A practical course on

Advanced systems programming in C/Rust

Dr. Atsushi Koshiba
Marcel Schneider

Today’s topic!

Performance

Outline

- Performance
    - What is the performance?
- Optimization
    - Cache locality
    - Multi-threading
    - SIMD instructions
- Measurement
    - clock_gettime()
    - perf
- Assignment

Performance

Metrics that quantify the effectiveness/efficiency of computer systems

- **Processing time (s)**
  - Simulation, numerical computation
- Throughput (Op/s, B/s, Tx/s)
  - HPC, database, network, storage
- Latency (s, cycles)
  - Database, network, storage, HFT
- Memory usage (Bytes, pages)
  - Embedded systems
- Energy usage (V, A, W, W/op)
  - Embedded systems, laptops, smartphones, tablets
- Binary size (Bytes)
  - Embedded systems, loading from remote
- Compiling time (s)
  - Testing, development

How to improve the performance

Many features affecting the performance of modern computer systems

- Developers need to carefully (explicitly) design their code to utilize the features

Hardware:

- Pipelining
- **CPU Caches**
  - Cache size
  - Caching protocol
  - False sharing
  - Cache misses
  - Prefetcher
  - Page misses
- **SIMD instructions**
- Branch predictor
- Interrupts
- Simultaneous multithreading
- Dynamic frequency scaling
- NUMA

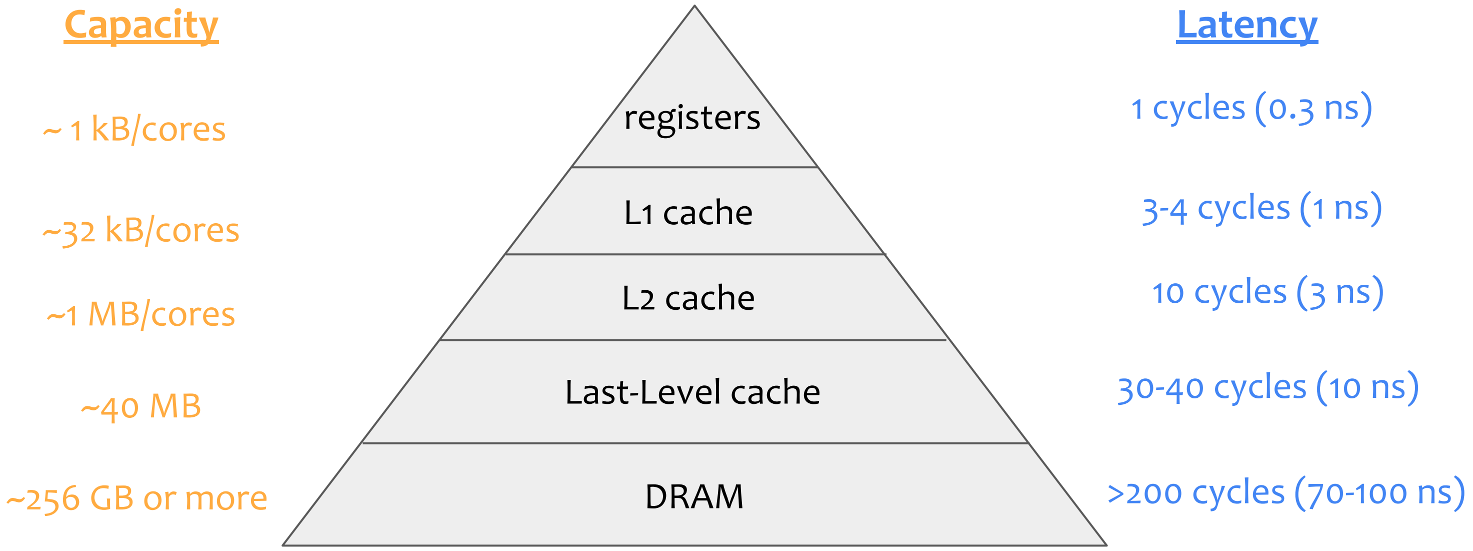
Software:

