Parallel Programming course. Repository structure

Obolenskiy Arseniy, Nesterov Alexander

Nizhny Novgorod State University

November 9, 2024

Contents

Parallel programming technologies

- **MPI**
- OpenMP
- TBB
- std::thread

Step-by-step guide to build the project

- Download all submodules
- Set up your environment
- Build the project with CMake

Download all submodules & Set up your environment

Listing 1: Git submodules

```
1  git submodule update --init --recursive
2
```

Listing 2: Download MPI

```
1  // Windows
2  msmpisdk.msi & msmpisetup.exe
3  // Linux (gcc and clang)
4  sudo apt install -y mpich openmpi-bin libopenmpi-dev
5  // MacOS (apple clang)
6  brew install open-mpi
7
```

Build the project with help CMake

Listing 3: Configure the build

```
cmake -S <source code/path to CMakeLists.txt> \
      -B <build directory> \
      -D USE_SEQ=ON \
      -D USE_MPI=ON \
      -D USE_FUNC_TESTS=ON \
      -D USE_PERF_TESTS=ON \
      -D CMAKE_BUILD_TYPE=Release ..

```

Listing 4: Build the project

```
cmake --build <build directory> \
      --config Release \
      --parallel

```

Directories

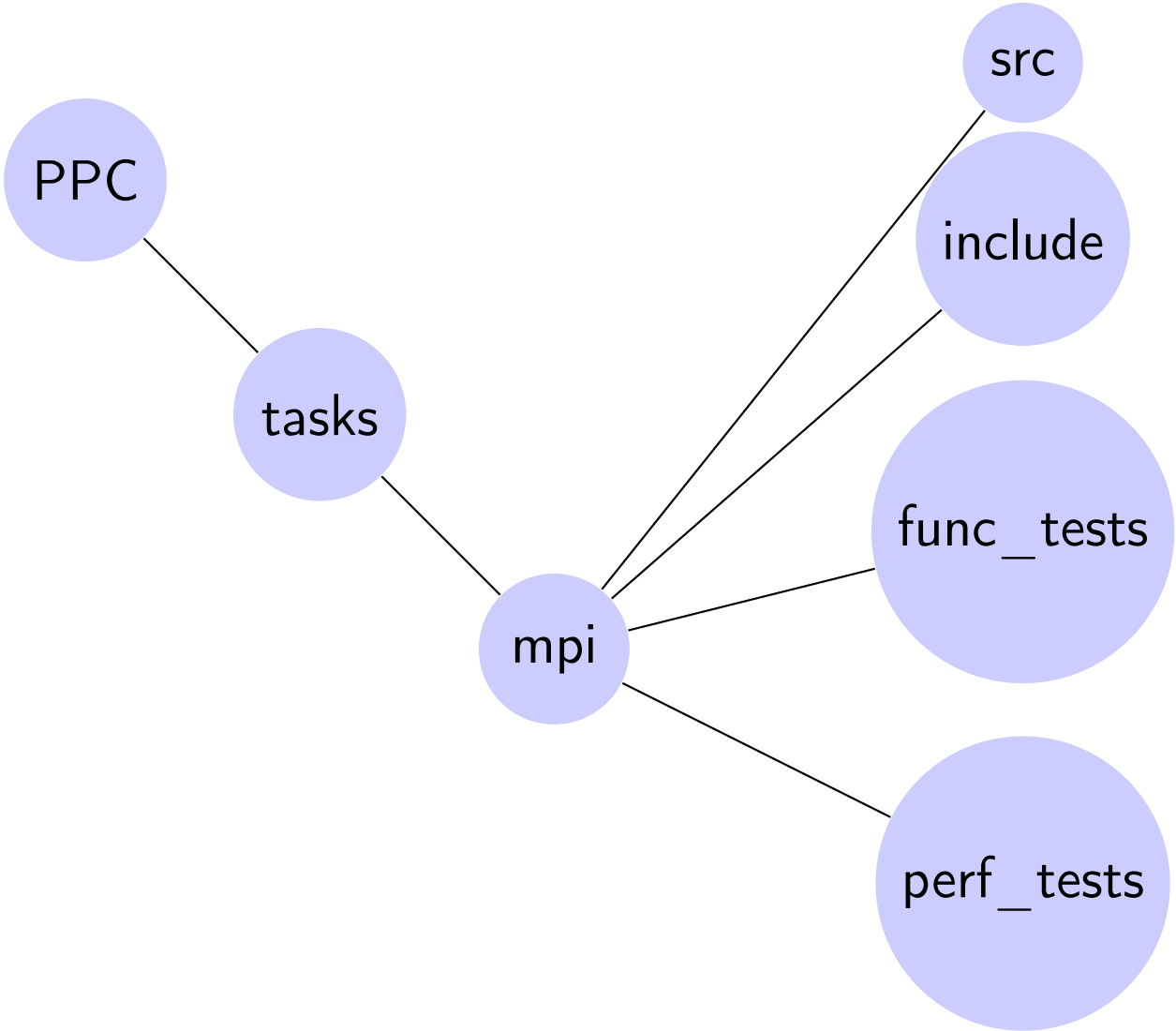
Directory	What is it?
.github/workflows	CI
1stsamples	Simple examples
3rdparty	Auxiliary libraries
cmake	CMake scripts
modules	API source code
scripts	Auxiliary scripts
tasks	Students tasks

Table: Root directories

Directory	What is it?
mpi	MPI
omp	OpenMP
seq	Sequential
stl	std:thread
tbb	Threading Building Blocks

