Implementation of Smith-Waterman algorithm in OpenCL for GPUs

Dzmitry Razmyslovich[1], Guillermo Marcus[1], Markus Gipp[1], Marc Zapatka[2] and Andreas Szillus[2]

RUPRECHT KARL UNIVERSITY OF HEIDELBERG

[1]Institute of Computer Engineering (ZITI), University of Heidelberg, Mannheim, Germany
Main contact: dzmitry.razmyslovich@ziti.uni-heidelberg.de
Other contacts: see http://li5.ziti.uni-heidelberg.de

[2]German Cancer Research Center, Heidelberg, Germany
Email: m.zapatka, a.szillus@dkfz-heidelberg.de

DEUTSCHES KREBSFORSCHUNGSZENTRUM IN DER HELMHOLTZ-GEMEINSCHAFT

Abstract

We present the implementation of Smith-Waterman algorithm done in OpenCL. This implementation is capable of computing similarity indexes between query sequences and a reference sequence with or without sequence alignment paths. In accordance with the requirement for the target application in cancer research the implementation provides processing of very long reference sequences (in the order of millions of nucleotides).

Biological Problem Description

All cancers are results of changes (aberrations) occurred in the DNA sequence of the genomes of cancer cells. The identification of even most complex aberrations can be done with Smith-Waterman algorithm by processing a long reference sequence and short query sequences. The former is a genome sequence which can be rather long while the latter are the product of the second-generation technology. The basic problem solved with the implementation presented lies in an alignment of short query sequences along a long reference sequence.

$$H[i,0] = 0, 0 \le i \le n,$$

$$H[0,j] = 0, 0 \le j \le m,$$

$$H[i,j] = \max \begin{cases} 0 \\ H[i-1,j] + IF \\ H[i,j-1] + RF \\ H[i-1,j-1] + S(i,j) \end{cases},$$

$$1 \le i \le n, 1 \le j \le m$$

Smith-Waterman Algorithm

The idea of alignment lies in filling the nxm matrix H, the similarity matrix, where n is the number of elements in a query sequence and m is the number of elements in a reference sequence. The values of the matrix are computed using dynamic programming according to formula 1. Each value H[i; j] is the measure of similarity of two subsequences: a query sequence up to the i-th element and a reference sequence up to the j-th element.

In most cases vast amount of sequencer data must be filtered at the beginning. This task only requires similarity values

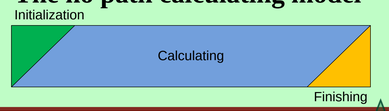
The possibility of faster computation of solely similarity values

Comparison of OpenCL implementation with other ones

NO PATH CALCULATING VERSION

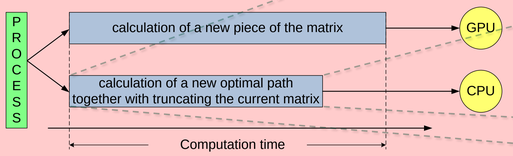
- Initialization – performs some calculations to make the wavefront technique usable
- Calculating – calculates the whole matrix excluding heading and ending
- Finishing – calculates the ending with saving the results

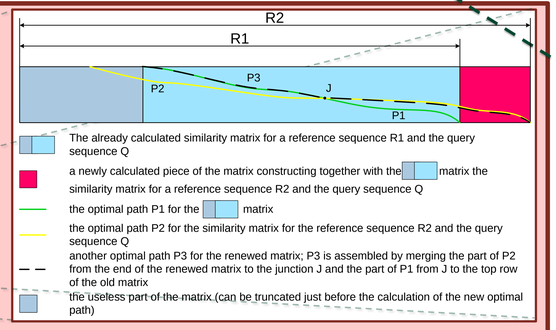
The no path calculating model



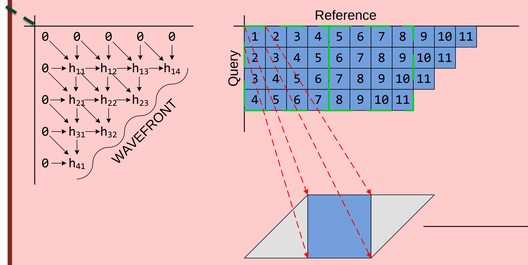
Step 1. The long reference sequence processing and online computation

The data size requirements put limits to the possibility of storing the matrix. It is necessary to use online computation of the paths. It means calculating the paths for the already calculated part of the matrix and truncating the matrix concurrently with computation of a new piece of the matrix. Choosing a nucleotide from a query sequence as a parallelization grain makes online computation possible.

