Energy Efficient Computation with Asynchronous Races

ABSTRACT

Race Logic encodes information as timing delay, rather than conventional logic levels, which cause some basic processing operations to become trivial to implement. The computation can then be observed by relative timing differences between injected signals. We propose an asynchronous version, a dramatic improvement over synchronous Race Logic and state of the art systolic implementations, by utilizing current starved inverters as delay elements. We chose the well-studied DNA sequence alignment problem for comparison and we show that, for the synthesized design in 0.18 μm process, even despite variations the asynchronous Race Logic is 10× more energy efficient and 4× denser at comparable speeds as compared to synchronous implementation.

1. INTRODUCTION

With the end of Dennard scaling, computer architects are faced with a very interesting problem. While the pace of Moore's law is still causing miniaturization of transistors, continually allowing us to pack more transistors on a chip, our ability to power them all at once is being curtailed. This problem, known as the dark silicon problem, is pushing general purpose computing in the form of multicore processors out of the way to make way for application specific accelerators to keep up with the tremendous rate of performance scaling that we are used to [1]. This approach of relaxing the constraints of general purposeness looks promising as it opens up a whole new spectrum of computation that has the potential to provide orders of magnitude of performance and energy advantages.

Approaches to specialization involve non-traditional digital encodings such as logarithmic encoding to speedup multiplication and division [2], or reducing complex computations into simple repeatable steps that can be performed iteratively such as the CORDIC algorithm [3]. Some technologies go as far as to forgo logical completeness for the sake of energy efficiency by utilizing the dynamics of the material system involved to perform complex computations [4]. Most of these approaches toy with information representation and manipulation to make the set of operations that are performed simple in the platform they are implemented in, often leading to wins in both performance and energy efficiency.

In this paper we propose a novel asynchronous implementation of Race Logic, an information processing paradigm, that was first reported in [5]. Race Logic utilizes a novel data representation to accelerate a broad class of optimization problems like the ones solved by dynamic programming algorithms. The core idea of Race Logic is to deliberately engineer race conditions in a circuit to perform useful computation. Information, instead of being represented as logic levels as is done in conventional logic, is represented as a *timing delay*. Computations can then be performed by observing the relative propagation times of signals injected into a configurable circuit, i.e. the outcome of races through the circuit.

Our specific contributions are:

- We design, for the first time, a full custom, synthesized, asynchronous Race Logic array in 0.18 μm technology that can report similarity between 50-symbol-long sequences and simulate its performance against the well-studied DNA sequence alignment problem.
- We quantify the effect of process variations both from a device and system level standpoint and show that at least for the considered application process variations have minimal impact on the performance and functionality of the asynchronous Race Logic.
- We show that the asynchronous Race Logic architecture is at best 10× more energy efficient, and 4× denser compared to previously reported synchronous and state of the art systolic implementations, while also being faster.

2. BACKGROUND

2.1 Main Idea of Race Logic

Race Logic encodes information in a *timing delay*. To understand how this might ever be useful, to begin let us consider the problem of finding the longest path through a directed acylic graph (DAG) as shown in Fig. 1.

Each of the edges in the DAG is labeled with a weight, and the longest path is of course simply the path from an input to output walking through the largest sum of edge weights. So how can a race condition compute this? The answer is to make each weight a *delay* and make each node wait around for the *last* of it's inputs to arrive before informing the nodes downstream. The simplest possible implementation achieving such operation is one where we enforce a set

