Parallelization of the banded Needleman & Wunsch algorithm on UPMEM PiM architecture for long DNA sequence alignment

M. Mognol, D. Lavenier, J. Legriel
Univ. Rennes, CNRS-IRISA, Inria - UPMEM

***Abstract*—This paper studies the performance of the UPMEM PiM architecture for long DNA sequence alignment. It investigate different optimization strategies and shows that, as for x86, UPMEM PiM can benefit from the specificity of its instruction set. The Needleman & Wunsch implementation is based on an adaptive banded dynamic programming algorithm. Contrary to common belief, the analysis of two alignment use cases showed that the DP algorithm has low memory requirements. However, performance evaluation shows that the UPMEM PiM architecture, which is primarily intended for data-intensive applications, can outperform server-grade CPUs by a factor of 8 for aligning long DNA sequences.**

I. Introduction

The alignment of protein or DNA sequences is a basic treatment in bioinformatics. It is involved in many pipelines for processing genomics data, and can be very time consuming, particularly for large datasets such as those generated by the third generation sequencers that output long DNA sequences. The error rate of these sequences, compared to the short sequences produced by second generation sequencers, is generally quite high and requires heavier processing with more complex algorithms.

Global alignment is one type of alignment, it compares two sequences entirely, from the start to the end of both of them. Global alignment of DNA sequences are used in many time consuming situations. One of them is the construction of a score matrix to build phylogeny trees. The input data set is a pool of "conserved" DNA or RNA sequences from different species such as the 16S Ribosomal RNA for bacteria. A pairwise comparison between all sequences gives a score matrix reflecting the distance between all sequences. Another situation where global alignment is omnipresent is in the problem of genome assembly, in the polishing phase where high quality consensus sequences are required. More generally, the production of high quality reads from raw reads produced by sequencers on the same genomics regions requires the computations of large consensus sequences.

Global sequence alignment is mainly based on dynamic programming algorithms. Shortly, it consists in finding the minimal number of basic edit operations to go from one sequence to another one. Edit operations are substitution (one character is transformed into another one), insertion (one character is added) and deletion (one character is lost). Needleman and Wunsch (N&W) [6] proposed an algorithm based on dynamic programming that finds the optimal way (the minimum of edit operations) to transform one sequence into another one. The algorithm complexity is $0(N^2)$ if N is the size of both sequences.

The N&W algorithm provides the optimal alignment between two sequences. Its complexity is, however, relatively high, which is particularly penalizing when long sequences are compared. In fact, when two sequences are close, the useful information to compute the alignment is located on the diagonal of the matrix. Under these conditions, it is sufficient to calculate only a band around the diagonal. The complexity is greatly reduced and the results are the same as long as the number of editing operations remains within a range compatible with the size of the band.

This algorithm has been widely optimized for the x86 architecture [1], [3], [8]. An efficient implementation can be found in the minimap2 tool [5] (and its underlying library KSW2). In this paper we study the massive parallelization of the N&W algorithm on a UPMEM PiM server and compared it to KSW2. Input datasets are (1) a set 16S Ribosomal RNA sequences and (2) a collection of raw long reads produced by a sequencer for consensus. We show that using a PiM server, an acceleration of up to 8.7x can be obtain compared to a standard server grade machine.

The rest of the paper is structured as follows: first the UPMEM PiM server and its programming environment are presented. Section III describes the N&W algorithm, its affine version, then the static and dynamic banded optimizations. Section IV details the parallelization of N&W on the PiM server. Section V provides results of different experimentation and finally section VI concludes this paper.

II. UPMEM PiM System

A. Architecture

UPMEM has developed a "Process-in-Memory" (PiM) [2] component that integrates DRAM memory and processing units on the same chip. The PiM device is in the form of DIMMs (Dual Inline Memory Module) where each module contains 2 ranks of 64 DPUs (DRAM Processing Units). Each DPU can only access a memory block of 64 MB (MRAM), for a total of 8 GB of memory per module.

The host server directly access data from the 64MB DRAM block of each DPU. It also has the responsibility to upload and launch specific programs on the DPUs. After a program

is launched, the whole rank is locked until all the DPUs on that rank have finished their computation.

One of the main features of the DPU is that it has its own instruction set (ISA) with fused jump instructions. This mechanism allowing cycle-free jumps after most instructions improves execution time by having efficient branching. Furthermore, to speedup memory accesses, a DPU houses 64 KB of working RAM (WRAM) which is accessed in one cycle, similar to a high bandwidth cache.

