17.3 GCUPS Pruning-based Pair-Hidden-Markov-Model Accelerator for Next-Generation DNA Sequencing

Xiao Wu[1,2], Arun Subramaniyan[1,2], Zhehong Wang[1], Satish Narayanasamy[1,2], Reetu Das[1,2], David Blaauw[1,2]
[1]University of Michigan, Ann Arbor, [2]Sequal Inc., Ann Arbor

Abstract

We present the first ASIC accelerator for pair-Hidden-Markov-Model (Pair-HMM) in DNA variant calling, which conventionally requires ~250T FLOPs per sequenced human genome. Using a hardware-algorithm co-design, we opportunistically replace floating point (FP) multiplication with 20b log-domain addition while employing bound checks to maintain (provable) correct results in downstream processing. FP computation is reduced by 43× on real human genome data. Implemented in 40nm CMOS, the 5.67 mm^2 accelerator demonstrates 17.3G cell updates per second (CUPS) throughput, marking a 6.6× improvement over our baseline ASIC implementation and 355× GCUPS/mm^2 improvement over a recent FPGA implementation [2].

Introduction

Recent advances in next-generation sequencing have enabled fast DNA identification for cancer, genetic disorders and pathogen detection. Short DNA fragments are sequenced in a massively parallel fashion, producing billions of DNA reads (strings of ~100 nucleotides: A, C, G, T) per human genome. Reassembling these DNA fragments to determine differences from a common reference genome (secondary analysis) requires extensive computation. Among several secondary analysis steps, variant calling is the final step, which identifies disease-related gene mutations and remains extremely time consuming. In particular, the Pair-HMM forward algorithm (or PFA) for variant calling requires ~250T FLOPs to infer mutation probabilities and contributes 70% [1] of variant calling computation. The PFA requires alignment matrix calculation with a complicated combination of floating point (FP) addition and multiplication to infer overall similarity of two strings, making it a difficult hardware optimization problem. The PFA has been mapped to a GPU [8] as well as FPGAs using systolic array [3-6] and ring-based topologies [2][7]. However, these methods are constrained by the availability of FP resources. Also, they are direct hardware mappings that do not re-optimize the algorithm to make them hardware friendly. Furthermore, no dedicated ASIC has been demonstrated to date to accelerate the PFA for DNA sequencing.

We propose the first ASIC implementation of the PFA and demonstrate its operation in a complete DNA sequencing flow. Using a new algorithm-hardware co-design, we alleviate the throughput bottleneck caused by limited FP resources by devoting FP resources only to estimated high-quality alignment regions (Fig. 1). For non-high-quality regions, FP multiplication is replaced with low-accuracy 20b integer *addition* by using log domain number representation greatly improving performance. By maintaining error bounds on this low-precision log-domain calculation, we detect and, as needed, rerun with high precision those few cases with possible failure, thereby obtaining (provable) correctness in the downstream analysis. As a result, FP computation is reduced by 43× when tested on real human data, and is replaced with 20 bit log-domain integer PEs (I-PEs) that are 4.6× smaller in area and 1.9× higher in performance than floating point PEs (FP-PEs).

Pruning-Based Algorithm-Hardware Co-Design

A conventional PFA calculates the probabilities of all alignments between a candidate mutation string and a DNA read using an alignment matrix (Fig. 1). The likelihood of a particular cell (i,j) representing a particular alignment is computed from the likelihood of its three neighboring cells: vertical neighbor cell (i-1,j) representing an insert transition, diagonal neighbor (i-1,j-1) for a match transition, and horizontal neighbor (i,j-1) for a delete transition (Fig. 1, bot, right). We make the key observation that the final score of the matrix is typically dominated by the probabilities of only a few alignment paths, thanks to high quality reads and a small likelihood of genetic mutations. To identify these dominant paths, the proposed pruning-based Pair-HMM algorithm executes in two phases (Fig. 2).

In the *scan* phase, an upper bound likelihood for each cell in the alignment matrix is computed using I-PEs operating in logarithmic number representation, which replaces multiplication with addition and significantly reduces hardware complexity. We observed that most dominant paths contain long consecutive diagonal transitions. Therefore, starting from the cell with the maximum matched score in the final row, the algorithm traces back along diagonal cells, pruning horizontally and vertically adjacent cells (i.e., treating their likelihood as zero). It continues this trace as long as the pruned cells, representing delete/insert transitions, do not contribute significantly to the cell score. The result of the first phase is a pruned region as well as an upper bound of the final score of the entire matrix.

