Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms

Advait Madhavan, Timothy Sherwood, Dmitri Strukov

University of California, Santa Barbara

advait@ece.ucsb.edu, sherwood@cs.ucsb.edu, strukov@ece.ucsb.edu

ABSTRACT

We propose a novel computing approach, dubbed "Race Logic", in which information, instead of being represented as logic levels, as is done in conventional logic, is represented as a timing delay. Under this new information representation, computations can be performed by observing the relative propagation times of signals injected into the circuit (i.e. the outcome of races). Race Logic is especially suited for solving problems related to the traversal of directed acyclic graphs commonly used in dynamic programming algorithms. The main advantage of this novel approach is that information processing (min-max and addition operations) can be very efficiently expressed through the manipulation of the natural delay chaining inherent to digital designs, which then results in superior latency, throughput, and energy efficiency. To verify this hypothesis, we designed several Race Logic implementations of a DNA global sequence alignment engine and compared it to the state-of-the-art conventional systolic array implementation. Our synthesized design shows that synchronous Race Logic is up to 4× faster when both approaches are mapped to a 0.5µm CMOS standard cell technology. At the same time the throughput for sequence matching per circuit area is about 3× higher at 5× lower power density for 20-long-symbol DNA sequences.

Categories and Subject Descriptors
B.2 [**ARITHMETIC AND LOGIC STRUCTURES**]: Design Styles – *Parallel.* B.6 [**LOGIC DESIGN**]: Design Styles–*Parallel circuits.* B.7 [**INTEGRATED CIRCUITS**]: Types and Design Styles–*Algorithms implemented in a hardware.*

General Terms
Algorithms, Performance, Design.

Keywords
Race Logic; Dynamic Programming; String Comparison; Shortest-Path Problem; Energy Efficient Circuits; Directed Acyclic Graph.

ISCA '14 Minnesota, USA
978-1-4799-4394-4/14/$31.00 © 2014 IEEE.

1. INTRODUCTION

As we look to more application specific designs to improve performance and energy efficiency, a broad array of options is available. At one end of the spectrum, application customization can rely solely on traditional encoding techniques, as is the case with existing FPGA supercomputers designed to accelerate problems in bioinformatics [1]. Fully customized circuits are also very common – e.g. to speed up audio and image processing [2]. At the other end of the spectrum, one can find somewhat exotic fully customized systems exploiting novel physics and based on nontraditional technologies such as the D-Wave computer, which utilizes quantum annealing phenomena to solve optimization problems [3]. Even for general purpose computing the recent trend is towards the integration of application-specific hardware accelerators [4], which is a way to cope with underutilization of integration capacity (the so-called dark silicon [5]) due to limited power consumption budget. As a result, future microprocessors are likely have many integrated hardware accelerators, which are only efficient in performing specific tasks.

In this paper we propose a novel computing approach, called "Race Logic", which utilizes a new data representation to accelerate a broad class of optimization problems, such as those solved by dynamic programming algorithms. The core idea of Race Logic is to use race conditions set up in a circuit to perform useful computation. While several implementations of Race Logic are possible (including both synchronous and asynchronous), in this paper we focus specifically on synchronous Race Logic, which can be implemented with conventional complementary metal–oxide–semiconductor (CMOS) technology. To make the evaluation of this idea more concrete, the performance of Race Logic is examined using the example of a well-studied DNA global sequence alignment task, which we later extend to consider another common problem in bioinformatics – protein string comparison. To summarize our contributions:

- We present, for the first time, a new method of performing a restricted but useful set of computations that make positive use of race conditions.

- We show that our synchronous Race Logic implementation is both feasible and practical through a synthesizable ASIC implementation for an important instance of dynamic programming optimization.
- We demonstrate that our specific implementation is more efficient (as measured in latency, throughput/area, and energy) than best known traditional designs by factors of 4, 3, and 200, respectively.
- We describe the design space of Race Logic for this class of applications more broadly, and propose and model several important optimizations that make this new class of designs even more useful.

The rest of the paper is organized as follows. The next section presents related work as well as the background required to better understand the sequence alignment problem. Section 3 describes the main idea behind Race Logic. Section 4 presents the implementation details and simulated results for the DNA global sequence alignment problem, while in Section 5 a more general architecture is outlined. Finally, the results are discussed and concluded in the last two sections.

2. BACKGROUND AND RELATED WORK

While the end of traditional CMOS scaling has re-energized the search for novel models of computation, there is a long history of application-targeted forms of computation seeking to avoid the overheads associated with traditional binary-encoded arithmetic operations. Even if we avoid a discussion of the many non-electrical computing devices developed from antiquity up into the modern era, there is still an impressive array of models to consider. However, most related work can be thought of as falling into one of three camps: non-traditional arithmetic encodings on traditional devices, methods for exploiting non-traditional computational substrates, and sequence alignment architectures.

2.1 Non-Traditional Digital Encodings

Information representation can play a major role in improving the energy efficiency or speedup of specific information processing algorithms. For example, performing multiplication and division operations can be complex and require special hardware for binary encoding schemes, but using logarithmic number systems [6] makes performing such operations as straightforward as addition and subtraction.

CORDIC algorithms extend this idea to perform vector rotations in a bit-serial manner by using clever encodings that allow computation for arbitrary angles by using simple iterative shift and add procedures. Many algorithms designed along these lines were conceived at a time when chip area was extremely expensive and performing parallel computation operations such as square root would be prohibitively expensive in area, but it is perhaps worth revisiting these ideas in the new context of power efficiency. However, while CORDIC and other bit-serial arithmetic provide density unlike our Race Logic encoding, those bits that are used tend to have a very high activity factors.

While not a numeric encoding scheme as such, arbiter based PUFs (Physically Unclonable Functions used in hardware security) are one of the few cases other than our work where race conditions have been exploited to a positive purpose [7]. These PUFs provide a method to uniquely encode a large pseudo-random function based on the physical unique characteristics of a chip. The function is evaluated by racing two paths, with variation-induced timing delay between them, against one another, to determine the winner. Unlike PUFs however, we use race conditions that are synchronized internally to reduce unknown variations, and more importantly we use those race conditions to perform actual computations rather than chip identification.

2.2 Novel Computing Substrates

With traditional computing structures struggling to improve performance per watt, there has been a renewed interest in non-traditional computing materials. One school of thought centers on replacing existing silicon based transistor technology by creating logically complete and energy efficient structures out of novel materials such as Carbon Nanotubes Transistors (CNTs) [8]. A second class of approaches seeks to augment traditional silicon with novel devices. For example, resistive switching devices can be coupled with silicon through a complementary and logically complete crossbar structure. This structure tightly integrates logic and memory making it a natural fit for memory intensive tasks such as pattern matching [9,10,11,12]. A third strategy is to forgo logical completeness (for the sake of general purpose computation), concentrating instead on performing those limited computations that are naturally governed by the dynamics of the material system involved. For example, reaction-diffusion systems made up of 2D chemical substrates can be used to solve 2D Voronoi Diagram Problems [13]. Chris Dwyer et al. use a hybrid of the above techniques to solve the block edit problem [14]. By utilizing the self-assembling properties of DNA to probabilistically cover all possible block orderings at fabrication, the effective runtime complexity of the algorithm can be reduced from $O(N!^2)$ to $O(N^2)$.