A DPU is a true multi-threaded processor that moves to another process every clock cycle. Up to 24 hardware threads, called tasklets, can be run simultaneously and share the DPU resources. The DPU instruction pipeline is 11 deep, meaning that each instruction needs 11 cycles to retire. Furthermore, this pipeline cannot simultaneously execute multiple instructions from the same tasklet. Consequently, for optimal performance, at least 11 tasklets must run in parallel.

Direct communication between DPUs is not possible as there is no interconnection network between them. Moving data from one DPU to another requires the intervention of the host and must pass through it. This architecture model must be taken into account when designing parallel algorithms for a PiM-based UPMEM system. In fact, the workloads on one DPU must be independent of the others to achieve good performance. For large systems (several hundred GB of main memory), the challenge is therefore to implement efficient parallel algorithms by wisely using the UPMEM PiM architecture and ensuring good load balancing between thousands of threads.

A complete PiM-based UPMEM system is generally composed of a legacy DRAM memory and a PiM DRAM memory as shown figure 1. The host (a multi-core processor) access both DRAMs in an undistinguished way.

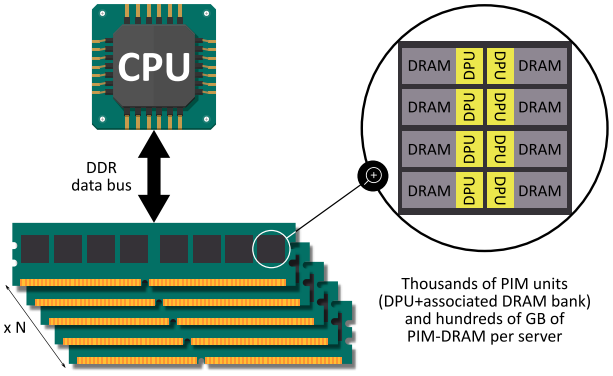

Fig. 1. Server with PiM DIMMs.

B. Programming environment

Programming an UPMEM-PiM server usually requires to write two programs (one for the host CPU, one for the DPUs) and to synchronize them together.

An illustrated example is the querying of a database permanently stored inside the PiM memory. The task of each DPU will be simply to process a small part of the database. The task of the host will be to broadcast a request to each DPU, to run the DPU, and to collect the results from each DPU. In that case, only two programs are needed.

DPU programs must be written in C. A library of routines is provided to handle DPU features such as tasklet synchronization or efficient memory transfers between the small WRAM and the 64 MB MRAM. Simulator and debug tools are also available to test the PiM parallelization of new programs.

From the host side, programs must be written in C, C++, Python or Java. This program controls the whole application. It has the job of initializing the DPUs with the right programs, moving data to or from the DPUs, synchronizing the DPU execution, etc. A library of primitive routines is also provided to perform these basic tasks.

III. Dynamic Programming Algorithm

This section recalls the dynamic programming (DP) technic for finding alignment between two sequences, and motivates the use of the banded approach for a PiM implementation.

A. Needleman & Wunsch algorithm

In the pairwise sequence alignment problem, we consider as input a pair of sequence $A = a_1, a_2, ..., a_i, ..., a_m$ and $B = b_1, b_2, ..., b_j, ..., b_n$ where a_i and b_j are chosen from a finite alphabet, e.g. A,T,G,C. The output is a sequence alignment and a score. The N&W algorithm computes the optimal score using the following recursion:

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + sub(a_i, b_j) \\ H_{i-1,j} - gap \\ H_{i,j-1} - gap \end{cases} \quad (1)$$

Where $sub(a_i, b_j)$ is the substitution cost and with the following initialization:

$$\begin{aligned} H_{0,j} &= j \times -gap \\ H_{i,0} &= i \times -gap \end{aligned} \quad (2)$$

For DNA sequences it has a positive value if $a_i = b_j$ (match) and a negative value otherwise (mismatch). gap correspond to the penalty cost for insertion or deletion. The global alignment score is given by $H_{m,n}$. To get this score and due to the recursion equation, the entire matrix H need to be computed.

Then, from the H matrix, the alignment is constructed with a traceback procedure by retrieving the path in the matrix that led to this score.

B. Affine gap extension

In fact, from a biological point of view, the gap penalty model presented in the previous section is not satisfactory: when successive gaps occur, the penalty is generally too high. Gotoh [4] introduced a better way to model gap penalty using an affine function. Two penalties are considered: the open gap penalty and the extension gap penalty. In that case, the recursion is more complex and three matrices (H,I,D) of size $m \times n$ need to be computed, instead of a single one for the original N&W algorithm. The alignment is constructed using the same traceback strategy, but information from the three matrices are required.