Table: Tasks directories

Directories



Task's prototype

Listing 5: include/ops_seq.hpp

```
1   class TestTaskSequential : public ppc::core::Task {
2    public:
3     explicit TestTaskSequential(std::shared_ptr<ppc::core::TaskData> taskData_) :
      Task(std::move(taskData_)) {}
4     bool pre_processing() override;
5     bool validation() override;
6     bool run() override;
7     bool post_processing() override;
8
9    private:
10    int input_{}, res{};
11  };
12
```

Task's source code

Listing 6: src/ops_seq.cpp | pre_processing

```
1  bool nesterov_a_test_task_seq::TestTaskSequential::pre_processing() {
2    internal_order_test();
3    // Init value for input and output
4    input_ = reinterpret_cast<int*>(taskData->inputs[0])[0];
5    res = 0;
6    return true;
7  }
8
```

Listing 7: src/ops_seq.cpp | validation

```
1  bool nesterov_a_test_task_seq::TestTaskSequential::validation() {
2    internal_order_test();
3    // Check count elements of output
4    return taskData->inputs_count[0] == 1 && taskData->outputs_count[0] == 1;
5  }
6
```

Task's source code

Listing 8: src/ops_seq.cpp | run

```
1  bool nesterov_a_test_task_seq::TestTaskSequential::run() {
2    internal_order_test();
3    for (int i = 0; i < input_; i++) {
4      res++;
5    }
6    return true;
7  }
8
```

Listing 9: src/ops_seq.cpp | post_processing

```
1  bool nesterov_a_test_task_seq::TestTaskSequential::post_processing() {
2    internal_order_test();
3    reinterpret_cast<int*>(taskData->outputs[0])[0] = res;
4    return true;
5  }
6
```

Task's functional tests

Listing 10: func_tests/main.cpp

```
1   TEST(Sequential, Test_Sum_10) {
2     const int count = 10;
3
4     // Create data
5     std::vector<int> in(1, count);
6     std::vector<int> out(1, 0);
7
8     // Create TaskData
9     std::shared_ptr<ppc::core::TaskData> taskDataSeq = std::make_shared<ppc::core
      ::TaskData>();
10    taskDataSeq->inputs.emplace_back(reinterpret_cast<uint8_t *>(in.data()));
11    taskDataSeq->inputs_count.emplace_back(in.size());
12    taskDataSeq->outputs.emplace_back(reinterpret_cast<uint8_t *>(out.data()));
13    taskDataSeq->outputs_count.emplace_back(out.size());
14
15    // Create Task
16    nesterov_a_test_task_seq::TestTaskSequential testTaskSequential(taskDataSeq);
17    ASSERT_EQ(testTaskSequential.validation(), true);
18    testTaskSequential.pre_processing();
19    testTaskSequential.run();
20    testTaskSequential.post_processing();
21    ASSERT_EQ(count, out[0]);
22  }
23
```

TaskData structure

Listing 11: TaskData

```
1  struct TaskData {
2    std::vector<uint8_t *> inputs;
3    std::vector<std::uint32_t> inputs_count;
4    std::vector<uint8_t *> outputs;
5    std::vector<std::uint32_t> outputs_count;
6    enum StateOfTesting { FUNC, PERF } state_of_testing;
7  };
8
```

Listing 12: Functions order

```
1  std::vector<std::string> right_functions_order  =
2      {"validation",
3       "pre_processing",
4       "run",
5       "post_processing"};
6
```

Task's performance tests - part 1

Listing 13: perf_tests/main.cpp

```
1   TEST(sequential_example_perf_test, test_pipeline_run) {
2     const int count = 100;
3
4     // Create data
5     std::vector<int> in(1, count);
6     std::vector<int> out(1, 0);
7
8     // Create TaskData
9     std::shared_ptr<ppc::core::TaskData> taskDataSeq = std::make_shared<ppc::core
      ::TaskData>();
10    taskDataSeq->inputs.emplace_back(reinterpret_cast<uint8_t *>(in.data()));
11    taskDataSeq->inputs_count.emplace_back(in.size());
12    taskDataSeq->outputs.emplace_back(reinterpret_cast<uint8_t *>(out.data()));
13    taskDataSeq->outputs_count.emplace_back(out.size());
14
15    // Create Task
16    auto testTaskSequential = std::make_shared<nesterov_a_test_task_seq::
      TestTaskSequential>(taskDataSeq);
17
```

Task's performance tests - part 2

Listing 14: perf_tests/main.cpp

```
1       // Create Perf attributes
2       auto perfAttr = std::make_shared<ppc::core::PerfAttr>();
3       perfAttr->num_running = 10;
4       const auto t0 = std::chrono::high_resolution_clock::now();
5       perfAttr->current_timer = [&] {
6         auto current_time_point = std::chrono::high_resolution_clock::now();
7         auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(
      current_time_point - t0).count();
8         return static_cast<double>(duration) * 1e-9;
9       };
10
11      // Create and init perf results
12      auto perfResults = std::make_shared<ppc::core::PerfResults>();
13
14      // Create Perf analyzer
15      auto perfAnalyzer = std::make_shared<ppc::core::Perf>(testTaskSequential);
16      perfAnalyzer->pipeline_run(perfAttr, perfResults);
17      ppc::core::Perf::print_perf_statistic(perfResults);
18      ASSERT_EQ(count, out[0]);
19    }
20
```

Practice

Practice

References

- PPC Repository
  https://github.com/learning-process/parallel_programming_course