CSc 422/522 — Parallel Programming Project

Programs due Tuesday, April 3 (by midnight)
Final programs and reports due April 10 (in class)

In this project you will learn about grid computations, develop efficient parallel programs, conduct timing experiments to analyze the behavior of your programs, and write a report presenting your results. The project is worth 60 points for undergraduates and 75 points for honors and graduate students. You may work on your own or with one other classmate.*

There are four parts to this project as described below: writing programs, conducting timing experiments, conducting other experiments, and writing a report. Graduate and honors students are to do all parts. Undergraduates can *either* (1) write just the first two programs and do the other three parts for those two programs, or (2) omit the other experiments—namely, write four programs, do timing experiments, and write a report.

Programs

The starting point for this project is the following four programs. You may write the programs in MPD or in C with the Pthreads library. Develop your programs on Lectura or at home, but run experiments on Parallel.

1. a sequential red/black Gauss-Seidel program
2. a parallel red/black Gauss-Seidel program
3. a sequential multigrid program
4. a parallel multigrid program

Gauss-Seidel and red/black programs are described in Section 11.1.5 of the text and a parallel program is shown in Figure 11.6. Multigrid methods are described in Section 11.1.6, but I do not give actual code. It is up to you to figure out the details for the sequential program and to parallelize that program.

Your first task is to write *efficient* programs. See the discussion on pages 539-40 of the text for some of the kinds of programming "tricks" you might want to employ to make your programs as fast as you can.

Initialize the boundaries of all grids to 1.0 and the interiors to 0.0. This will make it easy for you to check the correctness of your programs and the quality of the results for different algorithms.

For the parallel programs, divide the grid into strips so as to balance the computational load in a reasonable way. This includes all the grids in the multigrid programs. Use one worker process per strip. Implement an efficient *dissemination* barrier and use it when you need barrier synchronization. *Make sure your barrier is correct!* (You might first want to implement a simple barrier using semaphores or monitors.)

For the multigrid programs, use a four-level V cycle as illustrated in Figure 11.8. Preallocate separate matrices for each level. Use the restriction and interpolation operators described on

*If a two-person team consists of an undergraduate and honors student or an undergraduate and a graduate, then you must do the larger project. However, the undergraduate will get a maximum of 60 points.

pages 550-51 of the text. Use red/black Gauss-Seidel for the iterations on each level. Use exactly four iterations on each of the coarser grids, but use the command-line argument `numIters` (see below) for the number of iterations on the finest grid.

Input and Output

Your programs should have three command-line arguments in the following order:

> `gridSize` — the grid size, *not* including boundaries
> `numIters` — the number of iterations to use
> `numWorkers` — the number of worker processes (for the parallel programs)

Assume that all grids are square. For the multigrid programs, the value of `gridSize` is the size of the *coarsest* (smallest) grid. The size of the next larger grid should then be `2*gridSize + 1`, the next larger `2*(2*gridSize + 1) + 1`, and so on. As illustrated in Figure 11.7, the physical boundaries of all grids should be the same, but the mesh size and distance between points varies.

The output from your programs should be:

> the command-line arguments
> the execution time for the computational part
> the maximum error in final values
> the final grid values

Write the first three items to standard out. Write the data values to a file `data.out`.

To calculate the execution time, read the clock after you have initialized all variables and just before you create the processes (in the parallel programs). Read the clock again as soon as the computation is complete and the worker processes have terminated (in the parallel programs).

The maximum error in final values should be the maximum difference between the final values of points and 1.0. (One does not normally know what the final values should be, of course, but the maximum error is more interesting than the value of `epsilon` described in Section 11.1.)

Timing Experiments

Your second task is to run a series of timing experiments. In particular, you are to execute your programs for the following combinations of command-line arguments:

> program 1 for grid sizes of 100 and 200
> program 2 for grid sizes of 100 and 200 and for 1-4 worker processes
> program 3 for grid sizes of 12 and 24 (smallest grid)
> program 4 for grid sizes of 12 and 24 and for 1-4 worker processes

There are a total of 20 different timing tests.

For each sequential program and grid size, first figure out what the value of `numIters` should be so that the execution time of the program is about 30 seconds for that grid size. Then use the same value of `numIters` for the parallel versions of that sequential program. You will be using four different values for the `numIters` argument.

If you write your programs in MPD, use the `age()` function to calculate execution times. If you write your programs in C and Pthreads, use the `times` function, as illustrated in the `clock.c`

Pthreads program I handed out in class. (You can view the man page by executing "`man -s 2 times`".) The value of `CLK_TCK` is 100.

Additional Experiments

The experimental method consists of making and then testing hypotheses—or asking questions and then determining answers. There are lots of different questions one might want to ask about the above programs. Following are some examples; you can probably think of others.

Accuracy. How much time does it take for each algorithm to get the same level of accuracy? How much more accurate is the faster program for the same amount of time?

Scalability. How well do the algorithms scale as you increase the grid size? Does speedup increase, decrease, or stay about the same? What if you use more workers (5 or 6)?

Overheads. How much overhead is there in your parallel programs? For example, how much time does it take to create processes? How long does it take to perform a barrier for 4 workers? How much load imbalance is there (if any)?

Time/Space Tradeoffs. Is false sharing happening for the barrier synchronization variables? Can you get rid of it by padding the declarations? Is there a performance difference between implementing barrier synchronization using a procedure versus inlining the code directly in each worker process?

Compiler Optimizations. What is the effect of turning on compile-time optimizations? (This is done by using "`-O`" options to the `mpd` or `gcc` compilers.) How much faster are the programs? What happens to the speedups for the parallel programs? **Important Point**: If you use C and Pthreads and turn on compiler optimizations, be sure to declare shared synchronization variables as `volatile`; if you do not, the compiler might put these variables in registers, which will make your program incorrect.

Pick *at least three nontrivial questions* and set up experiments to determine the answer. The choice of what to do is up to you, but do not choose just the simplest things. You might look at three different kinds of topics, or go into depth on one topic, or do a combination.

Reports

Once you have done the timing and other experiments, write a report to explain what you have done and what you have learned. Your report should be a few pages of text plus tables and figures. It should have four or five sections, as follows:

- *Introduction.* Briefly describe the problem and what your report will show.
- *Programs.* Describe your programs, stating what each does and how. Explain the program-level optimizations you have implemented.
- *Timing Experiments.* Present the results from the timing experiments. Use tables to present the raw data and graphs to show speedups and comparisons. *Also explain your results.* Do not just present the output data! What do the results show? Why?
- *Other Experiments.* Describe the questions that you set out to answer, the experiments you conducted, the results you got, and your analysis of the results. Present the results in whatever form seems most compelling to you. Your analysis should explain why you think you got the results you did.
- *Conclusion.* Briefly summarize what your report has shown, *and* describe what you have learned from this project.

Electronic Turnin

By midnight on April 3 use `turnin` to submit your programs. The assignment name is `parallel`. The programs should be named `prog1`, `prog2`, etc. *Also submit a makefile that we can use to compile your programs.* In particular, if we execute

```
        make prog1
```

your makefile should compile the program and produce an executable file that resides in `a.out`. We should then be able to execute the program with command-line arguments as specified above.

If you modify your programs while conducting timing experiments—or if you write additional programs—turn them in on April 10 when you submit your report. Please append commented listings of the final versions of your programs for the timing experiments to your report. You do not need to turn in the actual output from any of your tests, but you should have it available or readily be able to reproduce it. In short, your report should contain all the information someone else would need to reproduce your results.