**Task System Implementation for Parallel Computing PART\_B**

**Introduction**

This project is an assignment for a parallel computing course, aimed at implementing and comparing different task systems for executing tasks in parallel. The implemented task systems include:

- Serial Task System: Executes tasks sequentially in a single thread.

- Parallel Task System with Spawning Threads: Spawns new threads for each bulk task launch.

- Parallel Task System with Spinning Thread Pool: Uses a thread pool that spins (busy-waits) when idle.

- Parallel Task System with Sleeping Thread Pool: Uses a thread pool that sleeps when idle, optimized for efficiency.

In Part B of the assignment, the task system is extended to support asynchronous task launches with dependencies, forming a task graph that respects inter-task dependencies.

**Prerequisites**

To build and run this project, ensure you have the following:

- A C++ compiler supporting C++11 (e.g., `g++` version 4.8 or later).

- The `make` utility for building the project.

- A Unix-like environment (e.g., Linux, macOS, or WSL on Windows).

**Building the Project**

The project uses a `Makefile` to compile the source code. Follow these steps to build it:

1. Navigate to the project directory (e.g., `part\_b/):

2. Compile the code using make:

This generates the runtasks executable in the current directory.

**Running the Project**

The runtasks executable allows you to run tests and evaluate the task systems. Here are some common usage examples:

**Running the Full Test Harness**

To run multiple tests using the provided Python script:

python3 ../tests/run\_test\_harness.py -n <num\_threads> -t <test1> <test2> ...

Example:

python3 ../tests/run\_test\_harness.py -n 16 -t simple\_test\_async super\_super\_light\_async

**Testing**

The project includes tests to verify correctness and performance, defined in `tests/tests.h` and listed in tests/main.cpp.

**Running Tests**

- Run individual tests with `runtasks` as shown above.

- Use the Python test harness for a full suite and comparison with a reference implementation.

**Interpreting Results**

- Correctness: A passing test indicates correct output. Failures suggest issues in task execution or dependencies.

- Performance: The test harness reports ratios against a reference. A ratio ≤ 1.2 (within 20%) is typically acceptable.

**Extending the Project**

To create custom tests:

1. **Define a New Test** in tests/tests.h:

.cpp

class MyCustomTask : public IRunnable {

public:

void runTask(int task\_id, int num\_total\_tasks) override {

// Custom task logic

}

};

void myCustomTest(ITaskSystem\* t, int num\_threads) {

// Test setup and execution

}

2. Register the Test in tests/main.cpp:

.cpp

const char\* test\_names[] = { ..., "my\_custom\_test" };

TestFunc tests[] = { ..., myCustomTest };

int n\_tests = sizeof(tests) / sizeof(TestFunc);

3. Run your test using runtasks or the test harness.

**Contributing**

- Optimizing the sleeping thread pool.

- Adding new synchronization methods.

- Supporting more complex task graphs.

**Running Example:**

