COMP90025 Parallel and Multicore Computing

Project 1 - Multiple Sequence Alignment

Aaron Harwood

School of Computing and Information Systems
The University of Melbourne

2021 Semester II

k Sequence Alignment

- A sequential algorithm, in C/C++, for computing the alignment of k sequences has been provided on Canvas.
- The sequence alignment problem is roughly, given two sequences of symbols, insert gaps into the sequences such that the penalty, that is a function of how many gaps are inserted and mis-matches in symbol alignment, is minimized, when the two sequences are compared symbol for symbol at each index location. This problem is commonly attributed to Bioinformatics where it is concerned with aligning sequences of amino acid base pairs, or gene sequences. However sequence alignment has many varied applications.
- The k sequence alignment problem is given k sequences on the input and is required to compute the sequence alignment between all $k(k-1)/2$ pairs.

Code Summary

- We have supplied you with skeleton code that does k sequence alignment.
- The skeleton code serves as the sequential code as well, for the purpose of correctness in output.
- The problem is read from standard input.

Example input

The problem instance has three parameters: mis-match penalty, gap penalty and number of sequences, k. Each sequence can be arbitrarily long (up to about 80,000 symbols max length).

```
$ cat seq.dat | ./seqalign-skeleton
Time: 132 us
602d0f604e8fb908195d53e681094f7d063c4168a33a18f32b4ca3d29f27073
5 4 9
```

Output

- We will not output all of the sequence alignments as there are $k(k-1)/2$ such pairs and they are largely unimportant to us, other than for determining correctness when compared to the sequential code. But we do want to know whether your code can generate the correct sequences and testing the minimum penalty found is insufficient for this. Therefore your code will compute a combination of SHA512 hashes (see the skeleton code for details) of the sequences and output the final hash only. You will also output a line after that with all of the minimum penalties for the $k(k-1)/2$ individual sequence alignment problems. See the supplied code to see exactly how this must be done.
- You are provided with a `sha512.hh` file that you must use to do the SHA512 hashes and you are not permitted to modify that file. See the supplied code for examples how to use it. There may be room for performance improvements around the combination of the hash results, which is accessible to you, and there may be some parallelism that can help here.

Tasks for you to do in Project 1 I

- The project is about experimentation with parallelism using OpenMP, on a single machine on Spartan (a multi-socket NUMA node), and reporting your experimental results and observations in a written report, along with your code that provides the best *speedup* that you managed to achieve. Since speedup is always with respect to a reference sequential implementation, your reference implementation must be clear (it can be the sequential implementation that we gave you). Sometimes it is preferable/better to consider speedup with respect to the same parallel program running with a single thread. You may consider *efficiency* as well, in the sense that you are interested in how well your parallelization makes use of the available resources.
  - Your experimental results *must* be obtained from a Spartan node with a NUMA architecture (multi-socket). So please start early. It will be efficient for you to run multiple experiments in a single job on Spartan, rather than running many smaller jobs.
  - It would be preferable if you requested a single node, i.e. an entire node free, so that you can exclude interference from other jobs.

Tasks for you to do in Project 1 II

  - You should run each experiment at least 3, but perhaps 10 times, in order to obtain a *mean* and *standard deviation* in run times.

- The basic form of an experiment is to show the speedup achieved for a given parallelization approach/strategy/setting. You can show speedup (and other measurements of your choice) achieved for various numbers of threads, e.g. $1, 2, 4, 8, 16, 32, \ldots$, and for various other parameters of your parallelization approach. Your experiments can show speedup "with and without" a given parameter or setting or use of a technique, e.g. synchronization hints and work-sharing schedules. Your experiments can also show speedup with and without compiler flags, however try to focus on OpenMP/parallel concepts.