C. Banded DP algorithm

As already mentioned, the complexity of the dynamic programming algorithm is in $O(m \times n)$. For long sequences, especially for DNA reads coming from the third generation sequencing machines, whose length range from 10kbp to 100kbp, the execution time can become prohibitive. If there is not much divergence between the sequences, the useful information for calculating the score is mainly located on the diagonal of the matrix. In other words, it is not necessary to compute the whole matrix, but only the cells around the diagonal. The strategy of the banded DP algorithm is therefore to fill the cells in a predefined band width near the diagonal. The complexity of the algorithm is reduced to $O(w \times (m+n))$ where w is the width of the band. Practically, this size can be set to a few hundreds, leading to a significant gain in term of execution time and memory space.

From the perspective of UPMEM's PiM architecture, this strategy is essential. Storing entirely inside the MRAM the three matrices of size $m \times n$ becomes impossible for large values of m and n. On the other hand, storing only a few hundred cells around the diagonal is quite possible.

D. Adaptive banded DP algorithm

In order to compute a correct score using the banded DP algorithm, the path leading to that score must stay within the band. Then, the size must be chosen according to estimation of local maximum cumulative number of gaps. It also depends of the difference between the length of the 2 sequences. Larger the difference, bigger the band size. Estimating cumulative gaps is not easy, and often leads to an overestimation of the band size, and consequently an excess of calculations.

The adaptive band proposed by [7] is an efficient heuristics to partially overcome these disadvantages. The computation of the band is performed on an anti-diagonal window as shown figure 2. At the beginning, the window is centered at position [0,0] of the matrix (top left corner). Depending on the values at the extremities of the window, it is shifted right or down to follow the most likely path.

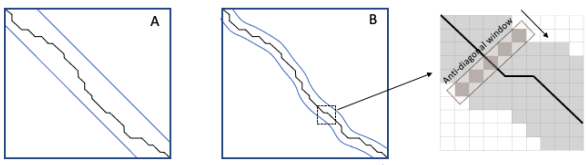

Fig. 2. (A) Fixed band: the optimal path must be located within the band whose size depend of the number of gaps and the length difference between the two sequences. (B) Adaptive band: the position of the window is adjusted according to values computed in the anti-diagonal

This heuristics significantly decrease the number of calculation as the size of the window can be much smaller than the size of the fixed band.

IV. Implementation

Our N&W implementation on UPMEM PiM server requires two programs, one for the host and one for the DPUs. Shortly, the host program dispatch the sequences into the DPUs, and the DPU program perform the alignment computation. This section details these two programs and how they synchronize together.

A. Host program

The host performs the following actions:

- dispatch sequences to the DPU
- synchronize the DPU computation
- collect the results

Despite its apparent simplicity, these actions require close attention to provide good performance, especially for optimizing the data transfers and ensuring efficient load balancing between DPUs.

Data transfer optimization: In order to reduce the size of the data flowing through the different server memories, characters of the DNA sequences are packed into 2 bits since the alphabet is limited to 4 characters: A, C, G and T. This greatly help to minimize the quantity of data transfered between the host and DPUs, it also reduce subsequent transfer between MRAM and WRAM.

Load balancing: The way in which the DNA sequences are distributed to the DPUs completely determines the overall load balancing. This part is essential because all DPUs in a rank have to finish their work before the results are collected. The interval of time between the fastest and the slowest DPU must be as small as possible.

As mentioned in III-C, the complexity of an alignment between sequences A and B gives the following equation estimating its workload:

$$W(A,B) = (m+n) \times width \tag{3}$$

The workload of a DPU is estimated based on the sum of all alignment it has to compute. The next step is to allocate similar workload for each DPU. First the overall workload is determined, then the average workload for each DPU is computed. Finally a simple greedy algorithm is applied to load every DPU with a workload near the average. If the number of sets is large compared to the number of DPUs, this simple algorithm provide a good load balancing while keeping preprocessing low.

B. DPU program

1) Score computation: As stated in III-B, 3 matrices need to be computed. However, to get the score ($H_{m,n}$) it is not necessary to keep all the values of these matrices. The computation of $H_{i,j}$, $D_{i,j}$ and $I_{i,j}$ requires only the presence of the neighborhood values $H_{i-1,j-1}$, $H_{i,j-1}$, $H_{i-1,j}$, $Di, j-1$ and $I_{i-1,j}$. Thus, instead of storing 3 matrices, only 4 anti-diagonals of size w (the width of the band) need to be updated: the two previous anti-diagonals of H and one anti-diagonal from the I and D matrices.

