A Time-Based Intra-Memory Computing Graph Processor Featuring A∗ Wavefront Expansion and 2-D Gradient Control

Luke R. Everson, *Member, IEEE*, Sachin S. Sapatnekar, *Fellow, IEEE*, and Chris H. Kim, *Fellow, IEEE*

***Abstract*— A mixed-signal, time-based 65-nm application-specific integrated circuit is developed for solving shortest-path problems. Digital circuits are collocated with the memory as intra-memory computing. The core follows similar principles from wave routing and, additionally, incorporates a gradient on the periphery of the core to implement the A∗ algorithm predicted distance heuristic. A leading pulse is propagated from start nodes and is asynchronously latched in neighboring vertex cells and pushed to its four neighbors. Applications include collision avoidance for self-driving cars, shortest path planning, and scientific computing, and are shown to be scalable across many cores. The chip achieves 559 million traversed edges per second at 105× improved energy efficiency compared with existing platforms such as field-programmable gate array and CPU. The processor operates nominally at 1.79 ns per node with peak power consumption of 26.4 mW.**

***Index Terms*— A∗ algorithm, graph computing, graphs, intra-memory computing, single-source shortest path (SSSP), time-domain computing, time-to-digital converter.**

I. Introduction

SINGLE-SOURCE shortest path (SSSP) problems have a rich history of algorithm development [1]–[3]. SSSP has many applications including AI decision making, robot navigation, VLSI signal routing, autonomous vehicles, and many other classes of problems that can be mapped onto graphs. Recently, accelerators have been included in system-on-a-chips specifically targeted at autonomous vehicles [4]. These approaches rely on conventional algorithms which sequentially traverse the search space. Performance is inherently limited by von Neumann systems in traditional computer architecture. As graphs become very large, this slow processing time can become a bottleneck in real-world applications. In this article, a time-based application-specified integrated circuit (ASIC) is presented to address this issue. The design leverages a dedicated hardware implementation to solve these problems in linear time complexity and at unparalleled energy efficiencies. Intra-memory computing describes an architecture where the digital logic is collocated within the memory array. This helps reduce data movement to overcome the memory bottleneck. A 40 × 40 four-neighbor grid implements a wavefront (WF) expansion with a first-in lockout mechanism to enable traceback. Outside the array, a programmable resistive ladder provides bias voltages to the delay cells which enables pulse shaping reminiscent of the A∗ algorithm [1]. Section II begins with a primer on graph fundamentals and a survey of the prior art. Section III describes the operating principle of the ASIC and details the core circuits that enable the unique functionality. Next, Section IV will describe quantitative measurement results and a thorough process, voltage, temperature (PVT) analysis. Applications will be presented in Section V and conclusions will be drawn in Section VI. The conference version of this article was published in [5]. Contributions of this extended version include prior art discussion, expanded description of chip functionality, and additional measurement results focused on variation analysis.

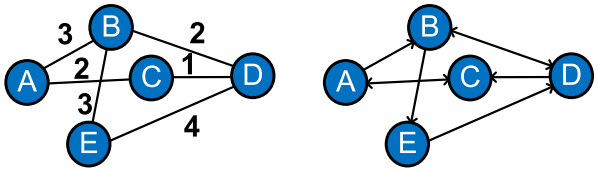


Fig. 1. Examples of an integer weighted undirected graph (left) and an unweighted directed graph (right).

II. Graph Fundamentals

A graph is a computation construction consisting of as a set of vertices and connections, or edges between them. Following edges is how graphs are traversed. One-way connections are called directed. They can also be bidirectional or undirected. Edges can have weights to bias certain paths. In Fig. 1, the left graph has weighted undirected connections. The right graph has directed edges, which can be seen by the arrows on ends of the edges. Graphs are not intrinsically interesting, but when they can be mapped onto real-world problems, well known mathematical techniques can be applied to solve a variety of problems.

A popular workhorse for SSSP is Dijkstra's algorithm [2]. Dijkstra's algorithm is classified as a greedy algorithm because

Manuscript received July 26, 2020; revised October 8, 2020 and December 15, 2020; accepted December 28, 2020. This article was approved by Associate Editor Edith Beigné. This work was supported by the National Science Foundation under Grant CCF-1763761. *(Corresponding author: Chris H. Kim.)*

The authors are with the Electrical and Computer Engineering Department, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: chriskim@umn.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2020.3048726.

Digital Object Identifier 10.1109/JSSC.2020.3048726

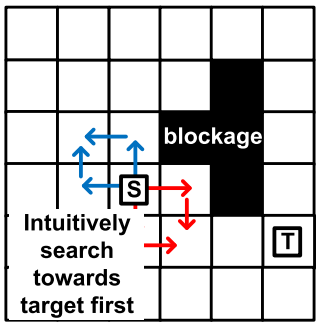


Fig. 2. Illustration of the intuition of the A∗ heuristic. S and T correspond to source and target, respectively.