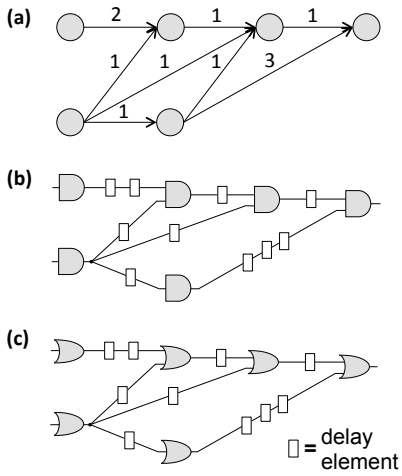


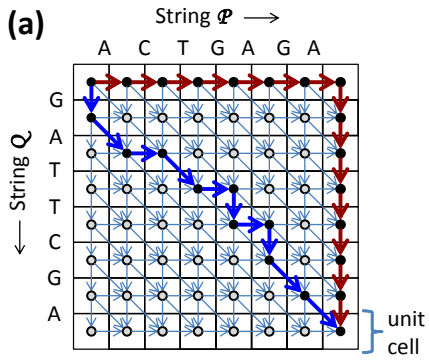
Figure 1: Example conversions of a DAG operation into a race-based computation. (a) An example of a DAG with weighted edges, (b) AND-type race computation calculating the longest path, (c) OR-type computation calculating the shortest path. The rectangular blocks represent delay elements that can be implemented in either a synchronous or asynchronous way.

of unit delays on the edges using flip-flops [5], and perform the "wait for last" operation with a single AND gate. The system starts in a state of all 0, and a "race" is begun by setting the inputs to 1 at the same time. The AND gate will of course not propagate a 1 downstream until *all* inputs have arrived and are set to 1. The "score" of the node is now equivalent to a time it takes for the signal (which is typically injected at the root node) to propagate down the graph to that node in question and the score of the last node is simply the longest path. An equivalent shortest path computation can be performed by replacing the AND gates with OR gates that perform the "pass the first" operation. Hence MIN-MAX computations are performed using OR/AND gates and ADD-BY-CONSTANT is performed by delay chaining.

The computation considered in Fig. 1 is certainly too simple to justify the use of Race Logic though it points to the potential computational power of such a simple system. Such DAGs (albeit more complex than the ones in Fig. 1a) are well known as visualizations/representations of Dynamic Programming algorithms, and finding shortest/longest paths are fundamental to a variety of applications including beat tracking in music information retrieval systems, dynamic time warping for measuring similarity between time series and seam carving. Measuring similarity between DNA strands is one particularly well studied application with industry demands such a low energy, high throughput and cost effectiveness. A more practical Race Logic architecture suitable for solving such string similarity tasks is presented in the next section. Because such architecture is application specific, we will first briefly discuss an application itself.

2.2 DNA Sequencing

Today's next generation sequencing methods churn out about a billion bases in a single run (each of length 25-100 bases) and millions of these sub-strings are required to be compared against each other. As the technologies for DNA sequencing



	A	C	T	G	_
A	1	4	4	4	3
C	4	1	4	4	3
T	4	4	1	4	3
G	4	4	4	1	3
_	3	3	3	3	-

Figure 2: (a) Example of DNA sequence alignment DAG (edit graph) for two particular strings P = "ACTGAGA" and Q = "GATTCGA" and (b) typical score matrix useg in DNA sequence alignment, transformed for race formulation.

continue to improve, the challenge is now in the processing of the vast amount of information. A typical bottleneck operation, whether it is in reference-assisted or de-novo DNA sequencing [6], is solving approximate string matching problem. In a simplest case, it is a problem of finding a similarity metric, also known as edit distance, between a query string P and a reference string Q when both are of length N. To compute the edit distance, a two-dimensional DAG, an edit graph, is constructed, which is essentially a two-dimensional representation of all the possible alignments between the two input sequences as shown in Fig. 2a. Any specific alignment is just a path in this graph where every edge corresponds to an edit operation. In particular, the vertical arrows represent insertions, horizontal arrows represent deletions and diagonal arrows represent matches.

To determine the relative merit of one particular alignment over another, e.g. to choose the alignment with the maximum number of matches, the concept of a score matrix (Fig. 2b) is introduced, which effectively defines the weight for each edge in the edit graph. Determining the "goodness" of the alignment is therefore finding either the shortest path in the graph in which a match is assigned a lowest score in the score matrix, or, conversely, the longest path where matches are rewarded with high scores in the score matrix.