While our approach in Race Logic does not currently use any non-standard silicon technologies, we embrace the philosophy of this third strategy in that our information representation limits the computation to min, max, and comparison, which are still powerful enough to be used to find the optimal path amongst all possible cases.

2.3 Sequence Alignment Architectures

While the point of our work is to explore the computational potential of races, we use the well-studied problem domain of sequence alignment to test the potential of this new logic. In this section we will touch upon the extensive body of prior work on sequence alignment, talk about an important systolic array realizations and discuss other implementations ranging from supercomputing platforms to FPGA related applications.

A common problem in bioinformatics is to estimate the similarity between DNA or protein sequences. These sequences can have different alphabet sizes varying from 4 in the case of DNA (A, G, C, T representing the neucleo-bases) to 20 for a closely related protein comparison problem, in which strings consists of letters representing a particular amino acids [13].

A typical string similarity metric originating in information theory is the Levenshtein distance, also known as "edit" distance. It can be intuitively understood as the number of "edit" operations namely: Insertions, deletions, and substitutions, which are required to convert one string to another. To understand these edit operations let us consider string P = "ACTGAGA" of length N = 7 and string Q = "GATTCGA" of length M = 7. Figures 1a and 1c show two methods of converting string P to Q. Here, columns with top row spaces represent insertions while bottom row spaces are deletions, and when lumped are known as "indels". Columns with the same characters in both rows are known as "matches" and different ones are known as "mismatches". In particular, the first method (Fig. 1a) involves deleting letters C, G and A and inserting G, T and C, while the second method (Fig. 1c) deletes string P completely and inserts string Q. An important point to note is that even though the alignment shown in Figure 1c has no matches and is the worst case, it is still an allowed alignment. Also, the number of matches plus the number of mismatches plus the number of indels can be equal to the sum of the length of the two strings, i.e. N+M in our case, as is shown in Figure 1c, but can never exceed it.

Figures 1b and d are alternate representations for two considered alignments, where the number in any position denotes the number of symbols present in Figures 1a and 1c up to that particular position. Note that there is only an increase in numerical value at a particular position when it houses a symbol and not a space. This representation is known as the alignment matrix as each column can be thought of as a coordinate in a two dimensional $N \times M$ grid which composes the edit graph (Fig. 1e). The edit graph is a directed acyclic graph (DAG) that is a two-dimensional representation of all the possible alignments between the two strings. Any particular alignment is just a path in this graph where every edge corresponds to an edit operation. The arrows show all the possible alignments; the vertical arrows representing insertions, horizontal arrows representing deletions and diagonal arrows representing matches. For example, dark blue and dark red arrows on Figure 1e correspond to the two specific alignments shown on Figure 1a and 1c, respectively.

(a)

$\mathcal{P}$	_	A	C	T	G	_	A	_	G	A
Q	G	A	_	T	_	T	_	C	G	A

(b)

$\mathcal{P}$	0	1	2	3	4	4	5	5	6	7
Q	1	2	2	3	3	4	4	5	6	7

(c)

$\mathcal{P}$	A	C	T	G	A	G	A	_	_	_	_	_	_	_
Q	_	_	_	_	_	_	_	G	A	T	T	C	G	A

(d)

$\mathcal{P}$	1	2	3	4	5	6	7	7	7	7	7	7	7	7
Q	0	0	0	0	0	0	0	1	2	3	4	5	6	7

(e)

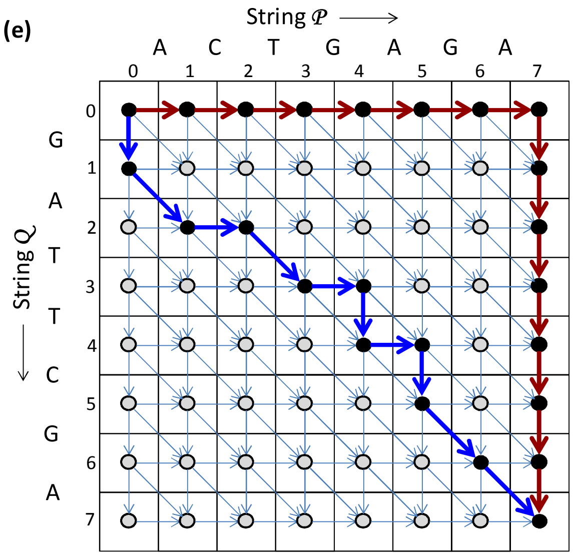


Figure 1. (a, c) Two possible alignments between strings $\mathcal{P}$ and Q and (b, d) their corresponding alignment matrixes and (e) edit graph.

Analyzing the merit of any particular path in the graph is equivalent to analyzing the merit its corresponding alignment. Given any two strings there are a large number of different paths and alignment matrices, each with its own arrangement of matches and indels. To determine the relative merit of one particular alignment over another the concept of a score matrix 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 longest path in the graph in the case when matches are assigned the highest values in score matrix (Fig. 2a), or, alternatively, the shortest path in the opposite case (Fig. 2b). Note that in general the penalty for the mismatch may also depend on a particular pair of letters, which is the case for matrix shown on Figure 2c.

Example max and min score functions are

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(-, P_j) \\ s_{i,j-1} + \delta(Q_i, -) \\ s_{i-1,j-1} + \delta(Q_i, P_j) \end{cases}, \qquad (1a)$$

$$s_{i,j} = \min \begin{cases} s_{i-1,j} + \delta(-, P_j) \\ s_{i,j-1} + \delta(Q_i, -) \\ s_{i-1,j-1} + \delta(Q_i, P_j) \end{cases}, \qquad (1b)$$

respectively, where *i* and *j* are row and column indices (Fig. 1e). Applying equation 1a and score matrix from Figure 2a converts the alignment problem to a longest path problem by rewarding matches with an increase in the score by 1, while using Equation 1b and score matrix from Figure 2b penalizes indels by 1 and mismatches by 2 and is equivalent to a shortest path problem. It is also worth mentioning that finding longest and shortest path with score matrixes on Figure 2a and 2b are equivalent problems. The shortest path formulation is more suitable for the considered implementations, and will be used in the synthesized design.