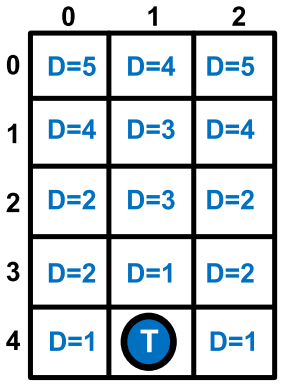


Fig. 3. Manhattan distance as the heuristic in A* SSSP. Blue numbers indicate the distance from the target, T. Gray numbers refer to the grid.

it always chooses the best available option. Nodes are added to the priority queue based on the distance to the source node. Shorter distances signify a greater likelihood to find a quicker path to the target and are serviced first. Dijkstra's algorithm is guaranteed to find the shortest route because at each node, the shortest path to it is recorded as the algorithm progresses.

One optimization for search is taking into account *a priori* information, such as searching nodes in the direction of the target first. This intuition is illustrated in Fig. 2. Cells in the direction of the arrows pointing toward T clearly should be searched before those cells in the direction of the arrows pointing away from T. This is precisely the modification that the A∗ algorithm [1] makes to Dijkstra's algorithm

$$\text{Cost}(n) = F(n) + H(n). \tag{1}$$

The cost assigned to a node, n, in the priority queue is shown in $(\text{Cost}(n) = F(n) + H(n))$ (1). $F(n)$ is the actual distance of the current node from the source; the cost assigned in Dijkstra's algorithm. $H(n)$ is a heuristic that predicts the distance of the current node to the target. A familiar distance heuristic is the Euclidean distance or straight-line path.

Fig. 3 details the Manhattan distance heuristic, commonly used in graph computing, named after the uniform grid streets in Manhattan, New York City. Outside the grid, the coordinates in both directions are listed. The predicted cost of the distance at any given point to the target is the sum of the differences of their horizontal and vertical coordinates. The relationship between these algorithms, the processor test chip, and the applications will be seen in Section III. First, we will discuss current state of the art work.

A. Prior Art

Due to the prominence of graphs in computing, there is an incredible breadth of work in the literature. The key focus of many of these works is to reduce the memory bottleneck common in graph applications. One-way Graphicionado [6] overcomes this limitation is by only accessing four bytes for each vertex compared with the convention 64 bytes, which reduces scratchpad storage. The authors use a special processing pipeline that is reconfigurable for workload-specific functions to extract parallelism from large-scale graph problems. The pipeline functions can be broken down into two phases; processing and apply. In the processing phase, every edge is operated based on software defined functions. Updates are stored in a temporary vertex property array. The update phase is initiated when all active vertices have been serviced. The update phase writes these temporary properties along with the existing properties of the vertex. Graphicionado custom modules are implemented for these two phases. The pipeline consists of nine processing stages and the apply pipeline is five stages deep. The pipeline uses prefetching in order to have all off-chip memory accesses sequential, meaning they do not have dependencies on any other pipeline stages. In addition, they apply slicing to transform graph structure to leverage symmetry to update source and destination vertex properties simultaneously. The scale at which [6] operates is vastly different compared with this work, as it is evaluated in a software version on a 16-core Xeon server with external DDR4 DRAM. The ASIC version is simulated using CACTI with a 64 MB on chip cache.

GraphP [7] is another example of a high-performance graph accelerator architecture. GraphP uses processing in memory to reduce data movement, which is enabled by 3-D stacking of memories. They develop a vertex-centric model, featuring a partition structure designed for "source-cut" partitioning to reduce storage overhead. Due to the unique physical structure, the authors implement an overlapping communication hierarchy for property updates. The overlapping comes from updates when the vertex has been partitioned. A key feature of the communication is that it is non-blocking. Local cores can compute intermediate results without communication latency to bypass the memory bandwidth limitation.

The memory storage bottleneck is not limited to CPUs. Field-programmable gate arrays (FPGAs) are typically incapable of storing large graphs in SSSP. Lei *et al.* [8] developed an extended systolic array priority queue (ExSAPQ) for large-scale processing. Since they store information off-chip, they are required to store duplicate copies of the look-ahead parameter for each of the processing elements to prevent data collisions similar to [7]. FPGAs have inherent parallelism, which is leveraged by having concurrent execution. The ExSAPQ ensures that no overflow happens when elements are shifted off-chip. In all three of these examples, data movement is the key design consideration.

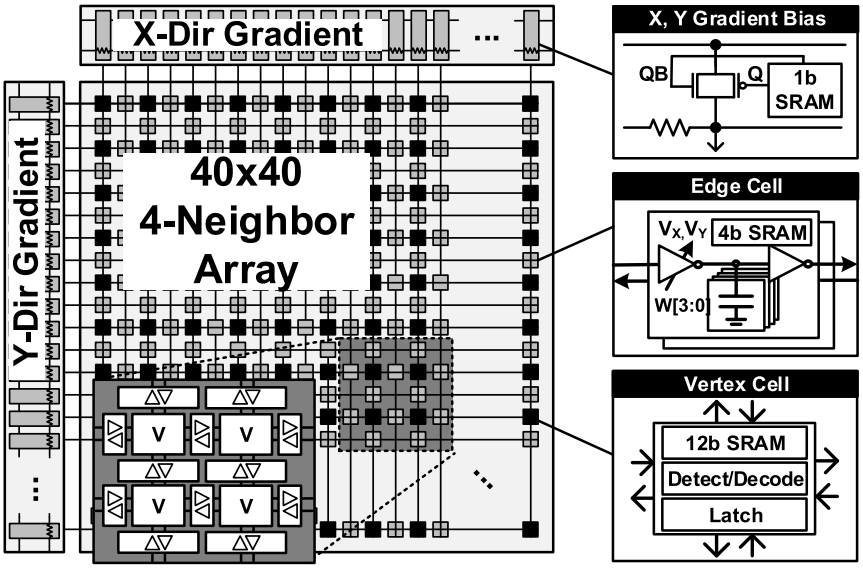


Fig. 4. 40 × 40 graph ASIC chip for solving SSSP problems based on 2-dimensional WF expansion [5].