- **Multi-threading**
- Software Caches
  - Page cache
- Scheduler
  - Energy aware scheduling
  - Real time scheduling
  - Overcommitment
  - Process priority
- Context switch
- Swapping
- Compiler/Linker optimizations
- ASLR
- CPU affinity
- CPU governor

1. CPU cache

DRAM accesses (~100 ns) are much slower than CPU instructions (< 1 ns)

- Multiple caches (L1, L2, L3/LLC) mitigate memory access latencies
  - Memory data accessed by CPUs are fetched to caches
  - Every cache miss invokes fetching a cache line of data (64 Bytes)



1. CPU cache

Cache locality

- Programs have a tendency to access the same set of data repeatedly
    - Spatial locality — location accessed recently is likely to be accessed again
    - Temporal locality — location nearby accessed recently is likely to be accessed
- The cache holds data recently accessed and neighbors of the data

Prefetchers

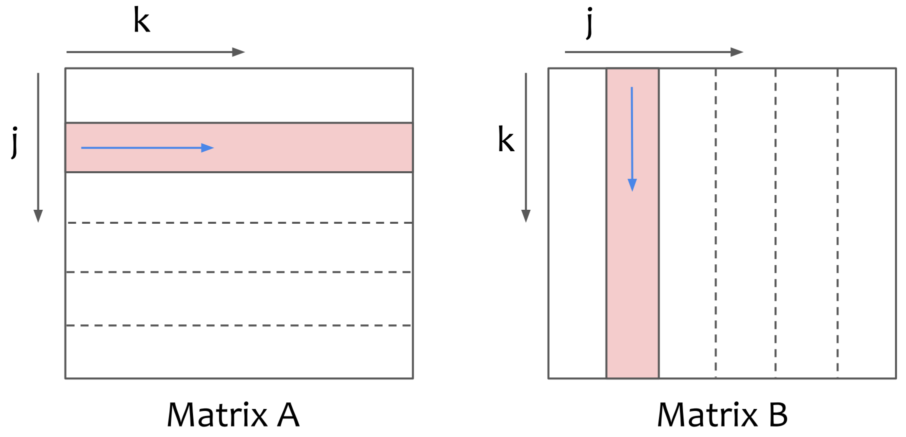
- Run independent from CPU cores
- Fetch data into caches in advance according to recent memory access patterns
    - effective for sequential accesses

Use case: cache locality

Naive matrix multiplication

- Matrix A, B are stored in row-major order
- Stride accesses to B elements over rows hurt cache locality
    - Cache misses occur frequently if matrix size is large enough

```
for (int i = 0; i < rows; ++i)
  for (int j = 0; j < columns; ++j)
   for (int k = 0; k < lengths; ++k)
    C[i][j] += A[i][k] * B[k][j];
```