(a)

	A	C	T	G	_
A	1	0	0	0	0
C	0	1	0	0	0
T	0	0	1	0	0
G	0	0	0	1	0
_	0	0	0	0	0

(b)

	A	C	T	G	_
A	1	2	2	2	1
C	2	1	2	2	1
T	2	2	1	2	1
G	2	2	2	1	1
_	1	1	1	1	1

(c)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	J	Z	X	_
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	-1	-1	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	-2	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	4	-3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	-3	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-1	-3	-1	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	-2	4	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	-3	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-4	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	-3	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	3	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	-3	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	2	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	0	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-3	-1	-1	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	-2	0	-1	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	-1	-1	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-2	-2	-1	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-1	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	2	-2	-1	-4
B	-2	-1	4	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	-3	0	-1	-4
J	-1	-2	-3	-3	-1	-2	-3	-4	-3	3	3	-3	2	0	-3	-2	-1	-2	-1	2	-3	3	-3	-1	-4
Z	-1	0	0	1	-3	4	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-2	-2	-2	0	-3	4	-1	-4
X	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-4
_	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Figure 2. Examples of score matrixes: (a) Longest and (b) shortest path score matrixes for DNA local sequence alignment problem [15, 16] and (c) BLOSUM62 (longest path) score matrix for protein sequence alignment problem [17].

Not only is the edit graph representation a handy tool for visualizing paths and their corresponding alignments, it is also closely tied to the concept of dynamic programming. In particular, dynamic programming relies on solving progressively larger sub-problems starting with a set of small problems and using the results of previous calculations for each new step. Each node on the edit graph calculates the score corresponding to the optimal solution of the sub-problem i.e. either shortest or longest path (depending upon the score matrix) from the root node to itself. Adjacent nodes utilize these optimal solutions to calculate their own score as the computation "wave" proceeds along the diagonal. The edit graph itself consists of all possible alignments represented as paths from the root node to the end node and hence the above method guarantees searching of the entire space for the most optimal alignment between the given strings.

The above concepts were brought to light in seminal works by Needleman and Wunsch [18] and Smith and Waterman [19] which in-turn spurred a considerable research interest in software and hardware methods for string comparison. Algorithms for string comparison vary from brute force dynamic programming methods to heuristic solutions that do not search the entire space but provide quick solutions [20]. The major obstacle for hardware implementations of DP sequence alignment had to do with the area complexity as the similarity metric for comparison i.e the score is cumulative and increases with array size. Any ASIC implementation would then need processing elements(PEs) that can store this worst case cumulative score and would hence lead to large, string length dependent sizes.

Lipton and Lopresti proposed a systolic array solution, that using maximum score dependent modular arithmetic, limits the number of bits of data that needs to be stored as well as shared between processing elements [16]. Hence, they were able to make sure the area scaling issues are mitigated, at the cost of extra circuitry outside of the systolic structure to recalculate the original score. To reduce the interconnect overhead, the systolic architecture utilized an tight encoding scheme that interleaves the alphabet and scores. Lipton and Lopresti not only addressed the area issue but theirs was the first paper to talk about anti-diagonal independence of elements in the edit graph and utilized this property for fine grain parallelism. The resultant hardware was a linear systolic array whose processing elements could differentiate between the alphabet and scores as perform comparison and addition operations. Newer architectures [21, 22] have built upon this Lipton and Lopresti work by adding markers in processing elements to trace back optimal similarity paths.

Other platforms on which sequence alignment has been performed, range from networked DEC Alpha workstations [23], to GPUs [24], to Supercomputers. SIMD supercomputing platforms, such as MasParMP2 [23], that have multiple low complexity processing elements that perform fine grain computation on multiple data streams parallely, achieving high performance at reasonable cost. On the other hand, general purpose MIMD supercomputer, Paragon [23] seemed to have performance on par with the networked DEC Alpha workstations [23]. Other implementations include FPGA based reconfigurable platforms such as Jbits [25] that utilize re-configurability to create a custom architecture using the entire string as a parameter or by defining custom instructions on FPGAs that can handle multiple input and flags at the same time to speedup computation [26]. Custom ASIC implementations such as BioSCAN utilize heuristics and a very high density implementation that results in high performance [27].

In contrast to these related works, we propose a new method of information representation that performs computation by setting up logical race conditions in a circuit. Though these race conditions are not race conditions in the circuit sense, they are race conditions in the software sense, as the time taken for computation across different logical path directly affects the end result. As we detail in the next section, due to this novel information representation, some operations such as addition and comparison become energy efficient due to the properties of delay itself.

3. RACE LOGIC

As mentioned previously, the general idea of Race Logic is to encode information in the timing delay. In the context of the considered operations on DAGs, 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. This is implemented by converting edge weights of a graph to the corresponding timing delays and replacing nodes with either AND or OR gates for max and min score functions, respectively.

To explain how score functions (Eq. 1) are implemented with Race Logic, consider the job of one node in the edit graph. It needs to choose the min of multiple different inputs, where each of those inputs is penalized by a constant value. If values are represented by a delay from a reference point *t* (the start of the computation), we can add a constant *c* to a value by simply delaying it by *c* time steps. More concretely a score of n, is represented by a Boolean signal "1" appearing at the output of the node n unit delays after *t*. Furthermore, when a signal is encoded in time, the *min* operation on a node in the graph receiving multiple inputs is equivalent to passing along the first arriving "1", which can be implemented with a simple OR gate. Similarly, as the AND gate passes the last arriving "1", the AND gate performs the max operation". The shortest/longest path DAG problem is therefore solved by measuring the time to propagate the signal from the root node(s) to the output node(s) for a graph, in which all nodes are replaced with OR/AND gates while edge with corresponding delays.

Figure 3a shows an example of a particular DAG with two input nodes and one output node converted to AND- (Fig. 3b) and OR-type (Fig. 3c) Race Logic circuits. For synchronous Race Logic, the unit delay is assumed to be equal to one clock cycle so that D Flip Flops (DFF) gates implements delay elements. In particular, DFFs can be shift-chained for the cases where the edge weight is a small number or, alternatively, an encoded configuration can be used to implement larger weights. Note that for practical reasons very large weights (or more the specifically max weight ratio) should not be too large, unless the weight is truly infinite (which can be implemented as a missing edge). The edit graph can be now thought of as a very deep pipeline, with competing paths to the final node from the root node, with all the flip flops initialized to "0".

To initiate a race computation, both for the OR and AND types Race Logic, the input nodes are given a steady value of "1". With every new clock cycle, the "1" signal propagates down the edges of the graph until it reaches another node, where it gets delayed until the other inputs of the node are also "1" in the case of AND-type Race Logic, or until it just propagates through to the next edge in the case of OR-type Race Logic. For the specific DAG shown in Figure 3a, it takes two cycles for the "1" signal to propagate to the output node and it can be easily verified that this corresponds to the shortest path. Note that the shortest/longest path value can be converted back to the common representation with a simple counter.

For a fixed-weight graph, like the one shown in Figure 3a, making a custom circuit to find the shortest or longest path may not be practical if the Race Logic computation need be performed only once (or even several times when computing paths between different input-output nodes). Even for an FPGA implementation of Race Logic, the configuration overhead is likely to overwhelm the running (useful) time of solving the problem. A more practical situation is to have a DAG in which weights of some (or all) edges are controlled by external conditions. Fortunately, this is true for the majority of sequence alignment problems, because in edit graphs the weights depend on the particular strings being compared. This allows for efficient reuse of the same Race Logic hardware, because the solution for the alignment problem will depend on the external conditions and must be recalculated for new pair of strings.