3. ASYNCHRONOUS RACE LOGIC

In the race formulation of the edit graph, as explained above, a rising signal navigates the mesh of timing delays and the computation performed is directly affected as a result of the differences in the timing of different paths. The signal propagation time affects the final result, while the effect of those timing differences is unknown ahead of time. While such behaviour might be similar to classical software race condition, note that in Race Logic "race conditions" are purposely introduced and are not non-deterministic in the circuit sense. The way the signal races are resolved tell us about the degree of similarity between two sequences.

3.1 Top-Level Architecture

Similar to the work in [5], in our attempt to design asynchronous Race Logic circuit for DNA sequence alignment problem we make the most of its repetitive nature and partition its implementation into unit cells. The unit cell for an OR-type Race Logic is shown in Fig. 3a. There are three inputs - top, left and diagonal, which receive a rising edge

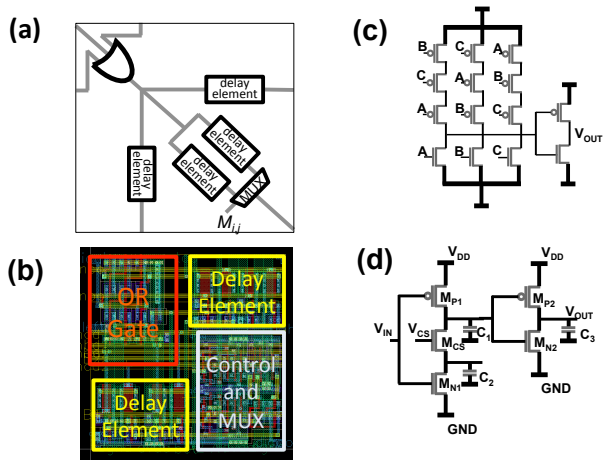


Figure 3: Asynchronous Race Logic implementation of DNA sequencing alignment problem: (a) Unit cell implementation,(b) layout of the unit cell shown in panel (a), (b) OR gate design ensuring equal delay propagation(note: all transistors are minimum size), and (c) simplified non-inverting delay element (real element also has cascode bias). In panel (a) M_{ij} is the output of the control circuit that determines the match/mismatch condition and chooses the required delay path based on the score matrix values.

from the preceding and adjacent cells, and the first arriving input is detected by an OR gate. The symmetric gate design shown in Fig. 3b was used to aim for equal delay on all "input to output" paths. The first arriving signal then passes through three delay elements, the top and bottom being always a constant delay for the considered application (representative of a insertion/deletion) and the diagonal one, selected through the multiplexer, being dependent on the match mismatch criterion.

The values of delay elements are set according to the particular score matrix. Though having fixed delays might be acceptable in some cases [5], in this paper we consider a more general case when the delay value can be programmed, thus allowing asynchronous Race Logic to solving a broader variety of alignment problems.

Since Race Logic encodes information in timing delay, the choice of delay element is critical to the performance of the architecture. Though synchronous delay elements such as D flip-flops can be clocked at very high speeds and are less sensitive to process variations, they are area and energy intensive [5]. We next describe the implementation of programmable delay elements and other design decisions for ensuring accurate timing in the asynchronous Race Logic architecture.

3.2 Delay Element

One of the goals of this work is to design Race Logic in which timing of delay elements is not determined by a clock but by its own intrinsic delay. Ideally, such a delay element should be controllable to account for an order of magnitude of dynamic range, required for more general purpose graph traversal applications, as well as be tolerant to supply, process and mismatch variations.

Different schools of programmable delay elements such as [7, 8, 9, 10, 11] were investigated. Capacitive control options like the ones discussed in [7, 8] do not seem promising due to square law scaling with respect to dynamic range of the delay, which leads to prohibitively large area costs. Problems of charge sharing in complex pull-down networks as shown in [11] are addressed in [10] but suffers from a lot of charge injection noise. We found that a cascode current source, controlled by a variable resistor is best suited for our application [12].