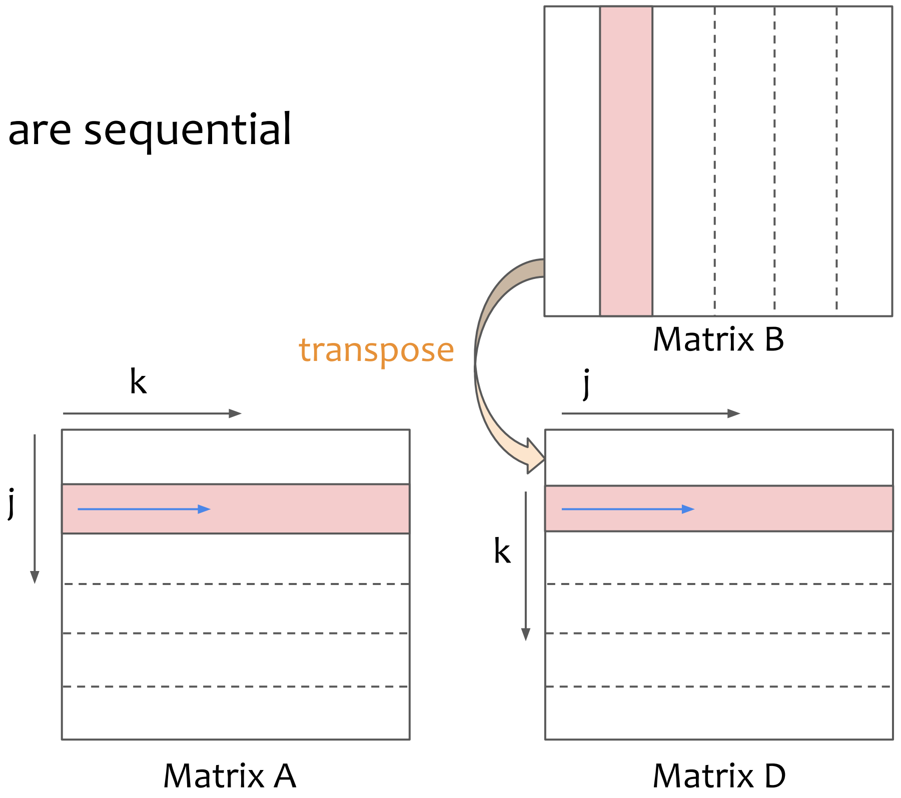


Use case: cache locality

Make the code cache-friendly

- Matrix D is the transpose of B
- Both accesses to A and D elements are sequential

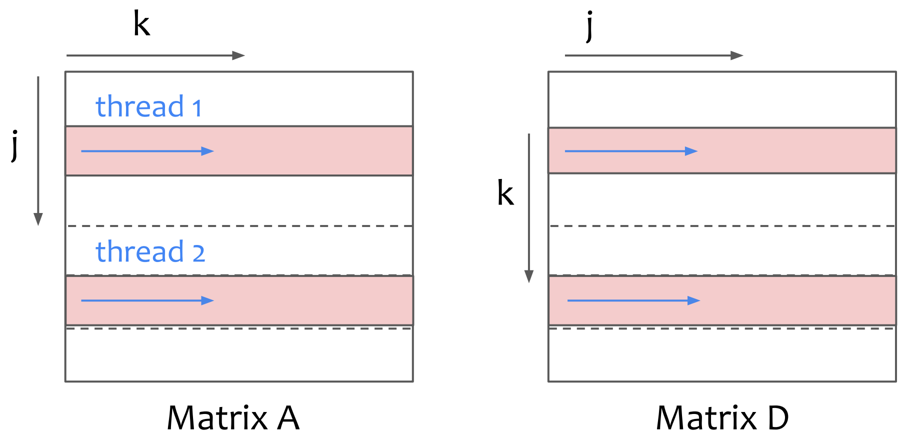
```
matrix_transpose(B, D);
for (int i = 0; i < rows; ++i)
  for (int j = 0; j < columns; ++j)
   for (int k = 0; k < lengths; ++k)
     C[i][j] += A[i][k] * D[k][j];
```



2. Multi-threading

Use multiple CPU cores to execute a single task (studied in Task 4)