As will be seen in this article, our A∗ chip positions the processing inside the memory storage to eliminate the need to store duplicate copies of the data and waste time reading the scratchpad. Although the A∗ chip can only utilize four neighbor connections, the speed of evaluation and efficiency of memory-compute co-location help mitigate this limitation. Modest architecture changes to accommodate diagonal routing would include four additional I/O ports on each vertex and extra metal routing. The key time-based idea would hold in this expanded case.

III. Time-Based Graph ASIC

A schematic of the chip is shown in Fig. 4. The architecture of the chip is based on a 40 × 40 4-neighbor grid. The dimensions were chosen to maximize the test chip area. The 1600 vertices have connections in the cardinal directions to adjacent vertices (N, S, E, and W). The key principle is the proportional relationship between computation time and the distance between nodes. This is accomplished by propagating a pulse between the vertices through the edges. Detailed care was taken during placement and layout to keep all routes matched to preserve this relationship. The gradient on the periphery of the array implements the distance biasing. Each vertex operates autonomously and asynchronously; it senses and stores the direction of the input pulses, prevents other pulses from overwriting it through the lockout mechanism, and propagates it to neighboring stages. The first pulse to arrive latches the cell. It can also accept a set of simultaneous input pulses without functional errors. The direction is decoded and stored locally for readout and represents the fastest way to reach that cell. Since the first pulse is the only pulse latched in each cell, tracing the pulse chain back to the start will reveal the shortest path. If more than one input gets latched, either of the inputs can be selected through a simple trace-back algorithm. At runtime, an arbitrary graph can be programed by the user. Initially, the core was designed for SSSP, but each evaluation contains all shortest paths to the start node realizing a significantly more computationally intensive task. The parallel and autonomous vertex control allows this to happen in linear time. Section III-A–III-C will describe each of the core circuit blocks in greater detail.

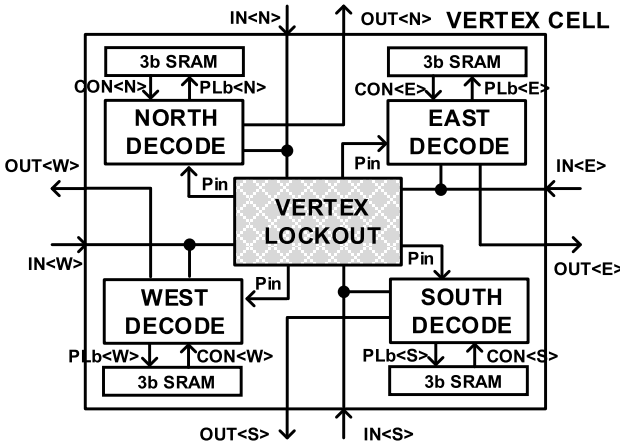


Fig. 5. Block diagram of the vertex. Modified from [5].

A. Vertex Cell

Fig. 5 shows the block diagram of the vertex cell. The vertex consists of four I/O ports, a lockout mechanism, and directional decoders. Storage is allocated as follows: four bits for enabling output pulses for each direction, four bits for storing which input arrived from that edge, and four bits for control, including if the vertex starts the pulse. These cells are distributed spatially with each direction circuit block. Cells are accessed by tapping out the Q/Qb storage nodes directly. The key lockout operation is described in Fig. 6. An input is presented to the vertex from the north which latches the vertex, and pulses are passed to the edges. In addition, late

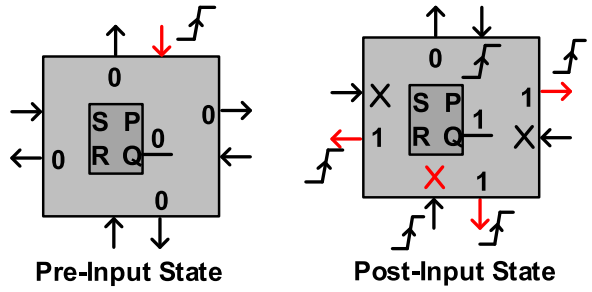


Fig. 6. Operation of the lockout mechanism. Input from North (left) causes outputs to be generated and blocks south input (right). State changes are highlighted in red. Modified from [5].

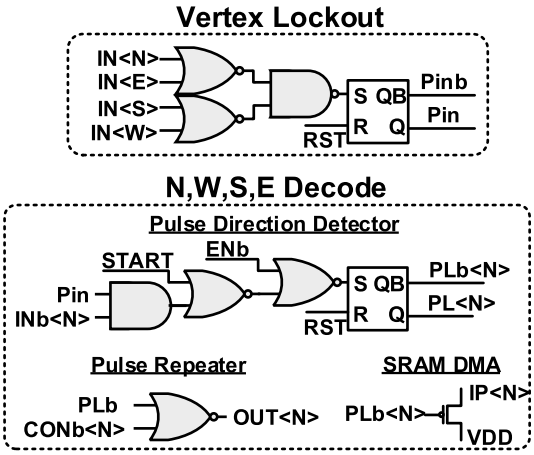


Fig. 7. Vertex circuit schematic. Modified from [5].