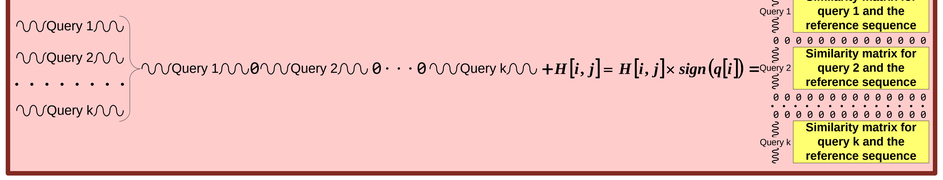




Step 2. The diamond calculation shape



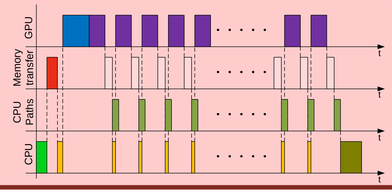
Step 3. The multiquery processing



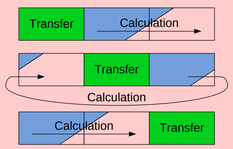
Step 4. The concurrent data transfer and kernel execution

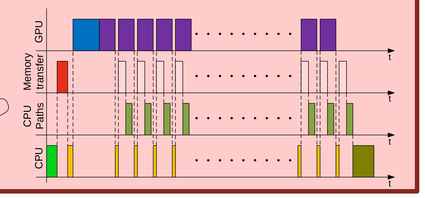
PROCESS

- Host initialization
- Input data transfer
- Scheduling running
- Precalculation kernel
- Calculation kernel
- Matrix transfer to host memory
- Path calculations
- Printing results

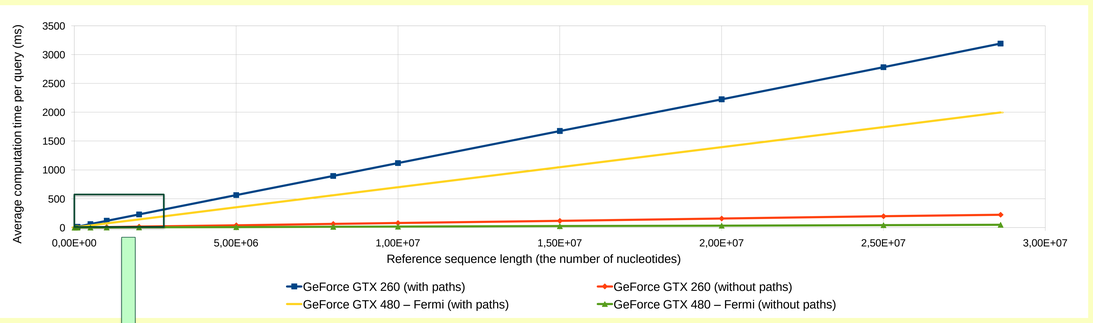


Calculating a window of the matrix together with transferring the previous window and its processing is possible, due to the independent functionality of the GPU DMA controller and GPU multiprocessors. And since path calculating and matrix block calculating tasks are processed with different devices, these tasks can also be handled concurrently. To enable the possibility to overlap data transferring and kernel execution, a ring buffer is allocated in device memory.

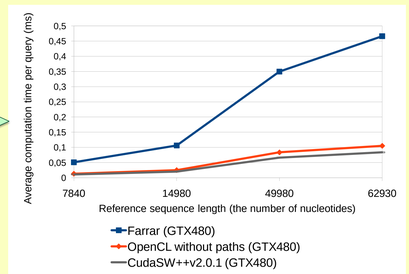


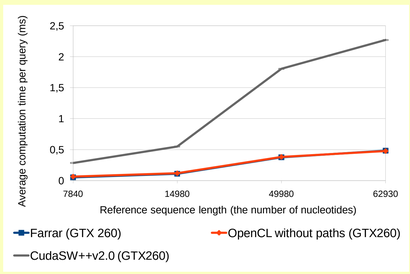


Results



The bandwidth of 5.9 GB/s was reached in the tests. The index doesn't depend on the tests data.





Test space includes:
- a reference sequence: the sequence of chromosome 21 - circa 28 million nucleotides in length (from the NCBI Build 36 of the human reference assembly);
- query sequences: a set of 36 nucleotide long reads of equal length from an Illumina genome analyzer.

We use a computation time to measure the performance. The computation time includes:
- the kernel execution scheduling time,
- the kernel execution time,
- device-to-host data transferring time (for the version with path calculating),
- the paths calculation time (for the version with path calculating),
- device-to-host results transferring time (for the version without path calculating).

Test platform:
- the NVIDIA GeForce GTX 260 GPU with 1.75GB of RAM, 30 multiprocessors and 216 cores;
- the NVIDIA GeForce GTX 480 GPU with 1.5GB of RAM, 15 multiprocessors and 480 cores;
- the Intel i7-920 CPU;
- 6GB of RAM;
- the Linux OS with the installed NVIDIA GPU Computing SDK 3.1.

Conclusion

Implementation strengths:

❖Principal advantage: alignment paths calculation;

❖Efficient processing of long reference sequences (up to 28 million in the tests);

❖High performance characteristics:
- on *GTX 480 (Fermi)*: competitive to CUDASW++v2.0.1 implementation and **4.5x** as fast as Farrar's implementation;
- on *GTX 260*: competitive to Farrar's implementation and **3x** as fast as CUDASW++v2.0 implementation;
- the acceleration in comparison with our CPU implementation is **14.5x** for the path calculating version and **610x** for the no path calculating version;

❖Heterogeneous platform independence.