Tasks for you to do in Project 1 III

- The multi-socket Spartan node is a NUMA architecture. Your experiments should include NUMA aspects of OpenMP to show that you have attempted to increase performance via thread affinity and memory allocation strategies. As well, you may consider experimenting with OpenMP tasks instead of using threads. All of these concepts are covered in the lecture slides.
- You may include aspects not discussed in lecture slides, e.g. SIMD in OpenMP. I don't recommend using SIMD instructions directly however you may if you wish experiment with that to see if you can obtain better speedups.
- You must prepare a written report:
  - Maximum 2000 words excluding figures and pseudo-code. Use the ACM style guide found here: `https://www.acm.org/publications/proceedings-template`.

Tasks for you to do in Project 1 IV

- (100–200 words) abstract: summarize the approach that you observed provided the best speedups, state the speedups and state the most interesting observation that you made.
- (500–750 words) Introduction: provide a brief overview of the problem being solved (i.e. sequence alignment) and discuss possible parallelization strategies at a high-level with justification of those strategies. You may research existing parallelization strategies and cite them. Discuss the high-level strategy that you took in your experiments. You may break this section into Background, Related Work and Proposed Approach sub-sections if you wish.
- (850–1300 words) Experiments: show the results of your experiments and analysis/discussion of the results
- (100–200 words) Conclusion: comment on the main results and aspects of the experiments that could be improved if you had more time

Assessment I

- Project 1 is worth **20%** of your total assessment. It is individual work.
  - Overall, assessment is based on your ability to achieve the best possible speedup (shortest running time) for the problem using OpenMP on a multi-socket Spartan node (NUMA node) and your ability to write an effective technical report. Proper use of OpenMP NUMA techniques will give you the ability to make maximum use of the processing units available on any single Spartan Node. The maximum theoretical speedup for a program using N threads is a speed up of N; but this depends on the sequential reference implementation. We will compile and run your submitted program with a number of test cases that are similar to the large example provided on Canvas. We will not provide you with the test cases that we will use. In particular for each of our test cases the value of k may be small (e.g. just a few sequences) or large, you should not assume that it will be sufficiently large to justify parallelization only at the level of sequence pairs. The penalty parameters have very little impact on runtime and will be just straight forward values like in the examples. Sequence lengths in our test cases will range from small e.g. 10,000 symbols, to large, e.g. 80,000

Assessment II

symbols. We will ensure that the memory requirement, given our test cases, for the sequential program never exceeds 32GB. You must ensure that your implementation does not require more than 32GB.

- Parallel Program (50%): The submitted program must be the program that you managed to obtain the best speedup for. You must put a comment at the top of your program that indicates which Spartan node (with slurm parameters for your job request as well if you wish) and compiler options that you used to achieve the speedup that you observed and any OpenMP environment variables that you used. If we cannot compile your program then you receive at most half of the marks for the Parallel Program assessment (we will give you a chance to demonstrate that it can be compiled), based on code quality. The correctness and wall clock time for each test case will be looked at. Error in the output will lead to marks being deducted. Poor or incorrect use of OpenMP in the program (e.g. use of techniques that actually harm performance, clauses that don't make sense, etc., and not attempting to program for NUMA) will lead to marks being deducted. Poor performance overall (not being able to achieve at least

Assessment III

some speedup or reduction in running time compared to our sequential implementation) will lead to a deduction in marks. The amount of marks deducted will reflect the severity of the issue.

- Written Report (50%): Assessment is weighted over the different parts of the report roughly by word count limitations given earlier. Assessment is based on coverage and depth of knowledge in relevant (to the problem) parallel computing techniques and theoretical understanding, correctness, demonstration of critical thinking (do not simply report observed data, but comment on and explain your observations as well as you can, connecting with theory where useful), and presentation of data (appropriate charts, tables) and overall writing style. Please ask us if you are unsure of what to write in your report.

Submission

- Submission of Project 1 is due on the Friday of Week 8. Submission guidelines will be given closer to the submission date. It will require you to submit your report and a single C/C++ file that represents the program that gives you the best speed up. You must put a comment at the top of your submitted program as specified in the assessment points earlier.