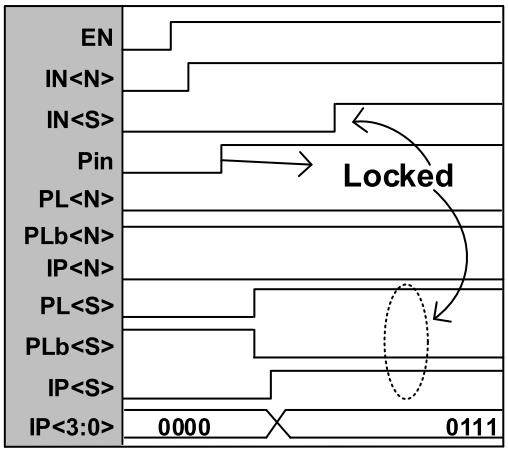


Fig. 8. Vertex timing diagram of lockout functionality. Modified from [5].

inputs are blocked from overwriting the cell state. Fig. 7 shows the schematic of the vertex cell. A NOR tree merges inputs to enable lockout. This is realized with the SR latch in the Pulse Direction Detector in Fig. 7. Once the pulse has been decoded, it is stored locally with a direct memory access (DMA) circuit. The output is propagated to the output by the pulse repeater if the cell is programed with a connection to that edge.

Fig. 8 shows a vertex timing diagram based on the following example of two pulses arriving: North first and then South.

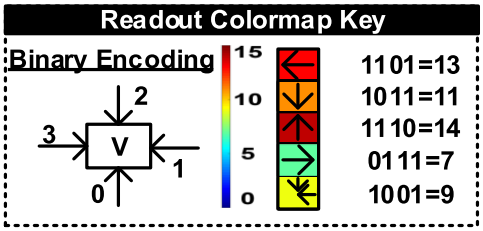


Fig. 9. Local SRAM storage encoding for traceback. Modified from [5].

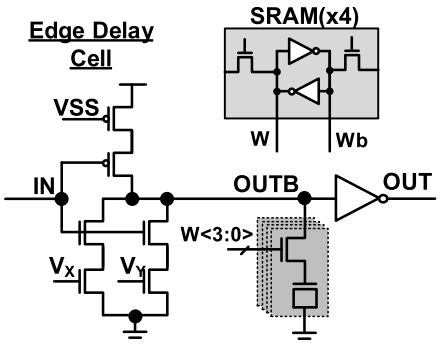


Fig. 10. Edge unit schematic modified from [5].

First, a global enable signal, *EN*, is asserted. This enables the core, which triggers a pulse in the array. The four inputs are merged together in a detection circuit to determine whether a pulse has arrived in the cell. In the example, *IN<N>* will flow through and latch P_{IN}, or Pulse Input. P_{IN} is compared with the input from each of the four directions. If P_{IN} is asserted and the input is not from the direction that asserted P_{IN}, Pulse Latch, in this example *PL<{S, E, W}>*, will assert. *PLb* is connected to the SRAM DMA which will flip the *IP<{S, E, W}>* SRAM that will be read out after evaluation during the path traceback. *IP<3:0>* is initialized low and the directions not activated are programed high after a pulse arrives by the DMA circuit. The Readout Colormap Key in Fig. 9 shows this encoding and will be carried through the other figures in this article. In this example, the input arrived from North, or "2," giving the stored cell value 1011_2, or 11_{10} encoded as dark orange. Last, the pulse is propagated to the neighbors that were not responsible for latching the cell and have a connection stored in their respective local SRAMs. This is where the "intra-memory computing" is realized; all circuits, connections, and pulse propagation occur inside the footprint of the physical memory array. Blockages are implemented as all outgoing connections disabled. Anytime a cell is read, it does so by tapping off the Q or QB, and the write happens locally by the SRAM DMA circuit in Fig. 7. The layout of the chip is constructed based on an SRAM architecture, which enables regular routing and high area utilization.

B. Edge Unit

The edge unit passes the pulse between vertices via a modulated delay through the analog gradient ladder and local cap loading. The edge schematic is shown in Fig. 10.

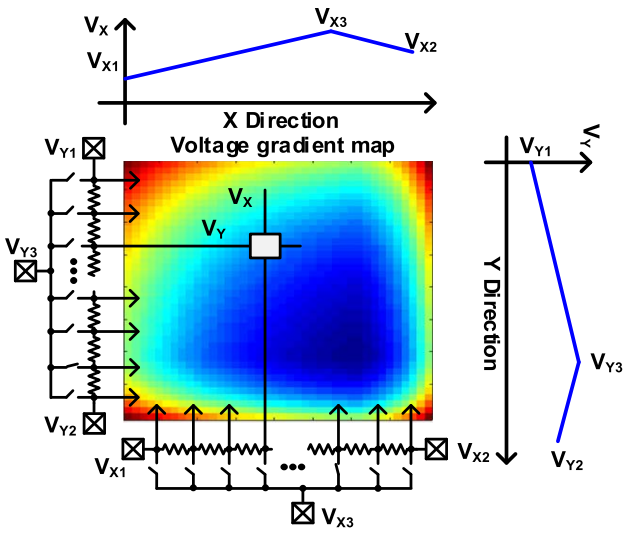


Fig. 11. Functional example of the gradient ladder. Modified from [5].