The memory footprint to compute the score of an alignment between two DNA sequences is reduced to: 4 integer arrays of size w that can fit inside the WRAM of each DPU.

2) Traceback algorithm: This procedure aims to provide the optimal alignment. It works on two steps: (1) it fill an extra data structure, during the score computation, memorizing how the optimal score has been reached; (2) it navigate through this structure to extract the optimal path.

(1): The data structure is an array, called BT, of size $(m+n) \times w$. The ith line of BT represents the ith anti-diagonal. Schematically, a cell of BT stores which neighboring cell of H has contributed to compute the maximal score. This score can come $H_{i-1,j-1}$, $I_{i,j}$ or $D_{i,j}$. This information is encoded into 3 bits as follows: 2 bits indicates the origin cell, H with match, H with mismatch, I, or D. 1 bit precise if, in case of the origin cell is equal to I or D (gap), this is an opening or an extension gap.

(2): From the BT array, the traceback generates a CIGAR string that specifies the alignment between the two DNA sequences. Starting from the BT cell attached to $H_{N,M}$ the function output the suite of elementary operations (match, mismatch, gap) that reflects the transformation of one sequence into the other one. The function stop when the first anti-diagonal is reached. As the CIGAR string is constructed from the end, it must be reversed for proper use.

3) Multitasking: As explained in II-A to reach optimal performance 11 tasklets must run in parallel at every clock cycle. There are essentially two ways to parallelize an alignment: (1) by performing multiple alignments simultaneously; (2) by using multiple tasklets to compute a single alignment. The first cannot be implemented with at least 11 tasklets due to memory constraints. The latter can be rather inefficient as it requires costly synchronisation between tasklets.

Here, an hybrid solution as been implemented: P pools of T tasklets to simultaneously align P pairs of sequences. Inside a pool, one tasklet acts as a master: it has the charge of initializing the various buffers and the shared variables. Once this is done, it synchronizes the other tasklets in its pool at the granularity of the anti-diagonal computation. Each tasklet computes a different slice of the 3 anti-diagonal matrices. This parallelization scheme is possible because all the cells of an anti-diagonal can be computed independently (IV-B1). The traceback procedure being in essence a sequential process of navigation through the BT table, it cannot be parallelized.

4) Assembly optimisation: As stated in II-A the ISA has some specific instructions, those are not always exploited by the compiler. In order to achieve better performance manual optimization can be performed. Two exemples are: (1) using instructions adapted to the application domain, i.e. the *cmpb4* instruction a SIMD instruction able to simultaneously compare 4 bytes. This is particularly interesting when comparing DNA strings. (2) Fused instructions that combine arithmetic or logical operations together with control flow. Such instruction performs operation on its operands and depending of the result can jump to a predefined address.

Manual optimizations of the assembly code of the anti-diagonal computations yield a 30% improvement.

V. Experimentation

This section reports experimentations that have been conducted on an UPMEM server of 256 GB of standard memory plus 160 GB of PiM memory (40 ranks). The CPU processors are two Intel Xeon 4215 housing 16 cores each running at a top frequency of 2.7 GHz.

Two applications requiring intensive use of banded DP algorithms have been selected in order to analyze the gain provided by a PiM implementation. For both applications the number of pool P is 6 and the number of tasklets T is 4. Those numbers have shown good pipeline usage for both applications (95% to 99%). The way data are dispatched, as well as the way results are collected, depends on the application and are managed by the host. We compare the execution time with KSW2 library which provides a very efficient CPU implementation of the Needleman & Wunsch algorithm. KSW2 implements the Suzuki-Kasahara algorithm [8] and is a component of minimap2 [5].

A. 16S RNA sequence comparison for phylogeny

The first experimentation aims to compute a distance matrix used for phylogeny analysis. It consists in making a pairwise sequence comparison between all the sequences of a 16S Ribosomal RNA sequence data set. For each pairwise comparison the score is returned, the alignment (CIGAR) is not required.

A set of 16S RNA sequences have been extracted from the NCBI bacterial databases (August 2022) keeping only complete 16S sequences. The final data set contains 9557 16S Ribosomal RNA sequences.

To compute the comparison matrix efficiently on UPMEM PiM, all DPUs compute the same number of cells. The workload analysis shows that only a few DPUs have a lower computational load and none are overloaded; a perfect split would only improve performance by 5%. The data set is fully distributed to all DPUs thus reducing pre-processing and transfer footprint. Then, the indexes of the first cell of the comparison matrix and the number of cell are sent to each DPUs.

