Parallel Smith-Waterman with OpenMP

ISITHA SUBASINGHE, University of Melbourne, Australia

Sequence alignment is the process of comparing and detecting similarities in biological sequences such as RNA and DNA. The traditional algorithm presented for this problem is inherently sequential, this report explores how the problem may be parallelised using the "wavefront" technique. The algorithm implemented implemented using OpenMP demonstrated a speedup of 9.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms**.

1 BACKGROUND

The Smith-Waterman [2] algorithm for “molecular

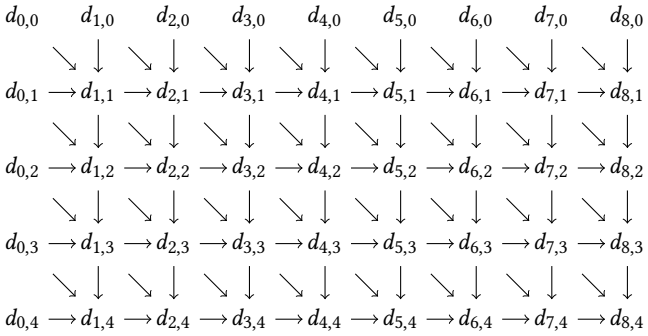

Fig. 1. Dependencies for values in Smith-Waterman matrix

2 RELATED WORK

2.1 SIMD

2.2 GPU

2.3 FPGA

3 PROPOSED APPROACH

3.1 Naive Algorithm

3.2 Parallel Algorithm

3.2.1 Restrictions. Unfortunately, there was a restriction in the implementation of our algorithm, namely the amount of memory available proved restrictive in exploring further opportunities for parrallelism. Computing the sequence alignment among all $\frac{k(k-1)}{2}$ pairs falls to the category called “embarrassingly parallel”, however, leads to far greater memory consumption and therefore was not explored.

Author’s address: Isitha Subasinghe, University of Melbourne, Melbourne, Victoria, Australia, isubasinghe@student.unimelb.edu.au.

Algorithm 1 Sequential Smith-Waterman

```
1:  procedure SeqSmithWaterman
2:      m,n ← dimensions of matrix
3:      pgap ← gap penalty
4:      pxy ← mismatch penalty
5:      dp[m][n] ← newMatrix(m+1,n+1)
6:      x ← sequence x
7:      y ← sequence y
8:      for i ← [0 . . . m] do
9:          dp[i][0] ← i ∗ pgap
10:     for i ← [0 . . . n] do
11:         dp[0][i] ← i ∗ pgap
12:     for i ← [1 . . . m] do
13:         for j ← [1 . . . n] do
14:             if x[i-1] = y[j-1] then
15:                 dp[i][j] = dp[i-1][j-1]
16:             else
17:                 dp[i][j] = min(dp[j-1][i-1] + pxy, dp[j-1][i] + pgap, dp[j][i-1] + pgap)
        return traceback(dp)
```

3.2.2 Forumlation of optimal speedup. We can apply Amdahl's law to determine the optimal speedup possible in our implementation. The equation for Amdhal's law is given by the following:

$$S = \frac{1}{1 - p + \frac{p}{n}} \tag{1}$$

Our algorithm's best p (fraction of work that can be executed in parallel) can be determined by the equation below:

$$p = min(m, n) \tag{2}$$

Note that the fraction of work that can be executed in parallel is dependant on the matrix itself, this is because the dimensions have an impact on how much we are able to parallelise.

3.2.3 NUMA awareness. NUMA (non-uniform memory access) becomes an area of concern when considering programs that use multiple cores. More accurately, when multi-core programs require access to the same memory locations as each other. The strategy used to help amortize the cost of NUMA in the Parallel implementation is called "First Touch". "First Touch" ensures that the NUMA node to initialise some memory location, will have that page allocated on a memory device physically closer to the NUMA node.
The steps for memory allocation can be given as follows:

- dynamic memory allocation call (malloc/new)
- initial write attempt of dynamically allocated memory
- page fault triggered when write is attempted
- physical allocation of memory performed and mapped via MMU

The parallel implementation exploits this property on moderns operating systems to help amortize the cost of non-uniform memory access. This is performed by initialising the matrix used for Smith-Waterman through multiple threads,

ensuring that each NUMA node has a slice of the memory and thus decreasing the number of transfers done over the interconnect.

4 EXPERIMENTS

4.1 Experiment Environment

Two experimentation environments were used through the development process. The development environment was used as a quick way to test and filter out ideas that weren't viable, while viable ideas were tested on the Spartan High Performance Computing cloud [1].

4.1.1 Development Environment.

- CPU - AMD Ryzen™ 7 3700X @ 3.6-4.4GHz
- Memory - 32GB
- Compiler - GNU 11.1.0

4.1.2 Spartan Environment.

- CPU - Intel(R) Xeon(R) Gold 6154 CPU @ 3.00-3.7GHz
- Memory - 32GB
- Compiler - GNU 10.3.0

4.2 OpenMP allocator

In the current implementation, there is some extra work performed to take advantage of the "First Touch" policy, we initialise the matrix to the value "0". While this is helpful for distributing the pages among the NUMA nodes, it comes at an additional processing cost. With the GNU 11.1.0 compiler toolchain, we are able to make use of the OpenMP allocator to do this distribution of pages, additionally some further optimisations maybe performed to increase performance, such as removing thread safety from the allocator and ensuring that the allocated memory is aligned.

5 FURTHER WORK

5.1 Generation of tiles

The parallel algorithm implemented does not make full use of the available computing power. When tiles are iterated over, they must wait until all the tiles in the same diagonal have completed their work. This is an unnecessary restriction, as it results in a large amount of time spent on the barrier. Further work could explore how to reduce the level of granularity in our dependencies, new tiles could simply be generated after a single tiles work unit has completed. This proved to be difficult to program in OpenMP as it needed a thread safe queue. While OpenMP does provide a locking mechanism, this is not sufficient to implement a queue for the purpose of a breadth first search of the generated tiles.

5.2 Inter-sequence SIMD parallelism

As noted in [], we could apply the inter-sequence SIMD parallelism into our current parallel implementation to obtain instruction level parallelism as well. While this is plausible, an important consideration that warrants investigation is determining the most efficient method to move data from the heap and into the SIMD register, this technique will likely have significant speedups with AVX512.

REFERENCES

[1] BERNARD MEADE, LEV LAFAYETTE, Greg Sauter, and DANIEL TOSELLO. 2017. Spartan HPC-Cloud Hybrid: Delivering Performance and Flexibility. https://doi.org/10.4225/49/58ead90dceaaa

[2] Temple F Smith, Michael S Waterman, et al. 1981. Identification of common molecular subsequences. *Journal of molecular biology* 147, 1 (1981), 195–197.

A APPENDIX

Algorithm 2 Parallel matrix allocation

```
    procedure NewMatrix
        m,n ← dimensions of matrix
3:      dp ← malloc(sizeof(int *)*m)
        size ← m*n
        dp0 ← malloc(sizeof(int)*size)
6:      for i ← [0 . . . size) parallel do
            dp0[0] ← 0
        for i ← [0 . . . m) parallel do
9:          dp[0] ← 0
        dp[0] ← dp0
        for i ← [1 . . . m) do
12:         dp[0] ← dp[i-1] + n
        return dp
```