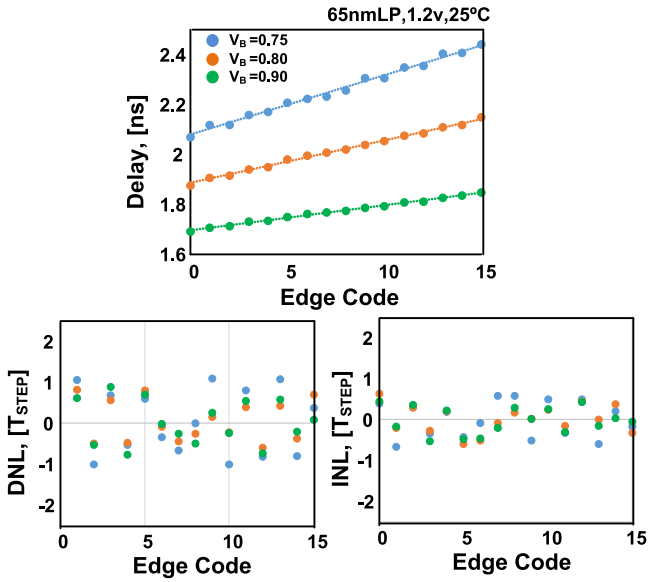


Fig. 12. Measured delay and linearity of the edge cell.

The delay is modulated by loading OUTB with 4 b MOSCAPs with the gates of the switches tied to local SRAMs. These 4 b weights are programmed at runtime based on the values from the map, and could also be used to compensate for process variation. 4 b may not be enough dynamic range for some applications, but it is complemented by the bias voltage gradient (V_X and V_Y) applied to each branch of the first inverter. The bias voltage values are tapped off the resistive ladder, which is proportional to the position of the edge in the array determined by the gradient. This ladder will be described in Section III-C. There are two NMOS branches to enhance the current summation linearity. Shorting the drains could have resulted in unequal contributions of the currents across different locations in the array. As the gate voltages of the NMOS increase, this would have reduced V_{DS} which would have had an outsized effect on the gate with the lower V_G.

C. Gradient A∗ Mapping

Fig. 11 shows how the gradient cell implements the predicted distance to the target. Two adjacent sides have a resistive ladder with taps at each row/column that span the entire core array. Since this is a 40 × 40 array, there are 41 resistors on each side. In addition, switches controlled by a register chain are placed between each resistor to set the voltage to *V3* (max voltage in Fig. 11). On the ends of the resistive ladder, there are two additional bias voltages (*V1* and *V2*). The difference between *V3* and the other two voltages is linearly dropped across each resistor, illustrated in the line graphs. Each connection should have input impedance from the MOSFET gate and draw very little current, thus current should flow only in the ladder. Only one tap of the resistive ladder is connected to *V3*, but an additional connection would provide an unbiased section of the graph. The position of the *V3* connection corresponds to the location of the target, determined by the application requirements. In Fig. 11, the delay of each cell based on the bias voltages seen by the X-direction and Y-direction gradient is encoded in the color of the cells in the array map. The darker colors correspond to the fastest cells because the maximum bias voltage is at *V3*. As the distance increases from the target cell, the bias voltage decreases causing the delay to increase. This essentially accelerates the pulse toward the destination in the same manner that the A∗ algorithm reduces the predicted cost as the search gets closer to the target node.

IV. Measurement Results

A. Edge Cell Delay Measurements

Fig. 12 contains the measurement results from the edge cell delay. Fig. 12 (top) contains a plot of average edge cell delay versus digital edge code for three different bias voltages. Over the three bias voltages shown in Fig. 12, it is possible to see how the delay can be modulated across the array by the gradient resistive ladder framework introduced in Section III-C. On the bottom, differential nonlinearity (DNL) and integrated nonlinearity (INL) are shown. DNL measures the difference between consecutive tuning levels, while INL is the difference from measured to the ideal level. The Y-axis represents these values normalized to one unit delay step. The deviations in the DNL and INL appear to toggle between positive and negative. This could be due to the least significant bit in the capacitor bank not having a significant contribution to the delay. This is apparent in the $V_B = 0.75$ V trace in the left plot. Every other point has a minimal increase in the delay; edge code two and three have nearly identical delays. In future designs, this could be rectified by increasing the size of the access switch to increase the effect of the connected capacitive load.