For example, Figure 4 shows an OR-type synchronous Race Logic implementation of the Smith-Waterman algorithm for DNA local sequence alignment with score matrix from Figure 2b. Here, the signal

$$M_{i,j} = \begin{cases} 1, & Q_i = P_j \\ 0, & Q_i \neq P_j \end{cases}, \quad (2)$$

is a matching condition between the corresponding pair of letters and is assumed to be implemented with XNOR gate. As it is obvious from the figure the structure is very uniform and is obtained by replicating unit cells hosting OR, DFF and AND gates. In order to simplify the circuitry, the scoring matrix is slightly modified by replacing weights for mismatches from 2 to infinity. It is straightforward to check that the original and modified scoring matrixes are equivalent and thus result in the same values of score for the nodes of the edit graph. Figure 4c demonstrates how the signal injected at the input node propagates through the edit graph for particular strings of DNA, which are similar to the previously considered example (Fig. 1). It is easy to check that propagation delay corresponds to the best alignment score between these two strings.

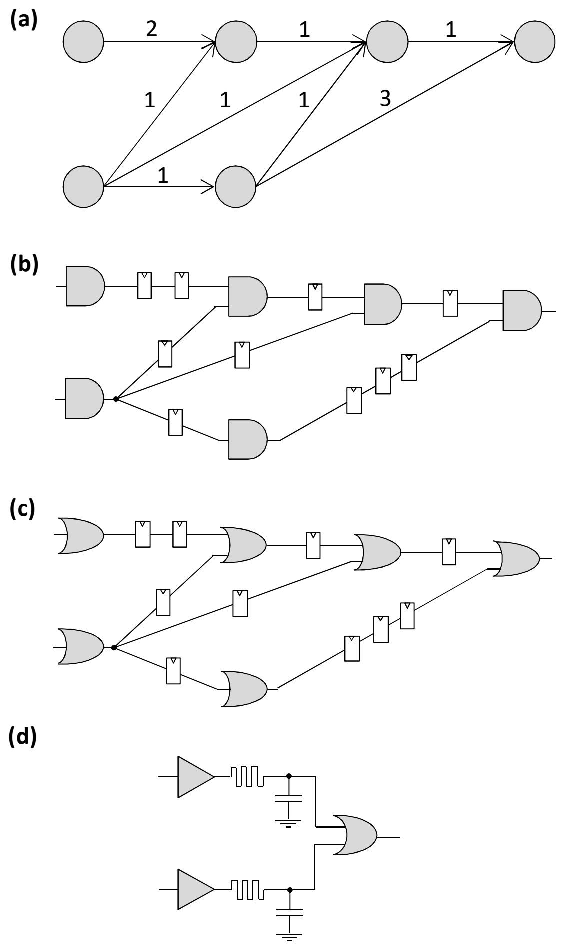


Figure 3. (a) Example of a DAG with weighted edges, and its corresponding synchronous Race Logic implementation using (b) AND and (c) OR types for longest and shortest paths computation, respectively. (d) Example of asynchronous Race Logic implemented with resistive switching devices.

4. CASE STUDY

4.1 Design Methodology

To make sure that the comparison is a fair one we implement the Lipton and Lopresti architecture using a recent standard cell technology and include all of the area optimizations as well as encoding schemes that were implemented in the original architecture. The process used is an AMIS 0.5µm process and the studies are done using both OSU standard cells as well as AMIS standard cells to study the tradeoffs involved in using a different standard cell set. For the implementation of both architectures, a parameterized and scalable Verilog code is synthesized using Synopsys Design Vision tool to get estimates of area. Power and timing information is obtained using Synopsys Primetime tool using a representative set of input vectors. Since the power consumed by the architecture is highly dependent on the input vectors, random input vectors cannot be used. A specific set of input vectors that follow the correct encoding is generated using a test-bench. These simulations are performed using the Modelsim tool, which generates toggle information of each net on the synthesized netlist. This toggle information is then used by Primetime tool with a 100% coverage (confidence metric) to estimate power values. Figure 5 shows how the latency, area, energy per computation, and throughput scales with different string length *N* for using the score matrix shown in Figure 2b.

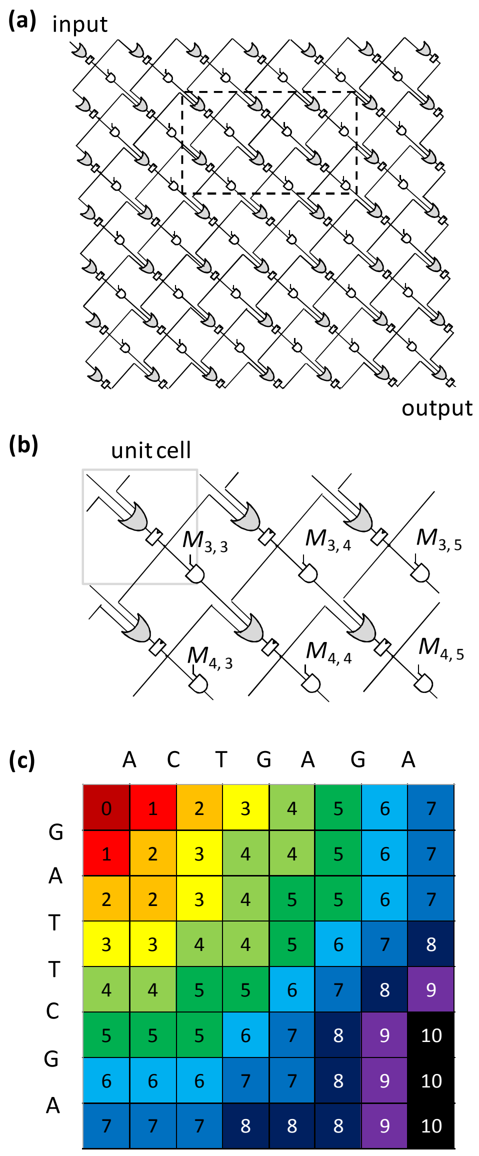
4.2 Analytical Estimates

Among the simulated performance metrics, area and latency scaling with string length are the easiest to understand. The area of the Race Logic scales quadratically with *N*. The latency scales linearly with *N*. In the worst case scenario, i.e. when the strings are completely mismatched, it takes 2*N*-2 cycles for a considered score matrix and only *N*-1 cycles in best case scenario, i.e. when the strings are completely aligned.

Derivation of energy and power requires more in-depth analysis. Let's assume that C_{clk} corresponds to the capacitances of DFFs that are clocked every cycle, and hence having an activity factor of 1, while the $C_{\mathrm{non\text{-}clk}}$ corresponds to all other capacitances that have data dependent activity factors. For both the best and the worst case scenarios all the non-clocked capacitances in the entire architecture are charged once per comparison. This can be seen very easily in the worst case, by following the horizontal and vertical edges on the edit graph, but is similar in the best case as the propagating "1" uses the diagonal delay elements to propagate to the extreme topological east and south blocks of the architecture. From the simulated power estimation results, the total power consumption is dominated by dynamic one. Hence the power dissipated can be written as

$$P = C_{clk} {V_{dd}}^2 N^2 f + C_{non-clk} {V_{dd}}^2 N^2 \alpha f \ , \qquad (3)$$

where α is the activity factor that is data dependent, V_{dd} is voltage supply, and f is a frequency of operation.