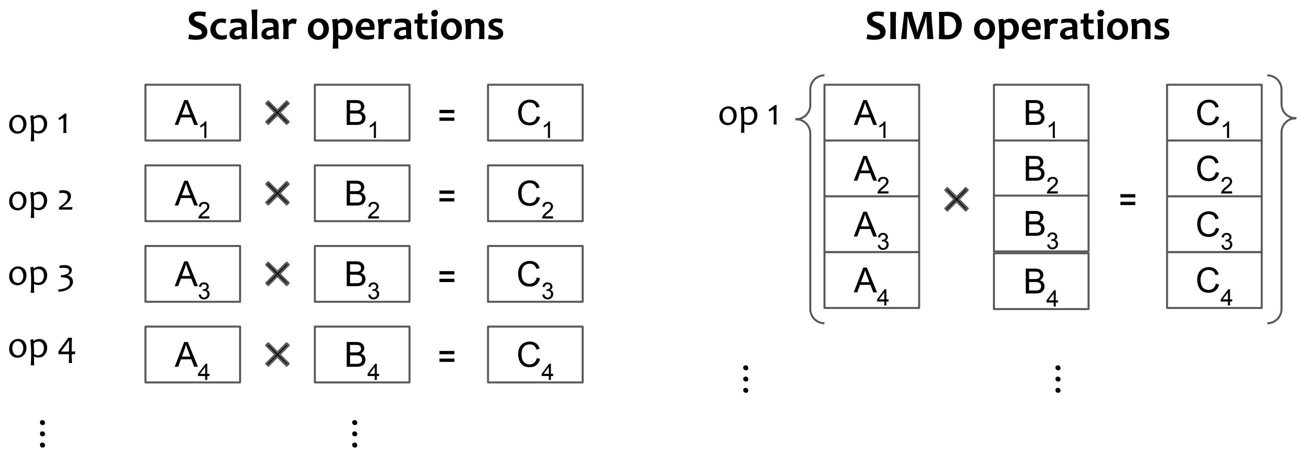
- Split data into small portions
  - avoid to hurt cache locality
- Create threads to compute each portion of data concurrently



3. SIMD instructions

Special instruction sets for Single Instruction Multiple Data (SIMD)

- Processing multiple data with a single instruction
  - SIMD registers (128-, 256-, 512-bit width) are integrated into CPU cores
    - Intel: Streaming SIMD Extensions (SSE), Advanced Vector Extensions (AVX)
    - ARM: Scalable Vector Extension (SVE)



SSE2 instruction sets

SSE2 supports 128-bit operations. See Intel Intrinsics Guide [1]

- Load/store inst.
  - __m128 _mm_load_ps (float const* mem_addr)
  - void _mm_store_ps (float* mem_addr, __m128 a)
- Arithmetic inst.
  - __m128 _mm_mul_ps (__m128 a, __m128 b)
  - __m128 _mm_add_ps (__m128 a, __m128 b)
- Conditional inst.
  - __m128 _mm_cmpeq_ps (__m128 a, __m128 b)
  - int _mm_movemask_ps (__m128 a)

[1] Intel Intrinsics Guide: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

Performance measurement

Hardware Counter

- Performance Monitoring Counters (PMCs) in CPU cores
- Monitor how many times a specific event occurred in a certain period of time
    - e.g., elapsed time (cycles), retired instructions, cache misses, CPU stalls

Tools

- **Linux system calls (e.g., clock_gettime)**
- **perf**
- Intel VTune
- AMD µProf

Performance measurement

- **int clock_gettime(clockid_t** clk_id**, struct timespec** *spec**)**
    - Get the current time of the clock in nanosecond time resolution
    - `clk_id` specifies a particular clock
        - `CLOCK_REALTIME` : system-wide real time clock. Changeable by configuration (e.g., NTP)
        - `CLOCK_MONOTONIC`: unchangeable monotonic time clock. Recommended

- **perf** [OPTIONS] **COMMAND** [ARGS]
    - Trace specific events through PMCs
    - `perf list` shows traceable events
    - `perf stat` runs a program and gather performance counter statistics
        - E.g., tracing LLC misses of all CPU cores during ls command:

```
$ sudo perf stat -a -e LLC-load-misses,LLC-store-misses ls
```

Assignment

Optimize the execution time of two simple programs

- Matrix multiplication
- Mandelbrot set

Tips

- Make the code cache-friendly
- Parallelize computations with SIMD instructions, multi-threading
  - Up to four CPU cores are available
  - Please only use SSE/SSE2 instructions