Due to the lack of high precision measurement circuits on the chip and no ability to probe internal points during pulse propagation, a unique strategy was required to generate the delay curve on the left. The graph was programmed to have only a single route across the chip, shown in Fig. 13(a). The path started in the top left and traversed across the array to

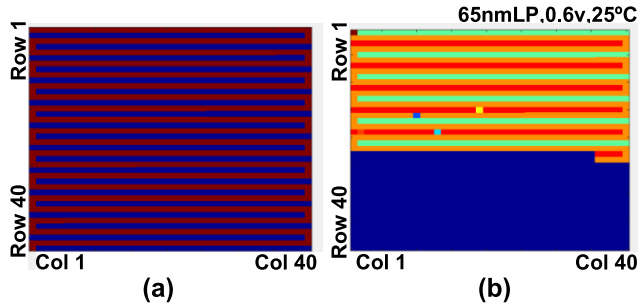


Fig. 13. (a) Programed map of the array. Single path is programed in a zigzag pattern. (b) Readout from chip with disabled after 2.75 μs.

the right (light green trace) until the edge was reached. The path moved down one unit and then proceeded back across the core (dark orange trace). This was repeated over each row in the array to yield the longest single path possible in the chip. During measurement, the enable signal was precisely controlled by off-chip test equipment. The total time enabled was divided by the number of cells that the pulse traversed. In the case of Fig. 13(b), the bias voltage was set to 0.6 V and the pulse traveled through 446 cells, yielding an average delay of 6.1 ns/vertex. This process was repeated for different edge codes and bias voltages. To apply the same bias voltage to all cells, *{V1, V2, V3}* were all set to the same voltage and no connection was made on the gradient ladder.

B. Resistive Ladder Voltages

The voltage steps in the resistive ladder circuit were optimized to increase the pulse propagation time relative to the detection window without compromising the linearity of the delay control. The detection window width is only two inverter delays (setting the RS latch in Fig. 7), which was small enough to remove the simultaneous arrivals during our testing using the optimized voltage steps. For very large arrays with numerous iso-delay paths; however, multiple inputs may still arrive within the detection window even with careful voltage step tuning. In this case, either of the paths can be chosen as an acceptable solution as the objective of path finding algorithms is not necessarily to find the golden solution but to find an acceptable solution.

It is worth noting that the outgoing signals will be launched according to the Pin signal timing (Fig. 7) and the SRAM cell values. For instance, if two signals arrive and get latched in the SRAM cells before the Pin signal fires, the other two outgoing signals will be launched. An additional post-processing step would be required to resolve the multiple input paths with same delay. This could be achieved by tracing the parent paths of each input, and if they share a common parent node, the path with fewer steps could be assigned as the shorter path. Further investigation on different post-processing methods would be beneficial.

C. Process, Voltage, and Temperature Analysis

In time-domain computing, it is important to consider PVT variations as it can affect the computation results. Next, we

TABLE I
MONTE CARLO EDGE DELAY SIMULATION

Monte Carlo=500, 65nm LP, 1.2V, 25° C

V_B ,[V]	μ ,[ns]	+3σ ,[ns]	+3σ/μ
0.6	0.569	1.23	2.16
0.9	0.099	0.12	1.25
1.2	0.054	0.065	1.20

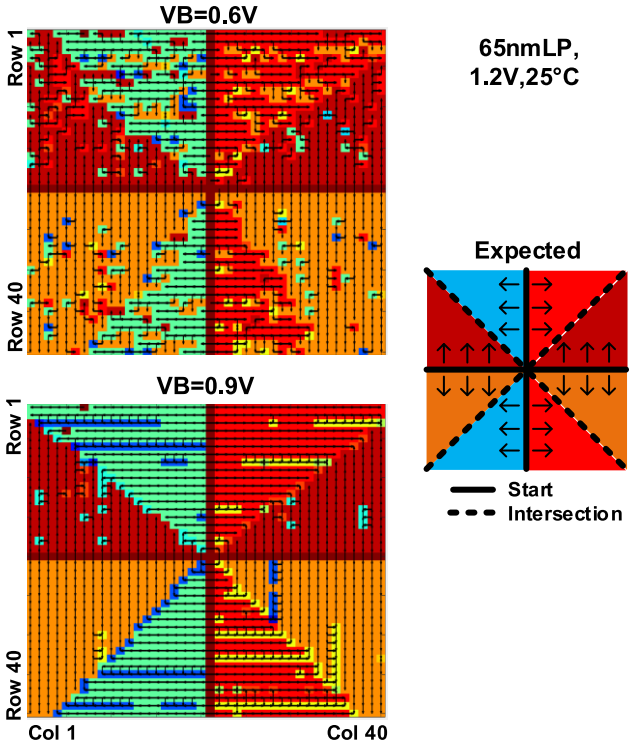


Fig. 14. Plus sign measured experiment shows the larger delay spread at lower voltages seen by more variation in the triangle slices.

discuss the impact of each type of variation on the operation of the proposed graph ASIC.

1) Process Variation: The proposed design is inherently tolerant to global process variation since all vertex and edge delays will be affected by the same amount. On the other hand, local process variation may result in an error in the arrival order for paths with similar delays. However, these errors are not real concerns since either paths can be chosen as the solution because of the modest impact (e.g., $<10\%$) of process variation on the path delays. Furthermore, the ratio of the standard deviation to the mean (σ/μ) of the path delay will decrease for longer paths comprising more delay stages so the overall impact on the final solution will be even more modest.

