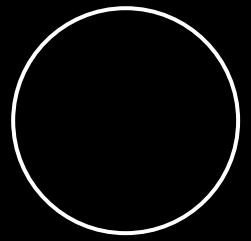
What is SIMD used for?
SIMD is extensively used for high-performance computing, particularly in applications such as video processing, image processing, video games, and scientific simulations.

Note:
Modern compilers can already auto-vectorize code efficiently.

Practical Demo



Recap-Quiz



Interesting links

- OpenMP tutorial/blog: http://jakascorner.com/blog/
- OpenMP specification: https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf
- OpenMP examples: https://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf
- SIMD intrinsics Rust: https://doc.rust-lang.org/stable/core/arch/x86/index.html
- SIMD intrinsics Intel: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
- Godbolt compiler explorer: https://godbolt.org/