Лаб 4 C++11 MULTITHREADING алгоритмууд

зорилго: Лекцийн хичээл дээр үзсэн ойлголтуудаа батагана, Статик тархалтанд суурилсан хуваарилалт зохион байгуулах алгоритм бичих болон mutex, condition varaibles хэрэглэж сурах, дадлагажих

1. C = A + B matrix (block, cyclic, block cyclic)

```
#include "../include/hpc_helpers.hpp"
#include <iostream>
#include <cstdint>
#include <vector>
#include <thread>
#include <inttypes.h>
#include <random>

template <
    typename value_t,
    typename index_t>
void sequential(
    std::vector<value_t>& A,
    std::vector<value_t>& B,
    std::vector<value_t>& C,
    index_t m,
    index_t n) {

    for (index_t row = 0; row < m; row++) {
        value_t accum = value_t(0);
        for (index_t col = 0; col < n; col++)
            C[row*n+col] += A[row*n+col]+B[row*n+col];
    }
}

template <
    typename value_t,
    typename index_t>
void cyclic_parallel(
    std::vector<value_t>& A,
    std::vector<value_t>& B,
    std::vector<value_t>& C,
    index_t m,              // number of rows
    index_t n,              // number of cols
    index_t num_threads) { // number of threads p

    auto cyclic = [&] (const index_t& id) -> void {

        for (index_t row = id; row < m; row += num_threads) {
            // value_t accum = value_t(0);
```

```
            // value_t accum = value_t(0);
         for (index_t col = 0; col < n; col++)
                C[row*n+col] += A[row*n+col]+B[row*n+col];
            // C[row] = accum;
        }
    };

    std::vector<std::thread> threads;

    for (index_t id = 0; id < num_threads; id++)
        threads.emplace_back(cyclic, id);

    for (auto& thread : threads)
        thread.join();
}

template <
    typename value_t,
    typename index_t>
void block_parallel(
    std::vector<value_t>& A,
    std::vector<value_t>& B,
    std::vector<value_t>& C,
    index_t m,
    index_t n,
    index_t num_threads) {

    // this function is called by the threads
    auto block = [&] (const index_t& id) -> void {
        //        ^-- capture whole scope by reference

        // compute chunk size, lower and upper task id
        const index_t chunk = SDIV(m, num_threads);
        const index_t lower = id*chunk;
        const index_t upper = std::min(lower+chunk, m);

        // only computes rows between lower and upper
        for (index_t row = lower; row < upper; row++) {
            // value_t accum = value_t(0);
            for (index_t col = 0; col < n; col++)
                // accum += A[row*n+col]+B[col];
                C[row*n+col] = A[row*n+col] + B[row*n+col];
            // b[row] = accum;
        }
    };

    // business as usual
    std::vector<std::thread> threads;

    for (index_t id = 0; id < num_threads; id++)
        threads.emplace_back(block, id);

    for (auto& thread : threads)
        thread.join();
}
```

```
template <
    typename value_t,
    typename index_t>
void block_cyclic_parallel(
    std::vector<value_t>& A,
    std::vector<value_t>& B,
    std::vector<value_t>& C,
    index_t m,
    index_t n,
    index_t num_threads,
    index_t chunk_size=64/sizeof(value_t)) {


    // this  function  is  called  by the  threads
    auto block_cyclic = [&] (const index_t& id) -> void {

        // precomupute the stride
  const index_t stride = num_threads*chunk_size;
  const index_t offset = id*chunk_size;

        // for each block of size chunk_size in cyclic order
        for (index_t lower = offset; lower < m; lower += stride) {

            // compute the upper border of the block
            const index_t upper = std::min(lower+chunk_size, m);

      // for each row in the block
            for (index_t row = lower; row < upper; row++) {

    // accumulate the contributions
    // value_t accum = value_t(0);
    for (index_t col = 0; col < n; col++)
                // accum += A[row*n+col]+x[col];
            C[row*n+col] = A[row*n+col] + B[row*n+col];
            // b[row] = accum;
            }
      }
    };

    // business as usual
    std::vector<std::thread> threads;

    for (index_t id = 0; id < num_threads; id++)
        threads.emplace_back(block_cyclic, id);

    for (auto& thread : threads)
        thread.join();
}


int main(int argc, char* argv[]) {

    const uint64_t n = 1UL << 10; // 1024
    const uint64_t m = 1UL << 10;
```

```
    const uint64_t p = 8;

    // compile : g++ -O2 -std=c++14 -pthread lab4_1.cpp -o out

    TIMERSTART(alloc)
    std::vector<no_init_t<uint64_t>> A(m*n);
    std::vector<no_init_t<uint64_t>> B(m*n);
    std::vector<no_init_t<uint64_t>> C(m*n);
    TIMERSTOP(alloc)


    TIMERSTART(init)
    for (uint64_t i = 0; i < m*n; i++)
        A[i] = rand() % 100;   // 0 - 99 numbers
    for (uint64_t i = 0; i < m*n; i++)
        B[i] = rand() % 100;   // 0 - 99 numbers
    TIMERSTOP(init)

    // printf("A matrix: \n");
    // for(uint64_t i = 0; i<5; i++){
    //  std::cout <<  A[i] <<" " << std::endl;
    // }
    // printf("\n");

    // printf("B matrix: \n");
    // for(uint64_t i = 0; i<5; i++){
    //  std::cout <<  B[i] <<" " << std::endl;
    // }
    // printf("\n");

    // TIMERSTART(cyclic_parallel)
    // cyclic_parallel(A, B, C, m, n, p);
    // TIMERSTOP(cyclic_parallel)

    // TIMERSTART(block_parallel)
    // block_parallel(A,B,C,m,n,p);
    // TIMERSTOP(block_parallel)

    TIMERSTART(block_cyclic_parallel)
    block_cyclic_parallel(A,B,C,m,n,p);
    TIMERSTOP(block_cyclic_parallel)

    // printf("C matrix: \n");
    // for(uint64_t i = 0; i<5; i++){
    //     std::cout <<  C[i] <<" " << std::endl;
    // }
    // printf("\n");
}
```