2) Voltage Variation: Supply voltage and bias voltage variation is critically important in this design, due to our leveraging of current starved inverters to set the delay of the edges, while implementing the gradient function. Table I lists the simulated delays of the current starved inverter edge cells for different bias voltages. The ratio of the standard deviation to the mean increases inversely with V_B. Fig. 14 shows an experiment

TABLE II
COMPARISON TABLE [5]

Architecture	This Work	FPGA [9]	μP	CPU [10]	GPU [1]
Product	ASIC	Xilinx Virtex	ARM Cortex-M0	Intel Xeon E5630	NVIDIA Tesla K20c
Technology	65nm	20nm	40nm	32nm	28nm
Voltage	1.2V	-	1.1V	0.7-1.35V	-
Peak Power	26.4mW	24.22W	127μW	20W/core	225W
Throughput [MTEPS]	559	731	$5.34\text{x}10^{-4}$	0.83	9.0
Energy per Node	0.328pJ*	33nJ	89.1nJ	24.1μJ	25μJ
Normalized Energy	1x	10^5	$2.7\text{x}10^5$	$1.19\text{x}10^6$	$2.3\text{x}10^7$

*55% from SRAM program (does not include cache access energy)
Energy/Node=Unit Delay*Unit Power, MTEPS = Million Traversed Edges Per Second

designed to show how the increased variation causes errors. Fig. 14 (top) was generated at $V_B = 0.6$ V (larger variation), and Fig. 14 (bottom) was generated at $V_B = 0.9$ V (reduced variation). Start cells are programed in the center in a plus sign and the pulse is launched at the same time. When all voltages $\{V_1, V_2, V_3\}$ in the gradient ladder are set to V_B, the pulse expands at a constant velocity. It is apparent that Fig. 14 (top) has more variation than Fig. 14 (bottom). This manifests itself as the sprinkling of different colors in the map. Ideally, each of the eight quadrants should be a solid color, so any deviation from that is considered an error. Recall that Fig. 9 describes the color encoding to the direction of the input pulse. Fig. 14 (bottom) patterns are streaks of different colors, which represent two different paths having equal time of arrival at those points. This is more of a static offset, not a random variation. Reducing the supply voltage would have a similar effect, further increasing the variation and introducing errors. This effect could be removed completely in our testing by increasing the bias voltage. The tradeoff is the reduced dynamic range of the delay. As seen in Fig. 12, the slope of the delay reduces at higher V_B. The total delay of the cell is equal to the following equation:

$$t_{\text{Total}} = t_{\text{Vertex}}(V_{\text{DD}}) + t_{\text{Edge}}(V_{\text{DD}}, V_B). \tag{2}$$

As V_B is increased, t_{Edge} decreases, while t_{Vertex} remains constant. This means the effect of the gradient on t_{Total} reduces. The specific application constraints should inform the user of the target V_B to achieve a balance in noise and gradient impact.

3) Temperature Variation: Temperature effects on performance should be muted in this architecture for several reasons. First, the distributed asynchronous evaluation methodology of the proposed design decentralizes the key operations in the chip resulting in reduced local heat generation. If one considers a case where the start point is the center of the array with no blockages, the pulse begins at the center of the chip and radiates concentrically to the edges of the chip. The number of cells evaluated at each time step increases at a fixed step, the perimeter of a square or *4n*. The area over which these cells are dissipating power is n^2. The power density decreases proportional to *4/n* making the self-heating impact modest compared with circuits with higher activity factors. Second, global temperature variation has little impact on the accuracy of the path finding operation as all propagation delays of the array will be affected in the same way. Third, local temperature variation such as hotspots will not have a significant impact on the local path finding operation as the delay stages in the vicinity will be affected in the same way. Further simulation studies would be required to quantify the effects of global and local temperature variation, but our first order analysis suggests that the basic operation of the proposed graph ASIC is inherently tolerant to temperature effects.

D. Comparison Table and Chip Summary

Table II is mainly included for general comparisons between architectures. References [9] and [10] are comparable in the sense that they attempt to map hardware platforms onto optimal A∗ implementations. It should be noted that the current version of our A∗ core does not account for diagonal movements or post-processing required to accommodate this. Our peak power is quoted when all 156 perimeter vertices are evaluating corresponding to the pulse originating from the center, equating to 183.1 μW/vertex. 55% of the power is due to SRAM access storing the pulse information *in situ*. During the program, there is short-circuit current in the two cross-coupled inverters of the SRAM. This fact highlights how low the actual compute energy is; each vertex is evaluated by toggling a few gates and two inverters on the edges. Compared with the state-of-the-art FPGA [9], μP, CPU [10], and GPU [10] implementations, our core has roughly five orders of magnitude superior energy efficiency. The energy does not include generating the bias supply or readout phase. Because the answer is encoded in the spatial positioning, the array needs to be read once, whereas in a conventional approach, the memory would have to be read many more times to retrieve data for processing.