To elaborate more on the choice of delay element let us consider Fig. 4. The standard way of constructing a delay element is by charging a capacitor with a controllable current and using a thresholding element to detect a voltage crossing. This is generally implemented using an inverter and controlling the current that is charging its output capacitance and is known as a current starved inverter. Transistors M_N and M_P behave like digital switches while the transistor M_{CS} is the current control transistor that discharges the output capacitance at a fixed rate, hence giving the required constant delay. The design decision at this point is the placement of the current control transistor. In Fig. 4a the current control transistor is placed in the discharge path with its drain connected to the source of the NMOS switch, while in Fig. 4b the current control transistor splits the output node of the inverter.

Though the two topologies seem similar, transient simulations reveal that charge sharing due to switching of M_n causes considerable differences in both timing and variability. In the former case, the rising edge switches transistor M_N into the linear region which causes charge sharing between C_{out} and C_{int} causing the output voltage to drop instantaneously (not controllably) to $V_{DD} \times C_{out}/(C_{int} + C_{out})$. This voltage is then discharged by the current control transistor M_{CS}. In the latter case, the rising edge causes C_{int} to be discharged through M_N quickly, while the output voltage stays at V_{DD}. This is then followed by a controlled discharge of the output node through M_{CS}.

Not only does the latter method produce longer and more controllable timing delays for the same bias, but is less susceptible to mismatch variations as well. This is due to the fact that output voltage starts discharging from V_{DD} in the latter case vs $V_{DD} \times C_{out}/(C_{int} + C_{out})$ in the former. The capacitances C_{int} and C_{out} are subject to mismatch variations and hence add uncertainty to the delay of the element.

It is worth noting that Fig. 4 shows only a single bias node for ease of explanation. In reality, the current control is performed by both bias and cascode nodes with optimal biasing to ensure minimal current variation with V_{out} during the discharge phase. The digitally switching transistors, M_N and M_P are minimum sized, while the bias and cascode current control transistors are sized to match the current mirrors in the current source.

3.3 Current Source

The precision of the delay element, barring its own process and mismatch variation, depends upon the precision of the current that is discharging the output capacitance. Hence the current source has a very important role to play. It should be tolerant of process variations and should provide a constant current independent of any power supply variations that may occur due to injection from digital switching activity. Moreover, the current source should be a variable one, preferably controlled by an external source, such that variety of timing delays can be implemented. For the specific case of comparison against work in [5], the dynamic range of the delay does not need to be very large, but for implementation with real data (e.g. DNA nucleotide sequences

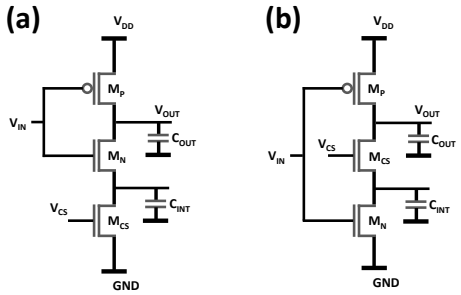


Figure 4: Current starved inverter topologies for variable delay generation with (a) control transistor at the bottom of the inverter stack and (b) control transistor splitting the output of the inverter.

from NCBI [13]) the dynamic range needs to vary by about an order of magnitude, hence requiring the current source to produce low variability currents over such ranges.

In this regard an op-amp based current source was designed that pins a fixed voltage across an off-chip variable resistor, hence utilizing the resistor to act as the current control element. This makes the design largely independent of process and supply variations. The bias voltage thus generated is then redistributed across all the required delay elements. Depending upon the number of independent delays needed (which is three for the considered case), replicas of the current source are made with different resistances, each of them variable, to tune to the required current. Another important aspect of the current source design was to size the bias node transistors relatively large, such that they would behave as low impedance nodes and be more tolerant to charge injection from switching activity from nearby digital nodes.

