Assignment Project Exam Help

15-418/618, Spring 2019

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

Assigned:	Wed., March 6
Due:	Wed., Mar. 27, 11:59 pm
Last day to handin:	Sat., Mar. 30

1 Overview

Before you begin, please take the time to review the course policy on academic integrity at:

http://www.cs.cmu.edu/~418/academicintegrity.html

Download the Assignment 4 starter code from the course Github using:

```
linux> git clone https://github.com/cmu15418/asst4-s19.git
```

In order to add support for MPI compilation for the GHC or unix.andrew machines, do one of the following:

- Add the following line to your file `~/.cshrc`:

  ```
  setenv PATH $PATH\:/usr/lib64/openmpi/bin
  ```

- Add the following line to your file `~/.bashrc`:

```
export PATH=$PATH:/usr/lib64/openmpi/bin
```

Assignment Objectives

In this assignment, you will explore the use of the MPI library to implement a program consisting of a number of independent processes that communicate and coordinate with one another via message passing. The application is typical of the *bulk synchronous* execution model seen in many scientific applications. Although the application is the same as you had in Assignment 3, you will find that your implementation is very different. OpenMP provides a data-parallel programming model, where the program consists of sequence of steps, each of which performs many operations in parallel. By contrast, an MPI program describes the behavior of an autonomous process that periodically communicates with other processes running the same code.

Assignment Project Exam Help

Machines

https://eduassistpro.github.io/

The MPI (for "Message-Pa s that can run on collections of machine ch has the advantage that it can scale to ve run on single, multicore processors, but using message passing, rather tha communication or synchronization.

Add WeChat edu_assist_pro

You can test and evaluate your programs on any multicore processor, inc unix.andrew machines. For performance evaluation, you will run your programs on the Latedays cluster.

Resources

There is a lot of information online about MPI. Some resources we have found useful include:

- General MPI Tutorial
- Longer MPI Tutorial from Lawrence Livermore National Laboratories
- Official documentation on OpenMPI v1.6, the version that runs on the Latedays machines

2 Application

Dr. Roland Dent, Director of the world-famous GraphRats project was quite excited to find that million-rat simulations are possible using a well-optimized simulator running on a multicore processor. But, he dreams of more. "There are billions of rats in the world. Shouldn't we be able to simulate billions of rats?" You have convinced him that such large simulations would require much more computing power, beyond what shared-memory systems can provide. It might be possible on a modern supercomputer, with ten thousand or more nodes that communicate by message passing.

As a feasibility study, you propose implementing an MPI version of the GraphRats simulator running on a single, multicore machine, but using only message passing to communicate and coordinate among the machine's cores. Your idea is to partition the graph into separate *zones*, mapping each zone onto a separate process (and relying on the OS to map each process onto a separate core.) Each process will keep track of the rats within its assigned zone, computing the new states of all of these rats. It will communicate with processes holding nodes adjacent to ones in its zone, both to share the node states and weights along the boundaries, and to pass along rats as they move from one zone to another.

Fortunately, the graphs used in Assignment 3 have natural partitionings into zones. Each consists of a collection of identical regions connected only by grid edges. Furthermore, the number of regions for each of these graphs is divisible by 12. We can therefore partition each graph into P zones, as long as $P \in \{1, 2, 3, 4, 6, 12\}$, with each zone containing the same number of regions.

Model Parameters

Assignment Project Exam Help

As before, each graph consist s indicating which nodes are adjacent. A k nodes. Some of the nodes are designated as "hubs," wi nded to specify a partitioning of each graph into 12 zones. The code for loading the graphs into memory has been extended to then coalesce these zones into P zones, where e.

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

Other aspects of the program (rats, reward functions) are the same as in Assign been simplified to support only batch update mode. The starter code can be com different simulators: `crun-seq`, suitable for sequential execution, and `crun-mpi`, providing the starting framework for an MPI-based parallel simulator. Compiler-directives based on the compile-time constant `MPI` designate the differences between the two.

The MPI-based simulator does the following, when invoked to run with P processes:

1. The master process (Process 0), reads a copy of the graph file.
2. It broadcasts the complete graph data structure to the other $P - 1$ processes.
3. Each process creates a set of data structures to represent its assigned zone.
4. The master process reads a copy of the rat file indicating the starting nodes.
5. The master process runs the entire simulation in sequential mode.

Your job then, is first to have the master distribute the rats, according to the zones containing their starting nodes. Then the processes should simulate their zones and exchange rats and node information with each other. Periodically, they will be directed to provide their node counts to the master process, so that it can supply this information as the program output.

