DART-PIM: DNA read mApping acceleRaTor Using Processing-In-Memory

Appendix B - Determining Thresholds and Physical Sizes

Abstract

DART-PIM [1], a DNA sequence Alignment acceleRaTor using Processing-In-Memory, is a comprehensive framework for accelerating the entire read-mapping process. DART-PIM uses emerging memory technologies to execute all read-mapping steps inside the memory units, thus circumventing the data transfer bottleneck. This document for DART-PIM detail the fine-turnings that were done for determining DART-PIM parameters, including buffer sizes and thresholds, conducted to achieve optimal performance with a focus on high area utilization and minimal energy consumption. To achieve this, we employed the capabilities of DART-PIM full-system simulator and single-crossbar simulator.

A. Determining DART-PIM Thresholds and Physical Sizes

In this section, we present the fine-tuning process of DART-PIM parameters, including buffer sizes and thresholds, conducted to achieve optimal performance with a focus on high area utilization and minimal energy consumption. To achieve this, we employed the capabilities of the full-system simulator.

*1) **Fine-Tuning the Size of the Linear Buffer**:* The number of rows in the linear buffer corresponds to the number of maximum reference segments in the crossbar. Thus, reducing the linear buffer size increases the required number of crossbars. Assuming a fixed number of 256 rows per crossbar, an increase in crossbars translates to a larger memory footprint and higher energy consumption. To determine the optimal number of rows, our system simulator meticulously partitioned all reference segments into crossbars, as described in the paper, while varying the sizes of the linear buffer. This process is illustrated in the graph presented in Figure 1a. Additionally, the utility of the linear buffer, defined as the percentage of used (non-empty) linear buffer rows, is computed. This utility is calculated by dividing the total number of PLs by the total number of linear buffer rows (i.e., the number of crossbars multiplied by the number of rows in a single linear buffer).

As the size of the linear buffer increases, the size of the Reads FIFO decreases due to the fixed number of rows per crossbar (set to 256). A smaller Reads FIFO implies faster saturation of the FIFO, necessitating more linear WF iterations and subsequently increasing the total execution time. The execution time overhead, calculated for x buffer rows as $(t(x) - t_1)/t_1$, where $t(x)$ and t_1 represent the time with x rows and 1 row, respectively, is also presented in Figure 1a.

From the graph, it is evident that a linear buffer with 32 rows yields the lowest product of execution time and total memory capacity, achieving a balance between a compact area and minimal execution time overhead. In this scenario, approximately 8.37 million crossbars are required, achieving a buffer utility of 74.3%, with an execution time overhead of 11.5%.

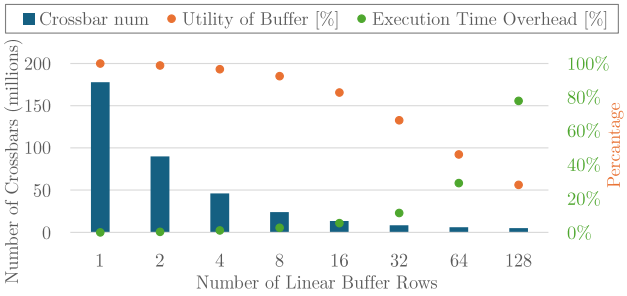
*2) **Fine-Tuning the Low Threshold**:* The low threshold (*lowTh*) plays a pivotal role in determining the execution strategy of minimizers within DART-PIM. When the number of PLs associated with a minimizer falls below or equals lowTh, the DP-RISC-V takes charge of the SW computations (only affine SW computations). Conversely, if the number of PLs surpasses the lowTh, the computations are performed in-memory (both linear and affine WF computations).

This threshold is essential due to the unique assignment of a different minimizer to each crossbar, rendering it inefficient to allocate a single crossbar for a small number of locations. Furthermore, this approach exploits the DP-RISC-V resources, which are not only responsible for the DP-memory computations but also leveraged for computational tasks. Additionally, storing these minimizers with few PLs in the DP-RISC-V conserves memory space.