The issue of charge injection into the bias nodes is a serious one and can cause large systematic errors in the delay of the circuit if not resolved. The $V_{\rm cs}$ node of the delay elements (Fig. 3c) are shared by the entire array and are hence subject to charge injection from the switching activity in the bottom NMOS transistor. Though the entire array does not switch repeatedly and simultaneously (which would cause a large simultaneous charge injection event), there is switching activity which is staggered by the delay elements themselves. Here we suggest two solutions, which are referred to as *local* bias and *global* bias, to overcome this problem. The idea behind global biasing strategy is to use the area above the entire array by high-quality MIM caps (4 fF/μm^2) and bypass the bias nodes. The large value of the capacitance significantly reduces variation in voltages from the injected charge. Another, albeit more power hungry (local) solution is to partition the array into blocks (e.g. 5×5 subarrays used in this work) and regenerate the bias voltages locally for each of these blocks while capping them with the aforementioned MIM caps. This significantly reduces the load on the primary bias network and decouples large sections of switching activity from each other and hence provides a more precise delay values.

4. RESULTS AND DISCUSSION

We have designed asynchronous Race Logic implementation for solving DNA sequence alignment problem for variable string lengths from $N = 5$ to $N = 50$. Data points were fitted to analytical equations for all desired metrics shown in Fig. 5. All the simulations were done in Cadence 6.1.0 using the Silterra 0.18 μm process. Though the implemented delay elements are programmable with roughly $10\times$ dynamic range, in the simulations we used specific score matrix in Fig. 2b.

4.1 Simulation Results

For the purpose of comparison, similar to synchronous Race Logic studies [5], we focus on two possible alignments resulting in the best and the worst case scores, which represent a perfect match and a complete mismatch, respectively. (The complete mismatch case results in $2N$ indels for a string length of N and rather unlikely in practical applications - see discussion in the next subsection.) Moreover, Dennard constant-field scaling laws were applied to make sure that the work in [5], which was performed in 0.5 micron, was scaled down to effectively be in the same process. In addition, previously reported performance results for the synchronous Race Logic were adjusted to account for larger dynamic ranges of the delays in the asynchronous Race Logic.

In particular, the following changes were made to the synchronous architecture. For the dynamic range to be increased from a factor of two in [5], to an order of magnitude, the number of D flip-flops has to be increased from 1 to 4 (assuming logarithmic encoding), which in-turn adds extra gates for score selection and latching. The latency is also affected as a result of addition of extra logic levels in the critical path between two D flip-flops, which manifests in the slowing down of the clock. Another repercussion of each unit cell housing 4 DFFs is in the power and energy, because the clocked capacitance [5] dominates the energy scaling of the synchronous architecture. The energy numbers that are reported for the synchronous design are after clock gating strategies, that attempt to reduce the third order energy scaling that exists as a result of continuous clocking of the entire Race Logic fabric. Nonetheless, the cubic scaling can be reduced but not completely eliminated.

Fig. 5 shows various performance metrics for asynchronous Race Logic and compare it with previously reported adjusted results for synchronous Race Logic. In general, even with pessimistic assumptions regarding the increase in area and energy of the synchronous design, asynchronous Race Logic performs better in almost all metrics (and as a result significantly outperforms highly-optimized conventional implementation [5]) except for power density (Fig. 5d), which is due to larger area of the synchronous architecture. The global biasing implementation is much more energy efficient and area efficient compared to local bias one. This is because in the global bias case, the distribution of timing information happens in terms of bias voltages which are gate-connected and hence consume negligible power, while the MIM caps supply any instantaneous charge required. In the local bias case, once the global bias voltages are distributed, local biases are regenerated for 5×5 array regions, which contribute to the cubic scaling of energy. This behavior is visible in the Fig. 5c as the global bias case has square law energy scaling and is considerably lower than the scaled synchronous as well as the local bias case which have cubic behavior.