үр дүн: N = 1024 авч үзэв.

```
# elapsed time (alloc): 4.0291e-05s
# elapsed time (init): 0.0439883s
# elapsed time (cyclic_parallel): 0.00578299s
(base) ┌─[jakitcs@jakitcs-VPCEH3C0E]─[~/Desktop/parallelprogrammingbook-master/chapter4/Lab4]
└──╼ $g++ -O2 -std=c++14 -pthread lab4_1.cpp -o out
(base) ┌─[jakitcs@jakitcs-VPCEH3C0E]─[~/Desktop/parallelprogrammingbook-master/chapter4/Lab4]
└──╼ $./out
# elapsed time (alloc): 4.1764e-05s
# elapsed time (init): 0.0454456s
# elapsed time (block_parallel): 0.00529414s
(base) ┌─[jakitcs@jakitcs-VPCEH3C0E]─[~/Desktop/parallelprogrammingbook-master/chapter4/Lab4]
└──╼ $g++ -O2 -std=c++14 -pthread lab4_1.cpp -o out
(base) ┌─[jakitcs@jakitcs-VPCEH3C0E]─[~/Desktop/parallelprogrammingbook-master/chapter4/Lab4]
└──╼ $./out
# elapsed time (alloc): 2.2471e-05s
# elapsed time (init): 0.0525985s
# elapsed time (block_cyclic_parallel): 0.00525695s
(base) ┌─[jakitcs@jakitcs-VPCEH3C0E]─[~/Desktop/parallelprogrammingbook-master/chapter4/Lab4]
└──╼ $
```

Дүгнэлт: Функц бүрт lambda функц ашигласан.

2. mutex

```
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <chrono>

int main() {

  std::mutex m;
  std::condition_variable c;
  bool done = false;

  auto child = [&]( ) -> void { // lambda function

      {
        // locked scope
      std::unique_lock<std::mutex> unique_lock(m);
            do {
                c.wait(unique_lock);
            } while (!done);

      }

    std::cout << "child" << std::endl;

  };

  std::thread thread(child);
  thread.detach();
  std::cout << "parent" << std::endl;
```

```
    {
      std::lock_guard<std::mutex> lock_guard(m);
      done = true;
    }

    c.notify_one();
    // thread.join();
  }
```