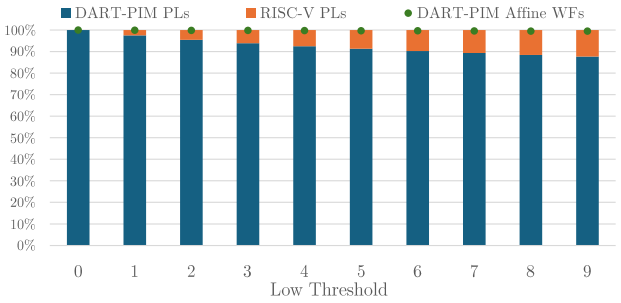
Increasing a lowTh results in more minimizers being handled by the DP-RISC-V. While Figure 1b demonstrates a visible rise in the percentage of DP-RISC-V PLs, the percentage of computed affine WFs by DP-memory consistently exceeds 99.5% across all simulated lowTh values. This observation aligns with the rarity of minimizers allocated to the DP-RISC-V in both reference and read datasets.

Additionally, a lowTh significantly influences the memory capacity utility of linear buffers. As the number of PLs associated with a minimizer decreases, the corresponding linear buffer rows are less utilized. This inefficiency is evident in Figure 1c. For DART-PIM, a lowTh value of three was chosen, resulting in 82.9% memory capacity efficiency for the linear buffer. Notably, 93.88% of PLs are allocated for in-memory execution, while 99.84% of affine WFs are executed in-memory.

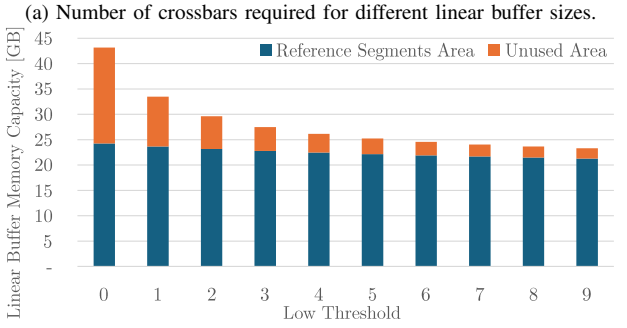
In addition, this selection impacts the size of the hash table used by the DP-RISC-V. With $7,660,814$ minimizers, each represented by a size of 24 bits, and $11,610,625$ PLs, each represented by 32 bits, the hash table size totals $69MB$, and is stored inside dedicated crossbars. This reduces computing memory by 50%, from $512GB$ with lowTh=0 to $256GB$ with lowTh=3, underscoring the memory efficiency achieved with the selected lowTh value.



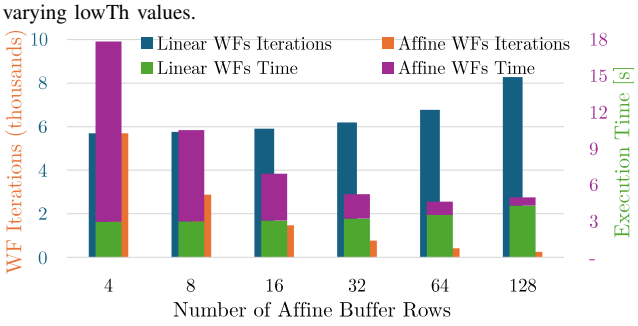
(a) Number of crossbars required for different linear buffer sizes.



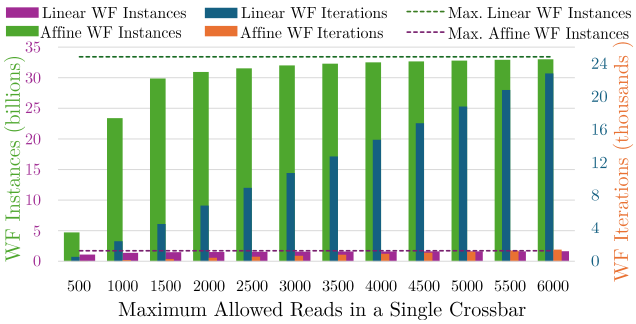
(b) Percentage distribution of reference PLs between the DP-RISC-V and DP-memory, and the percentage of affine WFs executed in-memory, for varying lowTh values.



(c) Total memory capacity utilization of all linear buffers for different lowTh values.



(d) Number of linear and affine iterations and execution time for varying affine buffer sizes.



(e) Number of linear and affine iterations and instances for different values of maxReads.

Fig. 1. Fitting the system parameters. Unless mentioned otherwise, lowTh=3.