0	1	2	3	4	5	6	7
1	2	3	4	4	5	6	7
2	2	3	4	5	5	6	7
3	3	4	4	5	6	7	8
4	4	5	5	6	7	8	9
5	5	5	6	7	8	9	10
6	6	6	7	7	8	9	10
7	7	7	8	8	8	9	10

Figure 4. An example of OR-type synchronous Race Logic implementing global sequence alignment of DNA strings: (a) The circuit for $N = M = 7$ and (b) zoom-in a particular part, and (c) corresponding example of an signal propagation via Race Logic for the particular choice of DNA strings (Fig. 1). The number inside each cell represents timing, i.e. clock cycle at which signal "1" reached the output of an OR gate of a particular unit cell.

Energy consumed per comparison can be calculated by multiplying power by the time taken per operation. Therefore energy dissipated per comparison for the best case and worst cases are

$$E_{best} = C_{clk}{V_{dd}}^2N^3 + (C_{non-clk} - C_{clk}){V_{dd}}^2N^2 \quad (4a)$$

$$E_{worst} = 2C_{clk}{V_{dd}}^2N^3 + (C_{non-clk} - 2C_{clk}){V_{dd}}^2N^2 \quad (4b)$$

correspondingly. The Equations 3 and 4 define the scaling law of energy and power with respect to N. Since C_{clk} and $C_{\text{non-clk}}$ are not known parameters we estimate them from fitting. The resulting equations from fitting for both the AMIS and OSU standard cell libraries are

$$E_{AMIS,best} = 2.65N^3 + 6.41N^2 \quad (5a)$$

$$E_{AMIS,worst} = 5.30N^3 + 3.76N^2 \quad (5b)$$

$$E_{OSU,best} = 1.05N^3 + 5.91N^2 \quad (5c)$$

$$E_{OSU,worst} = 2.10N^3 + 4.86N^2 \quad (5d)$$

where the units of energy are in pJ.

4.3 Energy-Optimized Architecture

One of the drawbacks of Race Logic is its third order energy scaling with string length N. By observing Equations 4a and 4b we can see that the capacitance associated with the clocked region of the fabric constitutes the cubic behavior. This is due to the fact that the area scales quadratically and the time taken per computation scales linearly, but most importantly this area is clocked every cycle. Fortunately, using a strategy known as a clock gating, this term can be greatly reduced.

By exemplifying the worst case; i.e. the case of maximum energy per computation, from our given score matrix, we can see that there is a time dependent "wavefront" of the propagating Boolean "1" as is shown in Figure 1. This wavefront represents the cells where the flip flops are changing state from Boolean "0" to "1". The cells that are away from the wavefront, i.e. the ones which have already changed state to "1" (gray cells in Figure 6) as well as the ones that are still "0" at this particular clock cycle (white cells in Figure 6a) are going to retain their state for the next clock cycle and hence do not need to be clocked. By employing a data dependent clock gating strategy we can turn off regions of the chip that are not being utilized to save power. Due to the regular structure of the Race Logic fabric the clock network can be designed as an H-tree. One of the major parameters that would determine the power savings would be the granularity of the H-tree, in other words, the number of cells that would be gated at once. Let us look at a 4x4 group of cells (multi-cell region) as shown enclosed in Red in Figure 7a. This multi-cell region can be thought of as the smallest group of cells that can be gated at once. During the operation of the circuit, if the cells that are grey in color have the Boolean value "1" then it means that the wavefront has crossed this multi- cell region and their values are not going to change in this operation. Also if the cells that are in black have the Boolean value "0", it means that the wavefront has not yet approached this multi-cell region.

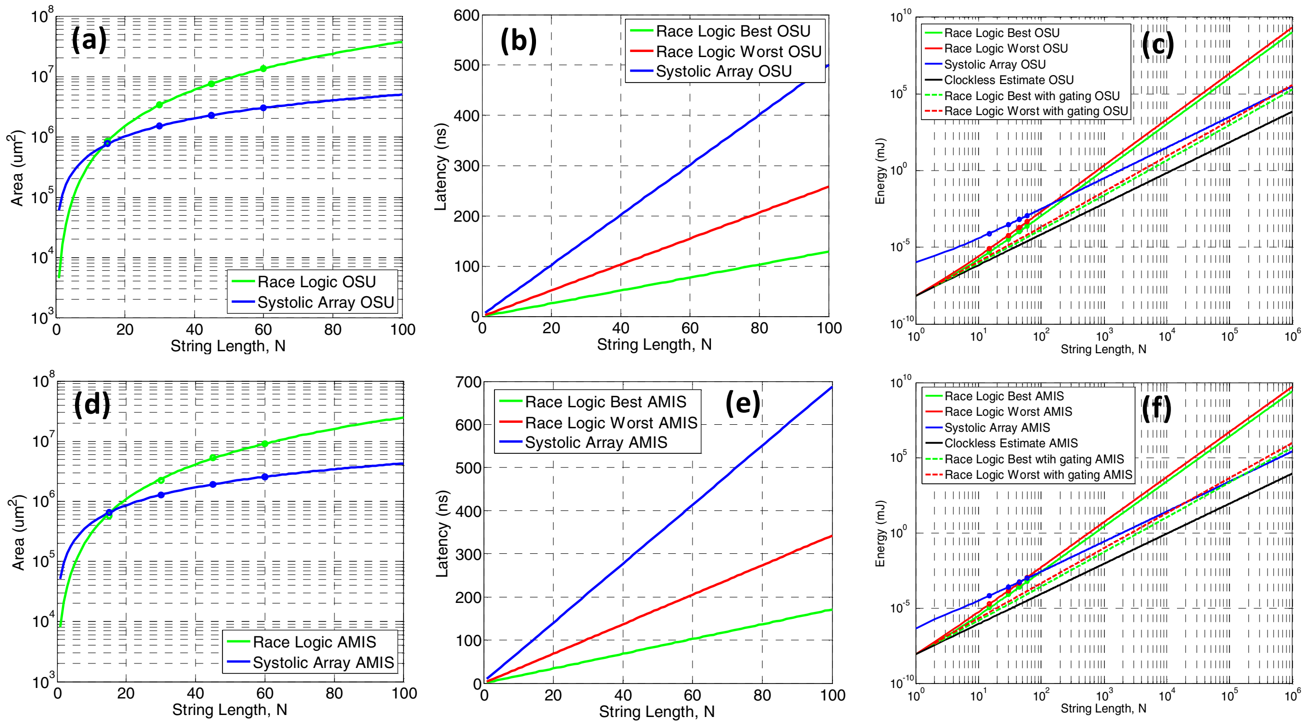


Figure. 5. (a, d) Area, (b, e) latency, and (c, f) energy per string comparison operation as a function of string length N for Race Logic and Lipton and Lopresti systolic array for two different CMOS standard cell library implementations. The points on panels a, d, c, and f come from simulation results while solid lines are analytical (fitted in case of energy figures) curves.