In the following *refinement* phase, the (small) un-pruned region is computed using FP-PEs (the likelihood of the pruned region is set to zero), resulting in a (tight) lower bound. Then, through bound comparison in a downstream analysis, the mutation with the highest likelihood among all candidate mutations is correctly determined as the final variant calling result in 98.5% of all cases. The failing cases (1.5%) are guaranteed to be identified and are recomputed using only FP-PEs.

Fig. 3 shows the pruning method in the scan phase in more detail. The cell (I, J) with the highest match score in the final row indicates the end position of a good alignment and is picked as the seed position for pruning. In the PFA, the match score at the seed M(I, J) is the weighted sum of M, I, D from the diagonally adjacent cell (I-1, J-1). If this match score is significantly larger than the insertion and deletion scores in cell (I-1,J-1), we would arrive at a slightly lower M(I, J) score with insertion and deletion scores set to zero in cell (I-1,J-1). This allows us to prune insertion and deletion scores in (I-1, J-1) and all their adjacent cells to the top and left, respectively. The pruning stops at the cell (Istop, Jstop) where the match no longer shows dominance over the insertion and deletion scores. For FP-PE computation, this results in a (small) rectangular region in the top, left of the matrix, followed by a string of consecutive match transitions (Fig. 3, shown in red).

Hardware Implementation

Fig. 4 shows the overall architecture of the proposed pruning-based Pair-HMM accelerator. It consists of 10 scan machines composed of 16 I-PEs each to upper bound and prune matrices, and 4 refinement machines composed of FP-PEs to accurately compute un-pruned regions. Refinement machines come in two sizes with 1× and 4× FP-PEs to accommodate the variable size of un-pruned regions. An on-demand arbiter streams in jobs from input memory, dispatches them to scan and refinement PEs and streams results to output memory.

Fig. 5 shows the hardware implementation of a scan machine, consisting of 16 PEs (Fig. 6), an input feeder to control PE traversal across the matrix, a binning-based log-sum module to avoid accuracy degradation in the last row, and an early stop detection module. Each PE uses 20 bits fixed point addition and a 15-entry table lookup in the log domain as substitutes for multiplication and addition, respectively, in the real domain. Instead of tracing back to determine the pruned region, the logic in the PEs prune cells as PEs traverse forward across the matrix, avoiding the need to store scores for the entire matrix. PEs work in parallel when traversing the matrix from left to right. As PEs traverse, an early detection module opportunistically stops the scan phase once the maximum score in one row is smaller than a threshold. This optimization takes advantage of downstream processing where extremely low Pair-HMM results are filtered completely, reducing workload even in the scan phase (by 18%). Because only adjacent PEs communicate with each other, routing complexity is greatly reduced.

Measurements

Fabricated in 40nm CMOS with 5.67 mm^2 die area, the accelerator reaches 120 MHz with 756 mW. Fig. 6 shows that the number of cells requiring FP calculation is reduced 43× (including re-computation due to bound check failure). The proposed accelerator was verified with real sequencing data and shows 17.3 GCUPS average throughput which is a 6.6× improvement over a FP-only baseline ASIC implementation (normalized to the same area). We obtain speedups of 355×

and 1344× in CUPS/mm^2 compared to FPGA [2] and NVidia K40 GPU [8] implementations, respectively. Fig. 7 shows the die photo.

References

[1] M. Carneiro, Intel, 2013.
[2] S. Huang, et.al, FPGA, 2017.
[3] C. Rauer, et.al, Intel.
[4] J. Peltenburg, et.al, BIBM, 2016.
[5] S. Ren, et.al, BIBM, 2015.
[6] M. Ito, M. Ohara, COOL chips, 2016.
[7] G.J. Manikandan, et.al, FCCM, 2016.
[8] S. Ren, et.al, Evol. Bioinform, 2018.