One possible concern is that performance and energy efficiency advantages may be lost when implementing circuits with more aggressive and more practical CMOS process nodes, in particular because asynchronous Race Logic might be more sensitive to process variations due to its inherently

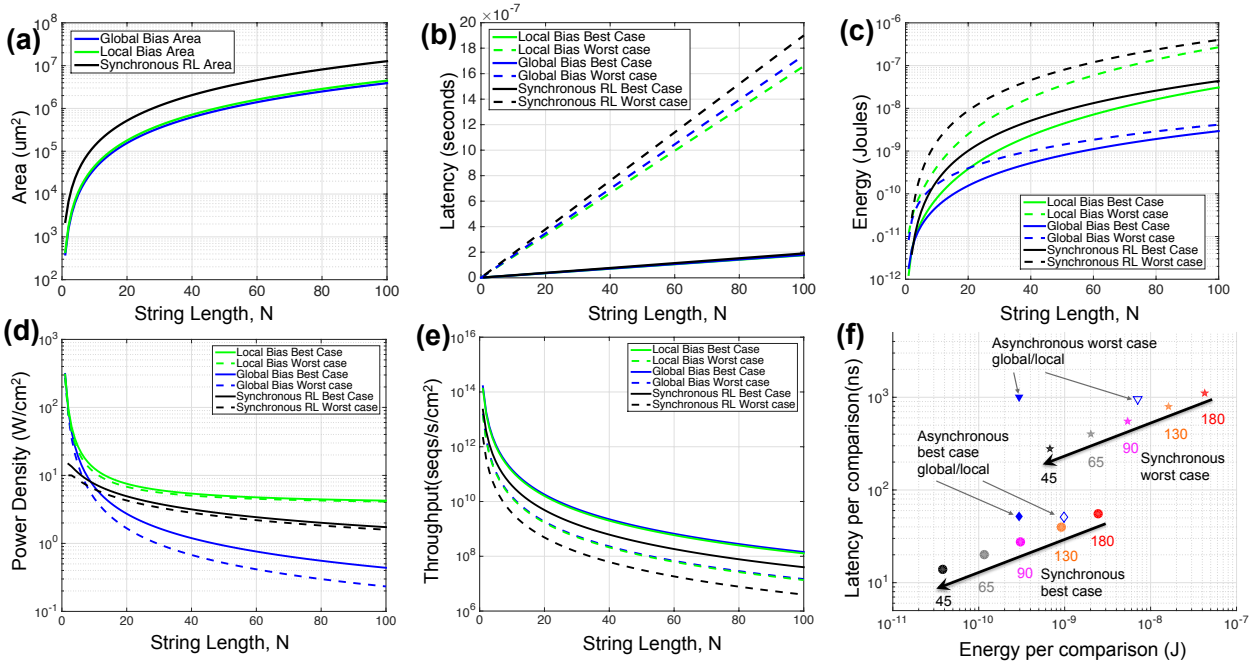


Figure 5: (a) Area, (b) latency, (c) energy per comparison operation, (d) power density, and (e) throughput as a function of string length N for synchronous and asynchronous Race Logic. (f) Energy-delay scatter plot for DNA string length $N = 30$, showing scaling for Synchronous Race Logic vs Asynchronous. The points following the arrows show the best and worst case results for the scaled synchronous Race Logic at different technology nodes.

analog design, compared to a purely digital one. To address this concern we first note that the relative area of the circuitry, that is sensitive to mismatch variations in asynchronous Race Logic, is approx. 30%, and therefore such circuitry can be aggressively scaled without increasing variations. To further compare scaling behaviour, Fig. 5f shows the estimated performance for scaling of the synchronous design relative to the 180 nm asynchronous design in this work. The latency estimates assume Dennard scaling, while the energy estimates also account for leakage power at low technology nodes. Though latencies of scaled synchronous versions always outperform the asynchronous version, the energy performance of the 180 nm asynchronous version is comparable to the energy performance between 90 nm and 130 nm for the best case and between 45 nm and 90 nm for the worst case. Moreover, in the next subsection we show that for the considered application variations in delay can be effectively tolerated at the functional level without much penalty in performance or energy efficiency.