For both the above cases, the multi-cell region shown in Figure 7a doesn't need to be clocked. By activating the clock of the multi-cell region on the arrival of the Boolean "1" on the black cells and deactivating it when all grey cells are "1" we can ensure that this multi-cell region is clocked only for a limited period of time, hence reducing energy consumption. Very fine granularity of this multi-cell region would increase energy dissipation due to a large number of multi-cell regions that require every cycle clocking, while very coarse granularity would mean clocking one multi-cell region for very long, also increasing energy dissipation.

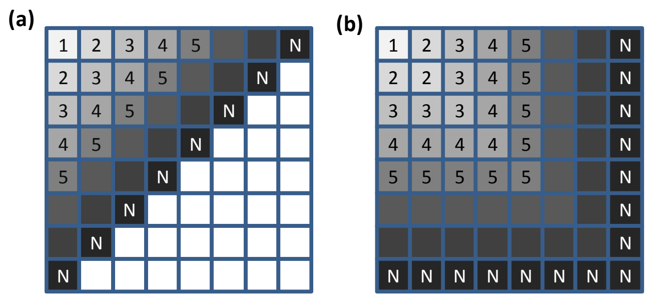


Figure 6. Propagation of wavefront for (a) worst case and (b) best case alignment for the score matrix shown in Figure 2b. Each shade of gray shows the wavefront at a different clock cycle as numbered.

To calculate the optimal granularity, we introduce a parameter m, which is the side length of one multi-cell region as shown in figure 7a. Now the worst case energy dissipation for the clocked part of the architecture is as follows,

$$E_w = C_{clk} {V_{dd}}^2 N^2 (2m-2) + C_{gate} {V_{dd}}^2 \frac{N^2}{m^2} (2N-2) \qquad (6)$$

where the first term represents the entire clocked capacitance being activated only for $2m$-2 cycles (i.e., the worst case number of clock cycles one multi-cell region remains active) and the second term represents the gating capacitance that the clock distribution network still has to clock, with C_{gate} is the actual capacitance, $(N/m)^2$ is the number of multi-cells regions and $2N$ - 2 factor is the total number of cycles. Solving for minimum energy, we get

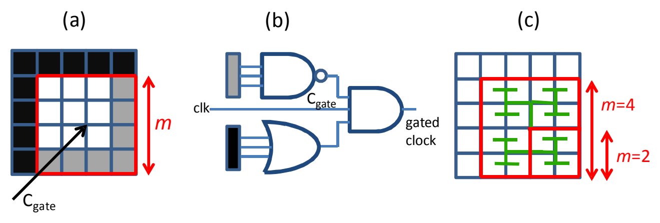


Figure 7. (a) 4×4 multi-cell region with a gated clock, showing associated clock gating capacitance of C_{gate} and (b) its circuit representation. (c) H-tree type clock network showing two cases of granularities of gating m = 2 and m = 4.

$$m = \sqrt[3]{\frac{2C_{gate}(N-1)}{C_{clk}}} \quad (7)$$

5. GENERALIZED RACE LOGIC ARCHITECTURE

The score matrix which was considered in a previous example is simple and easy to implement. However, score matrices for sequence alignment have evolved considerably from the time Lipton and Lopresti systolic architecture was published. Nowadays, important properties of the score matrix such as symbol size (N_{SS}) and dynamic range (N_{DR}) change from application to application and even the same application can have different dynamic ranges. One such example, the modern amino acid score matrices, which go by industrialized acronyms such as BLOSUM62 and PAM250, are large, complex matrices with above 400 elements and are highly tuned as a result of statistical behavior of amino acid sequences [26]. Therefore, it is worth investigating generalized Race Logic which can deal with different types of score matrices.

The first step is to convert any given matrix to the form which can be used with generalized Race Logic architecture. Using BLOSUM62 as an example (Fig. 2c), we prepare the score matrices for its Race Logic realization, by keeping a few things need to be kept in mind. Firstly, since our preferred type of Race Logic is the OR race, we must ensure that the highest similarity corresponds to the smallest score and hence the lowest latency. The BLOSUM62 score matrix rewards perfect matches with positive scores, substitutions with negative scores and indels are generally of the lowest score. It is possible to invert the score matrix from longest path to shortest path one by understanding the value in score matrix are obtained. In particular, the origin of these score matrices are based on log-odds score calculations as shown below,

$$S(a,b) = \frac{1}{\lambda}\log\frac{P_{ab}}{f_a f_b} \quad (8)$$

where $S(a,b)$ is the score for symbols a, b, P_{ab} is the joint probability of alignment of a and b, f_a and f_b are the probabilities of alphabet a and b by themselves and λ is a scaling factor to get integer values of scores. By inverting the above equation, and changing the scaling factor we can convert all diagonal elements from positive to negative and non-diagonal from negative to positive.

The next step is to obtain the equivalent score matrix with all positive weights since negative or zero weights cannot be implemented in a straightforward way in Race Logic. The solution to that problem is to add a fixed bias to values of score matrix corresponding to the indels and double of that fixed bias to the remaining ones, as the latter are one rank ahead in the edit graph (Fig. 1e).

The score matrix now consists of elements ranging from 1 to N_{DR}, with the scores along the diagonal being the smallest and indels being the largest. As can be seen in Figure 2c, modern score matrices contain a lot of repeating scores. When using one hot encoded DFFs for realization of delay, the area of a single Race Logic cell scales linearly with dynamic range and hence may have serious area repercussions for large values of dynamic range. Binary encoding with a saturating up-counter allows us to save on area by reducing the number of DFFs for the same N_{DR} as well as making sure that the counter doesn't overflow and restart the count. The generalized structure of complex Race Logic cell is shown in Figure 8, the Boolean "1" signal can come in either from the left, diagonal or top of the cell, which then passes through the OR gate and enables the saturating counter that begins to count 0 to N_{DR} clock cycles. The output of the each colored gate, represents a specific weight, which will trigger as soon as the desired weight is reached and the weight that is desired can be selected from the MUX whose inputs are the encoded forms of the alphabet. To ensure that the output signals that are generated are not pulses but fixed Boolean "1"s, the set on arrival circuit is placed which is reset at the end of each computation.

