PARALLEL IMPLEMENTATION OF FRINGE SEARCH

Lukas Mosimann, Christian Zeman

ETH Zürich
Zürich, Switzerland

ABSTRACT

Describe in concise words what you do, why you do it (not necessarily in this order), and the main result. The abstract has to be self-contained and readable for a person in the general area. You should write the abstract last.

1. INTRODUCTION

This project was part of the course *Design of Parallel and High-Performance Computing* given by Torsten Hoefler and Markus Püschel in autumn 2013 at ETH Zürich.

Motivation. Pathfinding is an important part in many applications, especially in computer games and robotics. Fringe search is a popular algorithm for single-pair shortest path problems. Unlike A* it won't guarantee to find the shortest path, but rather a path "short enough". The advantage of Fringe Search is that it generally outperforms A* as shown in [1].

In [2] a parallel implementation of Fringe Search has been done for a distributed memory environment. The goal of this paper was the implementation and benchmarking of a parallel version of Fringe Search for a shared memory environment.

Related work. This paper is mainly based on three papers. The Fringe Search algorithm was introduced in [1]. S. Brand and R. Bidarra implemented a parallel version in a distributed memory environment described in [3] and a bit more specific in the Master Thesis of S. Brand [2]. In this paper the implementation was done for a shared memory environment.

2. BACKGROUND: SHORTEST PATH PROBLEM

In this section we formally define the single-pair shortest path problem we consider and introduce two algorithms that are used to solve it.

Single-pair shortest path problem. A problem where the goal is to find the shortest path between a start and an end node in a directed or undirected graph. In our case we used a directed graph.

A*. A very popular algorithm that uses a best-first search approach for solving the single-pair shortest path problem. For the best-first search it uses a heuristic function (e.g. Euclidean distance to end node). It always finds the shortest possible path and therefore it's called optimal. It uses a priority queue for the nodes and therefore each insert into the queue has complexity $O(\log n)$.

Fringe Search. A single-pair shortest path algorithm that is similar to A*, but instead of a priority queue that always has the most promising node first, it stores the nodes in a simple list where an insert has the complexity $O(1)$ and visits a node if it's "promising enough" which is determined by a global threshold value that will be continuously increased. It's not optimal (it doesn't guarantee to find the shortest path) but it is generally faster than A*, as shown in [1]. An example of the algorithm can be found in figure 1.

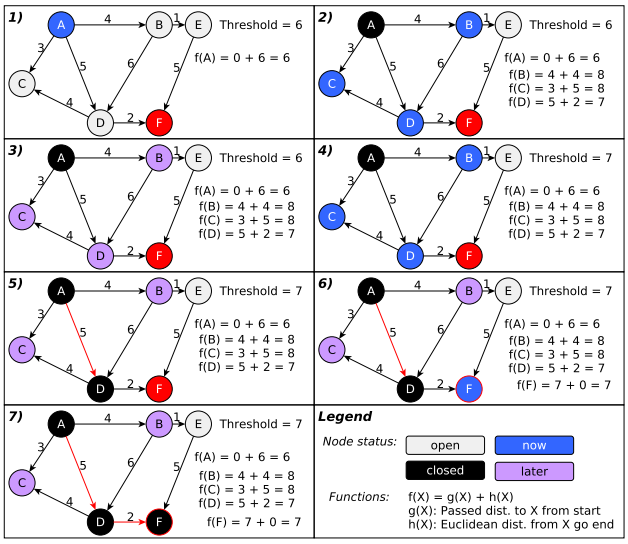

Fig. 1. Fringe Search example

3. CONCEPT AND IMPLEMENTATION

In this section we will show how the implementation has been done and illustrate the used locking concepts.

Sequential Fringe Search. As a first step we implemented a strictly sequential Fringe Search in order to have a

basis for the parallel implementation and also as a reference for the benchmarks regarding the speedup of the parallel version. The pseudocode is shown in algorithm 1.

Algorithm 1: Fringe Search (sequential)

```
create nowlaterlist;
add nodes zero and start to nowlaterlist;
threshold ← heuristicDist(start, end);
x ← start;
while x != zero do
    if x.f ≤ threshold then
        if x == end then
            delete node zero from nowlaterlist;
            return reconstructPath();        // done
        x.status ← closed;
        for each neighbour nb do
            if nb.state == open then
                calculate new distance;
                if dist < nb.g then
                    update nb.f and nb.g and set
                    nb.parent;
                    move nb right behind x in
                    nowlaterlist;
            else if nb.state = inactive then
                calculate nb.g and nb.f;
                nb.parent ← x;
                insert nb right behind x in nowlaterlist;
        delete x from nowlaterlist;
    else
        x ← x.next;
    if x == zero then
        increase threshold;
        x ← x.next;
delete zero;
return -1;     // no existing path to end
```

Locking concept.