4.2 Variation Analysis

To understand how the Race Logic implementation of the edit graph is affected by the delay variations in each of its elements, and how it affects the functional correctness of the overall architecture, a MatLAB model of the circuit architecture was designed into which variations were purposefully introduced.

To make sure that our assumptions were realistic, we took real data from the chromosome 1 of the Human Genome and simulated the process of shotgun sequencing followed by realignment Fig. 6a, which is a typical procedure in the de-novo sequencing [6]. In shotgun DNA sequencing, a section of the DNA sequence is split in into thousands of strands of equal length at random locations. The strands can overlap which defines its coverage. Fig. 6c shows an example of score histogram when comparing a particular DNA string with others in a given section with an overall number of nucleotides being 20 times the number of nucleotides in the section (coverage of 20).

We then use Monte Carlo technique to simulate signal propagation timing (i.e. score) in the Race Logic fabric for two pairs of DNA strings (Fig. 6b). In each run of Monte Carlo simulation, delay elements were initialized by adding normally distributed random variable with specific standard deviation to the exact delay value determined from the score matrix. The chosen values of standard deviations are crudely representative of pessimistic and optimistic scenarios for process and mismatch variations in the simulated 180 nm process. An interesting detail is that, though variations for each delay element are symmetric around mean value, the total score is almost always lower compared to the exact value. This is an artifact of OR-type Race Logic which favors fast signals over the slow ones.

The detailed analysis of the results of Fig. 6 suggests practical and efficient solution for tolerating variations and noise in asynchronous Race Logic. Indeed, it is known that the probability of small similarity regions in DNA strings is fairly high and goes down exponentially as the length of the similarity goes up [14]. Therefore, the similarity threshold can be defined below which the strings would be assumed similar by chance and not due to genuine alignment. Practically, that means that for OR-type Race Logic we could define a certain threshold beyond which the architecture will not look for similarity and move on to the next string. As Fig. 6c shows there are only few strings, e.g. with scores below 100, which should be considered for alignment, while the vast majority of strings would be discarded. Interestingly, this means that with an increasing dynamic range,

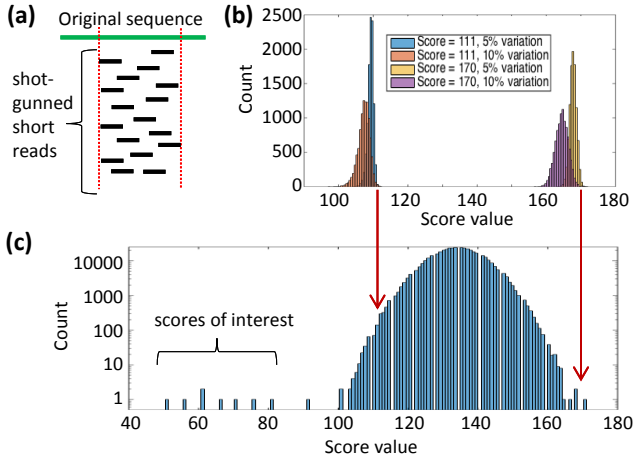


Figure 6: Preliminary variation study: (a) A cartoon of shotgun sequencing process, (b) Monte Carlo simulations (10,000 runs) of the alignment scores for two particular pairs of DNA strings for the OR-type Race Logic with variation-prone delay elements, and (c) representative score statistics in human genome shotgun sequencing. The reference DNA strings is the same in both panels (b) and (c). The query strings used in panel (b) were chosen such that their alignment scores with the reference DNA string correspond to the start and very end of the hump in the score distribution, which is highlighted with red arrows. (The simulation results show that the distributions on both panels are rather insensitive to the particular choice of DNA strings.)

the best-case (rather than worst case) paths become more representative of a typical situation.