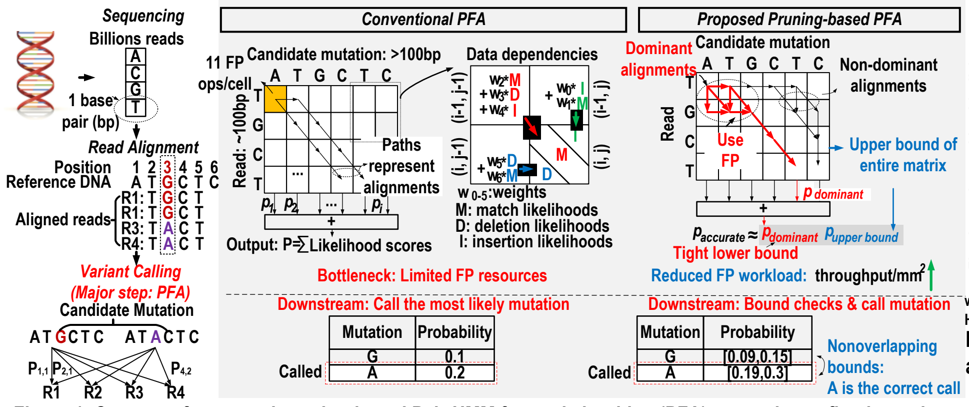


Figure 1. Concept of proposed pruning-based Pair-HMM forward algorithm (PFA) to accelerate floating point intensive (~250T FP operations/genome) DNA variant calling

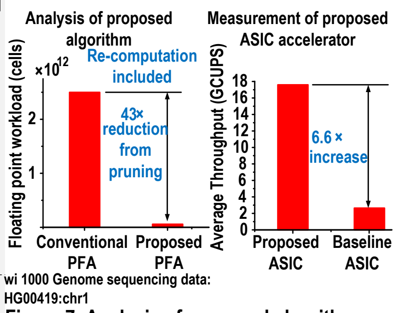
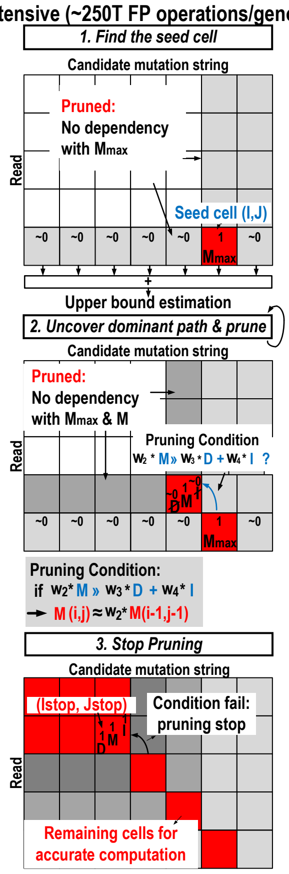
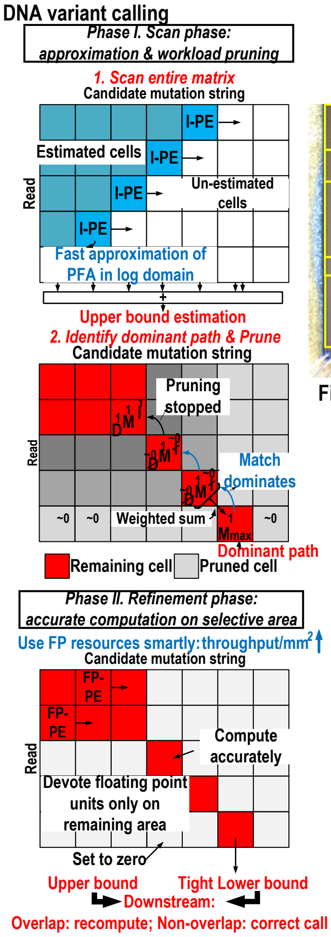


Figure 7. Analysis of proposed algorithm and measurement result of proposed ASIC



1. Find the seed cell
Candidate mutation string
Pruned: No dependency with Mmax
Seed cell (I,J)
Upper bound estimation
2. Uncover dominant path & prune
Pruned: No dependency with Mmax & M
Pruning Condition
Pruning Condition: if w2* M » w3* D + w4* I → M (i,j) ≈ w2* M(i-1,j-1)
3. Stop Pruning
(Istop, Jstop)
Condition fail: pruning stop
Remaining cells for accurate computation

Figure 3. Detailed pruning methodology of proposed pruning-based PFA



Phase I. Scan phase: approximation & workload pruning
1. Scan entire matrix
Candidate mutation string
Estimated cells
Un-estimated cells
Fast approximation of PFA in log domain
Upper bound estimation
2. Identify dominant path & Prune
Pruning stopped
Match dominates
Weighted sum
Dominant path
Remaining cell
Pruned cell
Phase II. Refinement phase: accurate computation on selective area
Use FP resources smartly: throughput/mm² ↑
Compute accurately
Devote floating point units only on remaining area
Set to zero
Upper bound
Tight Lower bound
Downstream:
Overlap: recompute; Non-overlap: correct call