TABLE I
SUMMARY OF CYCLE AND SWITCH COUNTS FOR EACH MEMORY OPERATION FROM THE SINGLE-CROSSBAR SIMULATOR DURING A SINGLE WF CALCULATION.

		MAGIC NOR	Writes	Reads	Total
Linear WF	**# Cycles**	254,585	4035	0	258,620
	# Switches	254,384	255,499	0	509,883
Affine WF	**# Cycles**	1,288,281	20,418	0	1,308,699
	# Switches	1,271,921	1,277,495	0	2,549,416

TABLE II
SUMMARY OF CONSERVATIVELY SCALED MAGIC NOR AND WRITE SWITCHING ENERGY AND CYCLE TIME FROM [2].

Operations	Energy/Time
MAGIC/write conservatively scaled cycle	2ns
MAGIC conservatively scaled energy	90fJ/bit
Write conservatively scaled energy	90fJ/bit

*3) **Fine-Tuning the Size of the Affine Buffer**:* The number of rows in the affine buffer significantly impacts DART-PIM's execution time. With an increase in the number of rows, fewer affine iterations are required since each row accommodates a single instance of affine WF computation, facilitating parallel execution. A higher number of affine buffer rows, however, results in fewer Reads FIFO rows, reducing the number of reads stored in each crossbar during linear iterations. Consequently, some crossbars may have no reads at all, diminishing the parallelism of linear iterations and necessitating more linear iterations.

The impact of varying affine buffer sizes on execution time is illustrated in Figure 1d, demonstrating the trade-off between affine and linear iterations. Leveraging cycle-accurate simulator results for both linear and affine WF iterations, as detailed in Table I, and accounting for conservative cycle scaling detailed in Table II, the total execution time for different affine buffer sizes is presented in Figure 1d.

We can state conclusively that an affine buffer with 64 rows achieves the minimum execution time for processing.

*4) **Fine-Tuning the Maximum Number of Reads Computed in a Crossbar**:* The distribution of minimizer appearances is non-uniform in both the reference and the reads. To mitigate the computational burden of aligning minimizers with numerous PLs in the reference, many aligners commonly disregard minimizers with more than $10K$ PLs. This has negligible repercussions on final results, as each read possesses multiple minimizers, enabling alignment using alternative candidates.

Similarly, in the reads, this phenomenon necessitates extensive computations during the calculation of WFs for all minimizers and reads. This significantly increases execution time, as a single crossbar in a certain iteration computes the WFs of the same read. Thus, the lower limit for the number of linear iterations is determined by the maximum number of appearances of a particular minimizer across all reads.

To address this, we introduced a maximum threshold for reads allowed per crossbar (*maxReads*). When a crossbar reaches the maxReads limit, no further reads are entered into its Reads FIFO. Similar to the threshold for the maximum number of PLs in the reference, maxReads optimizes execution time without compromising final accuracy.

As illustrated in Figure 1e, the number of WF iterations linearly increases with maxReads, while the number of WF instances follows a bounded growth function. The upper asymptote represents the total number of minimizers across all reads.

For instance, with $maxReads = 50k$, 94.7% of affine and 95.2% of the linear WF instances are computed. This configuration entails $255k$ linear iterations and $15.9k$ affine iterations, resulting in 328 billion linear WF instances and 16.2 billion affine WF instances. Consequently, during a single linear iteration, 1.29 million linear WF instances and 1.02 million affine WF instances are executed in parallel, on average.

In conclusion, the parameters chosen for all subsequent results are as follows: linear and affine buffer sizes are respectively set to 32 and 64, the low threshold (*lowTh*) is configured to 3, and the maximum allowed reads per crossbar (*maxReads*) is established at $50k$. These parameter settings were optimized to achieve a balance between execution time and area in the subsequent analyses.

REFERENCES

[1] Anonymous, "DART-PIM: DNA read mApping acceleRaTor Using Processing-In-Memory," in Proceedings of the 57th Annual IEEE/ACM International Symposium on Microarchitecture (submitted), 2024.

[2] M. S. Truong, E. Chen, D. Su, L. Shen, A. Glass, L. R. Carley, J. A. Bain, and S. Ghose, "RACER: Bit-pipelined processing using resistive memory," in MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021, pp. 100–116.