The simulator has a subset of options seen in Assignment 3:

```
linux> ./crun-seq -h
Usage: ./crun-seq -g GFILE -r RFILE [-n STEPS] [-s SEED] [-u (r|b|s)] [-q] [-i INT]
   -h        Print this message
   -g GFILE  Graph file
   -r RFILE  Initial rat position file
   -n STEPS  Number of simulation steps
   -s SEED   Initial RNG seed
   -q        Operate in quiet mode.  Do not generate simulation results
   -i INT    Display update interval
```

As before, you can use the Python program `grun.py` to visualize the simulation results.

To run a program under MPI, you use the program `mpirun`. A typical invocation could be:

```
linux> mpirun -np 6 ./crun-mpi -g data/g-t180x180.gph -r data/r-180x180-t32.rats -n 5 -q
```

This has the simulator run with $P = 6$.

Assignment Project Exam Help

3 Test Programs and Performance Evaluation

The provided program https://eduassistpro.github.io/ a number of MPI processes rather than OMP t

```
linux> ./regress.py -h
Usage: ./regress.py [-h] [-c] [-p PROCS]
    -h       Print this message
    -c       Clear expected result cache
    -p P     Specify number of MPI processes
       If > 1, will run crun-mpi.  Else will run crun-seq
```

Add WeChat edu_assist_pro

The provided program `benchmark.py` has the following options:

```
linux> ./benchmark.py -h
Usage: ./benchmark.py [-h] [-k K] [-b BENCHLIST] [-n NSTEP] [-p P] [-r RUNS] [-i ID] [-f OUTFILE]
    -h            Print this message
    -k            Specify graph dimension
    -b BENCHLIST  Specify which benchmark(s) to perform as substring of 'ABCDEF'
    -n NSTEP      Specify number of steps to run simulations
    -p P          Specify number of MPI processes
       If > 1, will run crun-mpi.  Else will run crun-seq
    -r RUNS       Set number of times each benchmark is run
    -i ID         Specify unique ID for distinguishing check files
    -f OUTFILE    Create output file recording measurements
         If file name contains field of form XX..X,
         will replace with ID having that many digits
```

The intention is that you run this program on either a GHC, a unix.andrew, or a Latedays machine. In all of these cases, it will automatically invoke `mpirun` with a set of arguments that specify the use of *processor affinity*, a specific way to map processes onto cores. Running on a GHC machine requires setting the number of processes to 1, 2, 3, 4, or 6. The other supported machines can also handle 12 processes. By default, the program will run each simulation three times and take the minimum of their execution times. This helps make the timings more reliable. You can change this with the command-line option '`-r`.'

The guidelines for using the Latedays machines are the same as for Assignment 3. The provided program `submitjob.py` is used to generate and submit the control files to the job queue.

The performance will be evaluated on the same combinations of graphs and initial rat positions as in Assignment 3. Each run will be benchmarked against the provided program: either `crun-soln-ghc` (GHC or unix.andrew) or `crun-soln-latedays` (Latedays). Performance points are computed as they were in Assignment 3: each benchmark will count up to 15 points, for a maximum total of 90 points.

4 Some Advice

Important Requirements

The following are some aspects of the assignment that you should keep in mind:

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

- You may only use MP PI processes. You cannot us m that could ultimatel
- Although performance will be measured with just 12 processes n on P processes, as long as 12 is divisible by P.
- You are free to add other header and code files and to modify the make C++ (or Fortran) if you like. The only code you cannot modify is in the file `rutil.c`. You must use the provided version of the function `imbalance`, which computes the imbalance factor β, and you must not attempt to reduce the number of calls to this function.
- You can use any kind of code, including calls to standard libraries, as long as it is *platform independent*. You *may not* use any constructs that make particular assumptions about the machine instruction set, such as embedded assembly or calls to an intrinsics library. (The exception to this being the code in `cycletimer.c`.)
- You may not include code generated by other parallel-programming frameworks, such as ISPC, OpenMP, PThreads, etc..
- Although your simulator will only be tested on graphs of up to size 180×180, and P up to 12, you should write your code to scale up to graphs of arbitrary size, arbitrary rat counts, and an arbitrary number of processes.

Useful Parts of MPI

The MPI standard is large and complex. You only need to use a core subset of its features. Ideas that are especially useful for this assignment include:

- Using synchronous and asynchronous send and receive constructs for point-to-point communications. Asynchronous communication is preferred, because it allows the processes to operate in a more loosely coupled manner. When exchanging data with adjacent zones, it works well to have a process first initiate all of its send operations, then perform the receives, and then wait for the sends to complete.

- Using the probe operation to determine the size of an incoming message. This is useful when sending variable length buffers of rats between processes.