6. DISCUSSION

As it is mentioned earlier, area scaling of the Lipton and Lopresti architecture is linear in N, while Race Logic has quadratic area behavior. In spite of such unfavorable area scaling laws, the constants associated with Race Logic are smaller than that of the systolic architecture due to the simplicity of the fundamental cells (Figs. 5a,d), which arises

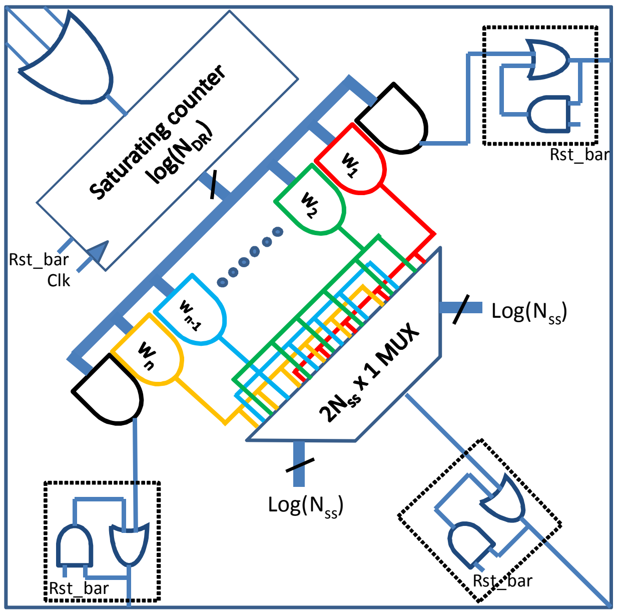


Figure 8. Generalized architecture for single synchronous Race Logic cell. The region within in the dotted black box is the set on arrival circuit.

due to the choice of data representation. Note that Lipton and Lopresti architecture requires a linear systolic array of $2N + 1$ element for a string of length N.

One important observation is that worst case scenario for comparison, i.e. complete mismatch of strings, is not representative of the typical needs of applications. More specifically, the typical requirement is to determine if string similarity is above a certain threshold. For example, due to the large volumes of DNA data availability, there is a need to know if sequences are genuinely aligned (share a common ancestral sequence or function) or are aligned by chance [28]. Statistically, it is known that the probability of small similarity regions in strings is fairly high and goes down exponentially as the length of the similarity goes up. Therefore, in such applications a similarity threshold is defined below which strings are assumed to be similar by chance and not due to genuine alignment.

This means that in our OR-type race implementation, a smaller score can be attributed to a higher level of similarity and a threshold score can be decided, beyond which the architecture will not look for similarity i.e. if the count exceeds the threshold value, the architecture will treat it as if the required match was not found and move on to the next pattern. This feature is very useful as the maximum possible score is known at each instant in time, and not only at the end of the computation. This also means that with increasing dynamic range, the best case scenario becomes more representative of a typical situation and the latency does not necessarily scales with dynamic range N_{DR}. For Lipton and Lopresti architecture, however, the entire computation has to complete, before which the maximum score can be ascertained.

Energy consumption is where the idea of Race Logic really shines through. Due to the fact that computation occurs only along the wavefront, the race logic structure can be gated effectively to save on energy. The systolic array on the other hand is linear and hence needs to be clocked every cycle. Hence, for small value of the string comparisons, Race Logic outperforms the systolic array for both the best and worst case scenarios (Figs. 5c, f). In general, architectures that focus on energy savings do so by reducing the latency (in other words trading off energy for time). Such architectures are not of much use when it comes to high performance architectures [29]. In our case it is the novel data representation that allows this architecture to maintain a fast operating speed as well as be energy efficient. A related result is that energy-delay product (Fig. 9c) and power density (Fig. 9b) are much smaller for Race Logic. The latter is also far away from maximum value of 200 W/cm^2 as defined by International Technology Roadmap for Semiconductors [30].

Even despite unfavorable area scaling law, the throughput per area of best case scenario of Race Logic is considerably

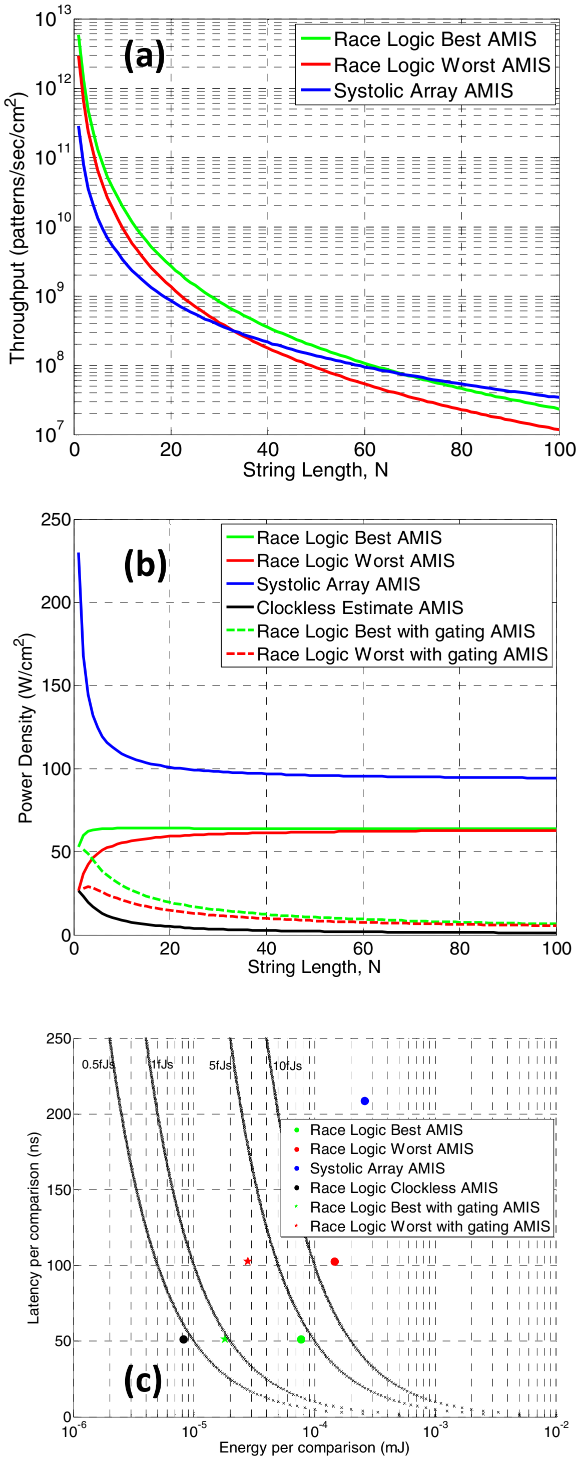


Figure 9. (a) Throughput per unit area and (b) power density as a function of string length N and (c) energy-delay scatter plot for N = 30 for Race Logic and Lipton and Lopresti systolic array for AMIS standard cell library. Black lines on panel c represent constant energy-delay curves.

better than that of the systolic array for $N < 70$ as can be seen in Figure 9a.

It is also worth mentioning that the Race Logic architecture was first implemented on an FPGA due to ease of use and re-configurability but the results were completely unexpected. The latency and energy numbers were very large, and upon further inspection, we realized that due to the large capacitance of the global wiring, the interconnect delay was more than the gate delay. Similarly, the energy numbers were larger than expected due to unnecessary charging the discharging of global capacitances. We concluded that the Race Logic architecture was not suitable for mapping on a general purpose FPGA and this is why we proceeded to perform a standard cell based custom design flow. We believe that a full custom design flow with tweaks to "clocked capacitance" values as well as appropriate sizing of the devices would yield further improvements in results.