Now comes the "beef" of the report, where you explain what you did. Again, organize it in paragraphs with titles. As in every section you start with a very brief overview of the section.

In this section, structure is very important so one can follow the technical content.

Mention and cite any external resources that you used including libraries or other code.

4. EXPERIMENTAL RESULTS

Here you evaluate your work using experiments. You start again with a very short summary of the section. The typical structure follows.

Experimental setup. Specify the platform (processor, frequency, maybe OS, maybe cache sizes) as well as the compiler, version, and flags used. If your work is about performance, I strongly recommend that you play with optimization flags and consider also icc for additional potential speedup.

Then explain what kind of benchmarks you ran. The idea is to give enough information so the experiments are reproducible by somebody else on his or her code. For sorting you would talk about the input sizes. For a tool that performs NUMA optimization, you would specify the programs you ran.

Results. Next divide the experiments into classes, one paragraph for each. In each class of experiments you typically pursue one questions that then is answered by a suitable plot or plots. For example, first you may want to investigate the performance behaviour with changing input size, then how your code compares to external benchmarks.

For some tips on benchmarking including how to create a decent viewgraph see pages 22–27 in [4].

Comments:

- Create very readable, attractive plots (do 1 column, not 2 column plots for this report) with readable font size. However, the font size should also not be too large; typically it is smaller than the text font size. An example is in Fig. **??** (of course you can have a different style).
- Every plot answers a question. You state this question and extract the answer from the plot in its discussion.
- Every plot should be referenced and discussed.

5. CONCLUSIONS

Here you need to summarize what you did and why this is important. *Do not take the abstract* and put it in the past tense. Remember, now the reader has (hopefully) read the report, so it is a very different situation from the abstract. Try to highlight important results and say the things you really want to get across such as high-level statements (e.g., we believe that .... is the right approach to .... Even though we only considered x, the .... technique should be applicable ....) You can also formulate next steps if you want. Be brief. After the conclusions there are only the references.

6. FURTHER COMMENTS

Here we provide some further tips.

Further general guidelines.

- For short papers, to save space, I use paragraph titles instead of subsections, as shown in the introduction.

- It is generally a good idea to break sections into such smaller units for readability and since it helps you to (visually) structure the story.
- The above section titles should be adapted to more precisely reflect what you do.
- Each section should be started with a very short summary of what the reader can expect in this section. Nothing more awkward as when the story starts and one does not know what the direction is or the goal.
- Make sure you define every acronym you use, no matter how convinced you are the reader knows it.
- Always spell-check before you submit (to us in this case).
- Be picky. When writing a paper you should always strive for very high quality. Many people may read it and the quality makes a big difference. In this class, the quality is part of the grade.
- Books helping you to write better: [5] and [6].
- Conversion to pdf (latex users only):

  dvips -o conference.ps -t letter -Ppdf -G0 conference.dvi

  and then

  ps2pdf conference.ps

Graphics. For plots that are not images *never* generate the bitmap formats jpeg, gif, bmp, tif. Use eps, which means encapsulate postscript. It is scalable since it is a vector graphic description of your graph. E.g., from Matlab, you can export to eps.

The format pdf is also fine for plots (you need pdflatex then), but only if the plot was never before in the format jpeg, gif, bmp, tif.

7. REFERENCES

[1] Yngvi Björnsson, Markus Enzenberger, Robert C. Holte, and Jonathan Schaeffer, "Fringe search: beating a* at pathfinding on game maps," in *In Proceedings of IEEE Symposium on Computational Intelligence and Games*, 2005, pp. 125–132.

[2] Sandy Brand, "Efficient obstacle avoidance using autonomously generated navigation meshes," M.S. thesis, Delft University of Technology, 2009.

[3] Sandy Brand and Rafael Bidarra, "Multi-core scalable and efficient pathfinding with parallel ripple search," *Computer Animation and Virtual Worlds*, vol. 23, no. 2, pp. 73–85, April 2012.

[4] M. Püschel, "Benchmarking comments," online: http://people.inf.ethz.ch/markusp/teaching/263-2300-ETH-spring11/slides/class05.pdf.

[5] N.J. Higham, *Handbook of Writing for Mathematical Sciences*, SIAM, 1998.

[6] W. Strunk Jr. and E.B. White, *Elements of Style*, Longman, 4th edition, 2000.