- Using broadcast to send copies of the initial rat positions from Process 0 (the master) to the others at the beginning of the simulation.

What is Provided

- You will find that the modifications you made to the starter code for Assignment 3 are not very useful here. You'd do better to work from the new starter code.

- The provided code stores a complete representation of the graph for each process. This uses more space than is necessa es, and rats. It also will not ha much memory actually ge

- The provided code has each process construct data structures re tored as fields in the graph_t structure (declared in file cr ted order.
  - Array local_node_list is a list of the nodes in the z local_node_count.
  - Array export_node_list is an array of P lists, where list j consists of the nodes in this zone that have edges to nodes in zone j. Its length is given by the field export_node_count[j].
  - Array import_node_list is an array of P lists, where list j consists of the nodes in zone j that have edges to nodes in this zone. Its length is given by the field import_node_count[j]. Given the symmetry of the graph, you can assume that the contents of export_node_list[j] for process i is identical to those of import_node_list[i] for process j.

Assignment Project Exam Help
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

What You Need to Do

- You will find comments in some of the .h and .c files with the header "TODO." These indicate some of the key places you will need to add or modify the existing code.

- You will need to allocate space to store information about the rats in each zone, as well as the buffers you use for communication via MPI. Generally, it is best to allocate these at the beginning of the program. Some you can allocate according to the maximum required size. Others you may want to allocate smaller amounts and then grow dynamically (via realloc) as needed.

- You will need to understand the processing steps in the function `do_batch` (file `sim.c`) and adapt them for use on a single zone. This will require several rounds of exchanging data with adjacent zones: rats, node counts, and node weights.
- When sending a rat to a new zone, you must also send along its associated seed for random number generation.
- You will need to implement the capability to have every process send its copy of the node counts to Process 0, and for Process 0 to collect these counts from other processes. These should be implemented as functions `send_node_state` and `gather_node_state`, respectively (file `simutil.c`.)

How to Optimize the Program

You will find that you need to represent different forms of sets for this assignment, e.g., the set of all rats within a particular zone. There are several common ways to do this:

Assignment Project Exam Help

- As a *bit vector*, where bit position i is set to 1 if i is in the set and to 0 if it is not. Although it is possible to pack multiple bits i `unsigned char` and just use one bit per b

https://eduassistpro.github.io/

- As a *list*, consistin
- As a hash table. These can be problematic for very large sets due to po dictable branches.

Add WeChat edu_assist_pro

Other Tips

- The data files include two very small graphs: one of size 2×2, and the other of size 3×3, along with associated rat position files. The former can be run with $P \in \{1, 2, 4\}$, while the latter can be run with $P \in \{1, 3, 9\}$. These can be useful when doing detailed debugging.
- You can do direct comparisons of two versions of your code by renaming one of the executables to be either `crun-soln-ghc` or `crun-soln-latedays` (depending on which class of machine you're using). Be sure to keep the original copy of this program, of course.

5 Your Report (20 points)

Your report should provide a concise, but complete description of the thought process that went into designing your program and how it evolved over time based on your experiments.

Your report should include a detailed discussion of the design and rationale behind your approach to parallelizing the algorithm. Specifically try to address the following questions:

1. What sequence of computations and communications is performed for each batch?

2. How did you maximize the decoupling of processes to avoid waiting for messages from each other.
3. How successful were you in getting speedup in your program? (This should be backed by experimental measurements.)
4. How did the performance scale as you went from 1 to 4 to 6 to 12 processes?
5. How did the graph structure and the initial rat positions affect your ability to exploit parallelism?
6. How did your program perform compared to your solution for Assignment 3? To what do you attribute the differences?
7. Were there any techniques that you tried but found ineffective?

6 Hand-in Instructions

Assignment Project Exam Help

You will submit your code via Autolab and your report via Gradescope. For the code, you will be submitting your entire directory tree.

https://eduassistpro.github.io/

1. Your code
   (a) **If you are working with a partner, form a group on Auto** ore submitting your assignment. One submission per group is sufficient.
   (b) Make sure all of your code is compilable and runnable.
      i. We should be able to simply run `make` in the `code` subdirectory and have everything compile.
      ii. We should be able to replace your versions of all of the Python code, as well as the file `rutil.c` with the original versions and then perform regression testing and benchmarking.
   (c) Remove all nonessential files, especially output images from your directory.
   (d) Run the command "`make handin.tar`." This will run "`make clean`" and then create an archive of your entire directory tree.
   (e) Submit the file `handin.tar` to Autolab.

Add WeChat edu_assist_pro

2. Your report
   (a) Please upload your report in PDF format to Gradescope, with one submission per team. After submitting, you will be able to add your teammate using the *add group members* button on the top right of your submission.