Parallel Java: 24-puzzle

This assignment for the parallel programming course consists of writing a parallel program in Java using the Ibis system and running it on the DAS-4.

The 15-puzzle (also called Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and many others) is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The puzzle also exists in other sizes, particularly the smaller 8-puzzle. If the size is 3x3 tiles, the puzzle is called the 8-puzzle or 9-puzzle, and if 4x4 tiles, the puzzle is called the 15-puzzle or 16-puzzle named, respectively, for the number of tiles and the number of spaces. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space. For this assignment, you'll be implementing a parallel version of the 24-puzzle.

15-puzzle (Wikipedia)

The goal of the assignment is to write an application which is able to determine the shortest number of slides possible, as well as the number of ways to solve the puzzle, given a certain starting layout. A sequential solver is available in the tar bundle in this assignment's Canvas page. The application includes a build.xml file for building the application with ant, as well as a doc and bin directory to hold any documentation, scripts and external dependencies for your application.

Although the application randomly creates initial puzzle layouts, it is deterministic: with a given setting and seed it will always generate and solve the same puzzle, allowing for easy benchmarking of the application. The different command line parameters drastically change the application runs. If the default parameters do not work for you, try to adjust them somewhat.

Requirements

Convert the given sequential version of the solver into a parallel application that it is capable of running on multiple DAS-4 machines in parallel. To communicate, use the Ibis Portability Layer(IPL). Implement a work queue to handle passing work to the machines. You may use a single work queue, located on one of the machines. You may, for instance, elect one of the machines as "master" using the election mechanism of the IPL.

Benchmark your application by running it on the DAS-4 system. Try to get as close to perfect speedups as you can. Also, try to explain the reasons why you get lower/better than perfect performance. Test your application on 1, 2, 4, 8 and 16 nodes. Make sure you use prun to submit your job, even for jobs using only one machine. If you run a job on the frontend instead of a node, you will both overload the frontend, which is used by a lot of people, and get useless performance measurements! You may use the timers provided by Java (System.currentTimeMillis or even System.currentTimeNanos) to measure the performance of sections of your code. Optimize your program such that the grain size of the leaf jobs is controlled to improve performance (i.e., do not put very small jobs in the job queue, but calculate those directly).

Write a short report (in English, about five pages) describing your experiences with Java, the difficulties you encountered and how you solved them. Include measurements of your program. Report elapsed execution times as well as speedups. Don't just show numbers, but present a speedup graph as well. If the program does not achieve linear speedup, give an explanation including a performance breakdown for the slowdown. The document should also describe how you implemented the programs and how you measured their performance. The programs themselves should contain useful comments that illustrate how they work.

Compiling and Running your Java programs

You may compile your Java programs with `javac` or with `ant` using the provided `build.xml`. Be sure to use the `module` command on the DAS-4 to enable those applications. For more information see the "DAS-4 for PPP users" documentation in the Course Documents on the Canvas PPP website.

To run a parallel program on the DAS nodes, you need to make use of the `prun` script (see "DAS-4 for PPP users" in the Course Documents, for more information on the DAS-4 and prun). Furthermore, to run your application, you may simply use `java` or use the provided `bin/java-run` script (which adds all jars to the classpath automatically). It expects to find a lib directory in the current directory.

sequential on one machine:
```
$ prun -v -1 -np 1 bin/java-run ida.sequential.Ida
```
ipl version on 8 machines:
```
$ prun -v -1 -np 8 bin/java-run ida.ipl.Ida
```
bonus assignment, 8 machines:
```
$ prun -v -1 -np 8 bin/java-run ida.bonus.Ida
```

For more information on compiling and running IPL applications, see the user's guide and programmer's manual of the IPL included in the distribution (and the assignment template).

Submitting

You have to submit your code (preferably containing useful comments that illustrate how the application works and is parallelized) and the report.

Important: Because we use automatic test scripts to test and benchmark your submissions, you must strictly follow the instructions below.

- Make sure that your submission has exactly the same directory structure as the provided template. The main function of your program should be in the Ida.java file of the ida.ipl package.

- Also, make sure that your parallel programs give the exact same output as the sequential program. We compare your application's output with the correct output with the diff command. This also includes the destinction between printing to standard out and standard error. If there is any difference (except for the run time you print), your submission will be rejected!

  If you reimplement (parts of) your program for the bonus assignment, please put your new program in the ida.bonus package. Additionally, if you add extra command line options to the application, make sure that you use reasonable default values, because our automated test scripts do not know your command line options.

  To help you with these checks a sanity-check script is provided in the bin directory of the template. Please do not submit anything which does not pass this test. The script has a single parameter, the address of the ipl server (for testing the ipl assignment). Note that the script does not check whether your application is behaving correctly, it just checks whether the output is formatted correctly.

- Please structure your code and scripts so that your submission includes all needed dependencies (like the IPL), and simply running "ant" in the root of your applications leads to the compiling of your application. The standard `java-run` script should also work. One way to ensure all this is to not change the template :)

- Create the report as a PDF file, and make sure you place the file in the pre-created docs directory.

- Edit the `build.xml` file and fill in your name and your VU net-ID in the appropriate variables at the top of the build file. Then create a "distribution" of your code suitable for submission by using the `ant dist` command in the root of your applications. Submissions not created with the `ant dist` command will be rejected. Make sure that the entire distribution compiles correctly, including the bonus assignment. If you tried, but did not finish the bonus assignment, please do not submit unfinished code that lead to compilation errors.

Documentation

- Java 1.8 documentation https://docs.oracle.com/javase/8/docs
- The ipl section at the Ibis website http://www.cs.vu.nl/ibis
- The code for the sequential implementation, available from the Canvas PPP site.

Grading

A correct implementation of all the requirements above is graded with 8, following the grading guides outlined during the kickoff meeting. Up to 2 bonus points (to grade 10) may be given for extra work on implementation, optimizations, testing and benchmarking. For example:

- Replace the sequential workers of the default assignment by multithreaded workers that can make use of multi-core processors of the DAS-4.

- Implement work stealing instead of the single centralized work queue.

To get the bonus points, it is up to your fantasy on how you improve the parallel application or your evaluation of the application. Optimizing the sequential algorithm however, does not count as a parallel optimization for the bonus assignment. If you reimplement (parts of) your program for the bonus assignment, please put your new program in the ida.bonus package. This way, it is easier for us to grade it, and additionally you do not break the entire assignment when you make a mistake in the extra code.

TODO list

In order to complete this part of the practical, please follow the following receipt:

- Get the sequential version of the IDA* Puzzle solver from this page.
- Create a parallel application using the IPL. The IPL is already provided in the template.
- Test your application on the DAS, using `prun`; see the "DAS-4 for PPP users" documentation in the Course Documents on the Canvas PPP website.
- Benchmark your parallel application
- Write a report about the practical. Include the difficulties you encountered, how you solved them, and the performance of the resulting application. Include both run-time and speedups graphs and listings
- Optionally you can try to get the bonus for this assignment. Include a description of your extra work for the bonus in your report.
- Test your assignment with the sanity-check script
- Create a "distribution" containing your code and documentation using the provided `dist` ant target (set the `name` property at the top of the `build.xml` file to your name!) and submit it using Canvas.