TABLE I
Accuracy of score.

	Static			**Adaptive**
band width	128	256	512	128
Accuracy (%)	70	81	85	86

Table I report the accuracy for different sizes of the diagonal band width. As it can be seen, the adaptive solution provides the same accuracy as the static formulation with four times fewer calculated cells. Accuracy is defined as the number of results matching the KSW2 score without any bands out of the total number of results.

Table II shows the execution time comparison between PiM and CPU implementation. With the same accuracy, the PiM implementation is 8.7 times faster than KSW2. The compute unit represent either the number of cores for CPU or the

number of DPUs for PiM. Execution time for PiM is reported for 3 configurations: 10, 20 and 40 ranks (5, 10 and 20 DIMM modules respectively).

TABLE II
COMPARISON OF COMPUTE TIME BETWEEN IMPLEMENTATIONS.

	CPU		DPU		
Width	512	128	128	128	128
Compute unit	32	32	640	1280	2540
Time (sec)	5882	1855	2585	1333	678
Speedup	1	3.1	2.2	4.4	8.7

B. Long read comparison for consensus sequence

The second experimentation compares many different sets of long reads before the construction of a consensus sequence. In a set, each read (or portion of reads) is expected to come from the same region of the genome. All possible pair in a set are aligned and the alignment (CIGAR) is needed for all of them. The objective is to compute a consensus sequence from the alignments.

The dataset is a list of set from a Pacific Bioscience sequencer and has 4814 sets of sequences with an average size of 20 Kbp. Each set is composed of 10 to 30 sequences that needs to be paired-aligned. In order to provide a realistic and time-consuming benchmark, and to fit all DPU with a substantial amount of data, the number of sets was quadruplet. The sets are dispatch to the DPUs after estimating their load, the goal is to send equivalent workload to all DPUs.

Table III details the accuracy for both static and adaptive diagonal band width. Adaptive band provides great improvement for retrieving the optimal path with smaller band width.

TABLE III
PERCENTAGE OF OPTIMAL PATH (CIGAR) RECOVERED.

	Static				Adaptive			
Width	128	256	512	1024	128	256	512	1024
Accuracy (%)	29	62	87	96	85	93	96	99

Execution times are reported in table IV. It shows similar results compared to V-A with a 8.4x improvement for DPU implementation over the x86 optimized KSW2 library.

TABLE IV
COMPARISONS OF KSW2(STATIC BAND ON CPU) AND DPU(ADAPTIVE BAND) ON EXECUTION TIME.

	CPU		DPU	
Width	512	128	128	128
Compute unit	32	32	1280	2520
Time (sec)	2386	698	507	284
Speedup	1	3.4	4.7	8.4

VI. DISCUSSION

The main target of PiM is data intensive applications as each DPU as its own bandwidth. Sequence alignment is generally considered a memory-bounded. However, the analysis performed with Vtune on the two use cases shows a high number of instructions per cycle and few memory stalls. The analysis does show a high number of memory operations (22%) which seems to have no impact on the overall performances. This is due to the fact that memory accesses are contiguous and do not solicit caches.

Despite sequence alignment being cache friendly, UPMEM PiM shows good performance against classical x86 architecture. Long-read technology is steadily developing, as shown by the introduction of the latest Illumina technology. As the error rate improves, adaptive band becomes even more attractive with a smaller bandwidth reducing the memory footprint. Our implementation shows that the first generation of UPMEM PiM can also accelerate cache-enabled applications, not just memory-related applications. The next generation of PiMs may prove to be even more effective for this task.

ACKNOWLEDGEMENTS

This research was funded, in whole or in part, by the French GenoPIM project (ANR-21-CE46-0012) and the BioPIM European Union's Horizon Europe programme for research and innovation under grant agreement No. 101047160.

REFERENCES

[1] J. Daily, "Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments," *BMC Bioinformatics*, vol. 17, no. 1, p. 81, Feb 2016. [Online]. Available: https://doi.org/10.1186/s12859-016-0930-z

[2] F. Devaux, "The true processing in memory accelerator," *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–24, 2019.

[3] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 11 2006. [Online]. Available: https://doi.org/10.1093/bioinformatics/btl582

[4] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022283682903989

[5] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 05 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty191

[6] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022283670900574

[7] H. Suzuki and M. Kasahara, "Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/09/07/130633

[8] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC Bioinformatics*, vol. 19, 2018.