Figure 2. Algorithm of proposed pruning-based PFA

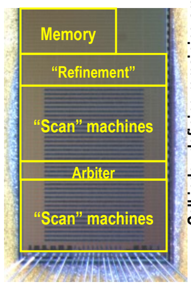


Figure 8. Die photo

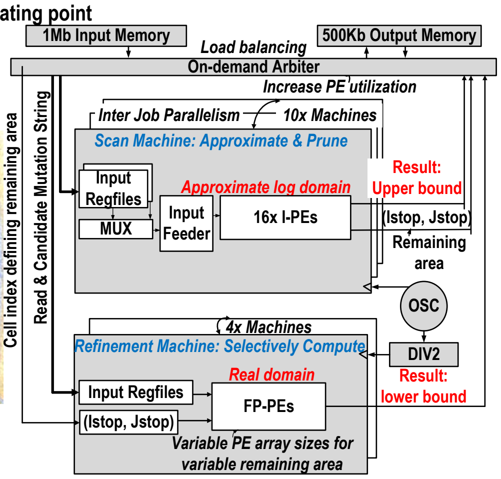
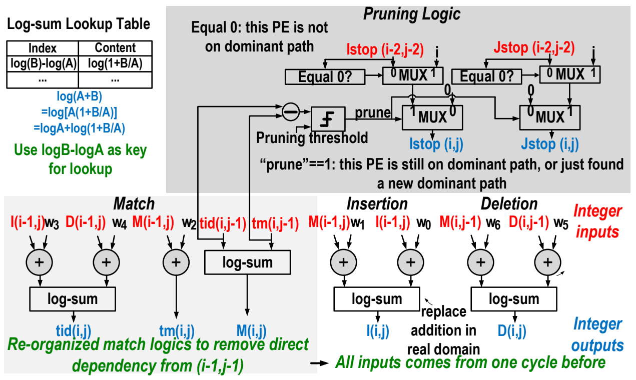


Figure 4. Top-level architecture of pruning-based PFA accelerator



Log-sum Lookup Table
Index | Content
log(B)-log(A) | log(1+B/A)
log(A+B) =log[A(1+B/A)] =logA+log(1+B/A)
Use logB-logA as key for lookup
Pruning Logic
Equal 0: this PE is not on dominant path
Istop (i-2,j-2)
Jstop (i-2,j-2)
Equal 0?
MUX
prune
Pruning threshold
Istop (i,j)
Jstop (i,j)
"prune"==1: this PE is still on dominant path, or just found a new dominant path
Match
Insertion
Deletion
Integer inputs
log-sum
tid(i,j) tm(i,j) M(i,j)
I(i,j) D(i,j)
replace addition in real domain
Integer outputs
Re-organized match logics to remove direct dependency from (i-1,j-1)
All inputs comes from one cycle before

Figure 6. Circuit implementation of log-domain I-PE

Table 1. Performance comparison

	[2]	[4]	[8]	[8]	This work
Platform	FPGA Arria 10	FPGA Virtex 7	CPU POWER8 (20 cores)	GPU Nvidia Tesla K40	ASIC
Technology	20nm	28nm	22nm	28nm	40nm
Chip area (mm^2)	858*	858*	1298	561	5.67
Topology	PE ring	Systolic array	N.A.	N.A.	Pruning-based
Frequency (MHz)	230.73	166.7	3420	745	120
Power (W)	20***	20***	851	235	0.756
Peak floating point performance (GFLOPS)	1366	2000	1094.4	4290	6.9
Throughput normalized to floating point perfoamance (CUPS/FLOPS)	0.0216	0.0113	0.0024	0.0006	2.5072
Throughput normalized to area (GCUPS/mm^2)	0.0086**	0.0129**	0.0006**	0.0023**	3.0511
Power efficiency (GCUPS/W)	1.4750	1.1300	0.0031	0.0111	22.8836

* Only reticle size was available ** Normalized to technology node *** Extracted from different benchmarks

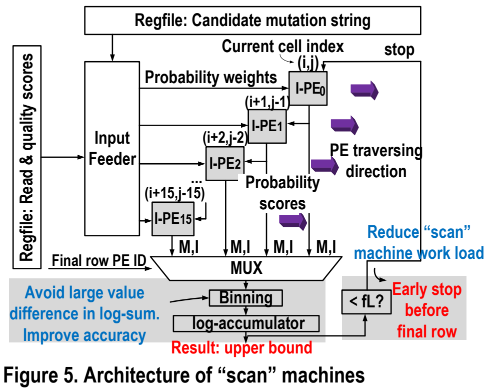


Figure 5. Architecture of "scan" machines