When looking for alignments with small scores in such score distribution, variations and noise can be resolved efficiently by slightly increasing the threshold. For example, for the 10%-variation case all strings of interest with a score value below 90 can be detected by setting the threshold to ~ 92 and the penalty of such adjustment would be identifying few false-negative strings whose score is in between 90 and 100. From Fig. 6, it is clear that the increase in runtime due to threshold adjustment and additional work of finding false negative would present just a minor overhead considering that most of the time will be spend screening the strings with high (> 100) scores.

5. CONCLUSION

Our objective was to investigate asynchronous Race Logic and in particular address some of the issues pertinent to synchronous implementation such as third order energy scaling and difficulty with handling of a more complex score matrix. To accomplish this goal we proposed an architecture in which delay is not controlled by a universal clock but rather by a current controlled delay element. The designed delay element was based on current starved inverter topology that was controlled by an external resistor based variable current source. This design allows for a larger dynamic range at lower area and energy costs. As a specific example, we implemented the well-studied shotgun DNA sequence alignment problem and compared synchronous versus asynchronous design styles. The simulation results for the synthesized design in 0.18 μm technology show that, asynchronous implementation is at best 10× more energy efficient, 4× denser and has slightly smaller delays as compared to synchronous one, which by itself significantly outperformed conventional highly-optimized systolic array implementation for sequence alignment problem. Moreover, we show that for at least the considered application, the process and mismatch variations have negligible impact on the asynchronous Race Logic performance and functionality.

Acknowledgment

Removed for blind review.

6. REFERENCES

[1] H. Esmaeilzadeh, E. Blem, *et al.*, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 365–376, IEEE, 2011.

[2] E. E. Swartzlander and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Transactions on Computers*, vol. 24, no. 12, pp. 1238–1242, 1975.

[3] J. E. Volder, "The cordic trigonometric computing technique," *Electronic Computers, IRE Transactions on*, no. 3, pp. 330–334, 1959.

[4] A. Adamatzky and B. de Lacy Costello, "On some limitations of reaction–diffusion chemical computers in relation to voronoi diagram and its inversion," *Physics Letters A*, vol. 309, no. 5, pp. 397–406, 2003.

[5] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: a hardware acceleration for dynamic programming algorithms," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 517–528, IEEE, 2014.

[6] R. Ekblom and J. B. Wolf, "A field guide to whole-genome sequencing, assembly and annotation," *Evolutionary applications*, vol. 7, no. 9, pp. 1026–1042, 2014.

[7] R. Zhang and M. Kaneko, "A feasibility study on robust programmable delay element design based on neuron-mos mechanism," *Proceedings of the 24th edition of the great lakes symposium*, pp. 21–26, 2014.

[8] P. Andreani, F. Bigongiari, *et al.*, "A digitally controlled shunt capacitor cmos delay line," *Analog Integrated Circuits and Signal Processing*, vol. 18, pp. 89–96, 1999.

[9] N. Mahapatra, A. Tareen, and S. Garimella, "Comparison and analysis of delay elements," *Midwest Symposium on Circuits and Systems*, pp. 473–476, 2002.

[10] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: design and analysis," *IEEE Transactions on Very Large Scale Integration*, vol. 11, pp. 871–878, 2003.

[11] M. Saint-Laurent and M. Swaminathan, "A digitally adjustable resistor for path delay characterization in high-frequency microprocessors," in *Mixed-Signal Design, 2001. SSMSD. 2001 Southwest Symposium on*, pp. 61–64, IEEE, 2001.

[12] P. Mroszczyk and P. Dudek, "Tunable cmos delay gate with reduced impact of fabrication mismatch on timing parameters," in *New Circuits and Systems Conference (NEWCAS), 2013 IEEE 11th International*, pp. 1–4, IEEE, 2013.

[13] "Ncbi blast webpage." http://www.ncbi.nlm.nih.gov/blast/.

[14] R. Mott, "Alignment: statistical significance," *eLS*, 2005.