Finally, the most optimal implementation of Race Logic is asynchronous and in the analog domain. Most importantly, the asynchronous Race Logic does not have a clock network which is the reason for third order energy scaling with N. This is highlighted by clockless estimates in Figures 5 c, f, and 9 b, c. In fact, it can be observed that optimized gated design as discussed in Section 4.3 is a step in that direction. Moreover, resistive switching devices [31] can be used to implement configurable edge weights (Fig. 3d), which would provide increased advantages in area and energy.

7. SUMMARY

Race Logic is not intended to be any sort of replacement for traditional design practices in general purpose logic. However, often times we consider the world of hardware design cleanly broken up between "digital" systems which encode values as bits on a wire, and "analog" system which encode values as continuum of levels on a wire. Race Logic is an interesting point that in some sense lies between them. All wires in the system are driving either a 0 or a 1, as in traditional digital logic, but the *time* at which those values are driven encodes the data. Certain problems, such as min and max, become trivial to implement, but of course there are many other relationships that are then harder to calculate. However, min and max are powerful operators and sufficient (with the proper routing of values) to solve complex optimization problems, including but not limited to those solved by dynamic programming. The techniques presented in this paper point to a new class of architectures useful when these optimizations need to happen with exceedingly low power or high throughput. To show that this is a fruitful direction we need a comparison with an optimized traditional implementation in a well studied area of work. To this end we designed Race Logic implementations of DNA local sequence alignment problem using global alignment algorithm and compared to that state-of-the-art systolic implementation. The modeling results support dramatic energy advantage of Race logic in energy and substantial improvement in throughput.

8. ACKNOWLEDGMENTS

The authors would like to thank Alex Mishchenko, Robert Brayton, Behrooz Parhami, for valuable discussions and the anonymous reviewers for their insightful comments. This work is partly funded by NSF grants CCF-1017579 and CNS-0910734 and Hellman Family Foundation Fellowship.

9. REFERENCES

[1] B. Schmidt. *Bioinformatics: High Performance Parallel Computer Architectures.* CRC Press, Boka Raton, FL, 2010.

[2] K. K. Parhi. *VLSI Digital Signal Processing Systems: Design and Implementation.* John Wiley & Sons, Inc. New York, NY, 1999.

[3] M.W. Johnson, M.H.S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A.J. Berkley, J. Johansson, P. Bunyk, E.M. Chapple, C. Enderud, J.P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M.C. Thom, E. Tolkacheva, C.J.S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature* 473:194-198, 2011.

[4] J. Cong, V. Sarkar, G. Reinman, and A. Bui. Customizable domain-specific computing. *IEEE Design & Test of Computers* 28: 6-15, 2011.

[5] H. Esmaeilzadeh, E. Blem, R.S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the International Symposium on Computer Architecture* (*ISCA'11)*, pp. 365-376, San Jose, CA, June 2011.

[6] E.E. Swartzlander, A.G. Alexopolous. The sign/logarithm number system, *IEEE Transactions on Computers* C24(12):1238-1242, 1975.

[7] G.E. Suh, C.W. O'Donnell, and S. Devadas , Aegis: A single-chip secure processor. *IEEE Design & Test of Computers* 24.6: 570-580, 2007.

[8] M. M .Shulaker, G. Hills, N. Patil, H. Wei, H. Y. Chen, H. S. P. Wong, and S. Mitra. Carbon nanotube computer. *Nature*, 501:526-530, 2013.

[9] A. Madhavan and D. B. Strukov. Mapping of image and network processing tasks on high-throughput CMOL FPGA circuits. In *Proceedings of IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC'12)*, pp. 82-87, Santa Cruz, Oct. 2012.

[10] F. Alibart, T. Sherwood, and D. B. Strukov. Hybrid CMOS/nanodevice circuits for high throughput pattern matching applications. in *Proceeding of Adaptive Hardware and Systems (AHS'11)*, pp. 279-286, San Diego, CA, June 2011.

[11] J. Li, R. Montoye, M. Ishii, K. Stawiasz, T. Nishida, K. Maloney, G. Ditlow, S. Lewis, T. Maffitt, R. Jordan, L. Chang, and P. Song. 1Mb 0.41 μm^2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing. In *Proceedings of Symposia on VLSI Technology and Circuits (VLSITC'13),* pp. C104-C105, Kyoto, Japan, June 2013.

[12] Q. Guo, X. Guo, Y. Bai, and E. Ipek. A resistive TCAM accelerator for data-intensive computing. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'13),* pp. 339-350, Davis, CA, Dec. 2013.

[13] 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* 309(5): 397-406, 2003.

[14] C. Dwyer, A. R. Lebeck, and D. J. Sorin. Self-assembled architectures and the temporal aspects of computing. *Computer* 38(1): 56-64, 2005.

[15] N. Jones and P.A Pevzner. *An Introduction to Bioinformatics Algorithms*. The MIT Press, Cambridge, MA, 2004.

[16] R. J. Lipton and D. Lopresti. A systolic array for rapid string comparison. In *Proceedings of the Chapel Hill Conference on VLSI*, pp. 363-376, 1985.

[17] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences* 89(22):10915-10919, 1992.

[18] 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* 48(3):443-453, 1970.

[19] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology* 147.1: 195-197, 1981.

[20] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403-410, 1990.

[21] D. T. Hoang. *A Systolic Array for the Sequence Alignment Problem*. Department of Computer Science, Brown University, 1992.

[22] D. T. Hoang and D. P. Lopresti. FPGA implementation of systolic sequence alignment. *Lecture Notes in Computer Science* 705:183-191, 1993.

[23] R. Hughey. Parallel hardware for sequence comparison and alignment. *Computer Applications in the Biosciences* 12(6):473-479, 1996.

[24] Y. Liu, W. Huang, J. Johnson, and S. Vaidya. GPU accelerated Smith-Waterman. *Lecture Notes in Computer Science* 3994:188-195, 2006.

[25] S. A. Guccione and E. Keller. Gene matching using JBits. *Lecture Notes in Computer Science* 2438:1168-1171, 2002.

[26] I. T. Li, W. Shum, and K. Truong. 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). *BMC Bioinformatics* 8(1):185, 2007.

[27] R. K. Singh, D. L. Hoffman, S. G. Tell, and C. T. White. BIOSCAN: A network sharable computational resource for searching biosequence databases. *Computer Applications in the Biosciences* 12(3):191-196, 1996.

[28] R. Mott. Alignment: Statistical significance, *Encyclopedia of Life Sciences*, 2005.

[29] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein. Scaling, power, and the future of CMOS. In *Proceedings of Electron Devices Meeting (IEDM'05)*, pp.7-15, Washington, DC, Dec. 2005.

[30] International Technology Roadmap for Semiconductors, (http://www.itrs.net/) 2005.

[31] J. J. Yang, D. B. Strukov, and D. S. Stewart. Memristive devices for computing. *Nature Nanotechnology* 8:13-24, 2013.