Fig. 15 shows the die photo of the test chip. Table III highlights the chip summary. The chip is fabricated in a 65-nm LP CMOS process. The 40 × 40 array contains 1600 vertices and 6400 edges. The perimeter vertex cells have dummy edges to provide equal loading. The delay resolution on each edge is a 4 b binary weighted capacitor bank in conjunction with the analog bias gradient, which implements the A∗ heuristic. At 1.2 V, the peak power is 26.4 mW (simulated), which corresponds to every cell on the edge evaluating when a pulse

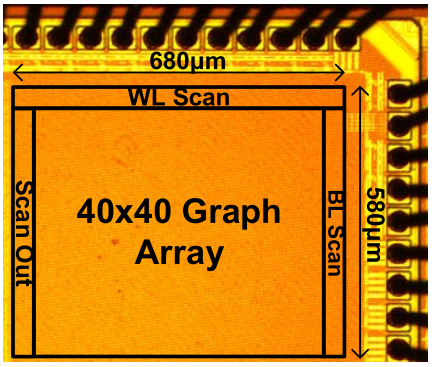


Fig. 15. Die photograph. Modified from [5].

TABLE III
CHIP SUMMARY. MODIFIED FROM [5]

Applications	A* shortest path, obstacle avoidance, scientific computation (optics)
Technology	65nm LP CMOS
Architecture	Time-based
# of Vertices	40x40=1600
# of Edges	6400
Edge Resolution	4b + Analog Gradient
Voltage	1.2V
Peak Power	26.4mW (simulated)
Delay per Node	1.79ns @ [V_B=.9V, V_{DD}=1.2V] (meas)
Power per Node	183.1µW (simulated)
Energy per Node	0.328pJ (simulated)

is started from the center of the chip. The delay per node is 1.79 ns at an edge bias of 0.9 V consuming 183.1 μW.

V. APPLICATIONS

A. Collision Avoidance via Voronoi Diagrams

The principle of the Voronoi algorithm is that it segments a plane such that the partitions represent the closest position to a start seed. In this ASIC, Voronoi diagrams are solvable when no bias gradient is applied. When there is no gradient, the pulse travels at a uniform velocity. This is precisely what k-nearest neighbor classification in machine learning computes. Voronoi diagrams also model various natural structure formations such as bone structure and cell orientations. Computational fluid dynamics meshes are also generated using these principles. Collision avoidance for autonomous vehicles can also be mapped to Voronoi Diagrams [11].

Fig. 16 highlights collision avoidance scenario. Pulses are simultaneously launched from the sides of the obstacles (shown as black boxes). Obstacles are implemented as having all of their connections to adjacent cells disabled. Shown in white is the path that maximized the distance between the obstacles. In the bottom left of Fig. 16, the boundary between obstacles "1" and "2" is highlighted. The white line is added as a visual aid, but it is clear that the WFs met between the two blockages. The other callout shows three WFs intersecting. Along the top and side of the main array, the bias voltages for

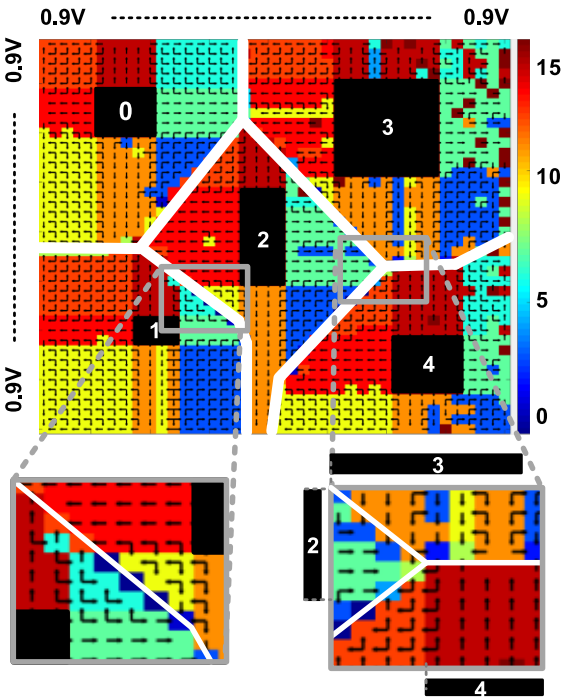


Fig. 16. Collision avoidance example. Modified from [5].

V1 and *V2* are equal at 0.9V. This application leverages the autonomous vertex operation, vastly reducing the evaluation time compared with a standard von Neumann machine, which services each vertex serially.

B. Shortest Path Planning

SSSP motivated the development of this ASIC. Fig. 17 solves SSSP from the top left (start, S) to the bottom right (target, T). Finding a shortest path in a grid without any blockages is trivial, as shown in Fig. 17(a). Every down & rightward move will have the shortest path. This is also seen by looking at the distance from S stored in each node, and following an increasing path. However, when blockages are present, it becomes a more interesting problem. In the case of Fig. 17(b), all paths above the blockage in red dominate the paths in green that go below. This is because the lower path is required to move exactly left one unit, instead of always down and right path in Fig. 17(a). Shown in blue is where the two paths meet. In addition, Fig. 17(c) solves the problem with a gradient in the A$*$ framework. The cost to move between the nodes is no longer a uniform step, but the sum of the gray "voltage" in the row and column. The gradient improves the dominant paths because the equal point has moved one unit to the left (9 versus 12.2) yielding a more efficient route.

In SSSP, Fig. 18 shows this example applied to the chip under different bias conditions. In this application, a map is shown with blockages shown as black blocks. The WF is initiated in the upper left and propagates down and across the core. Fig. 18 (left) does not have a voltage gradient applied and each edge has the same weight. This gives the WF a very

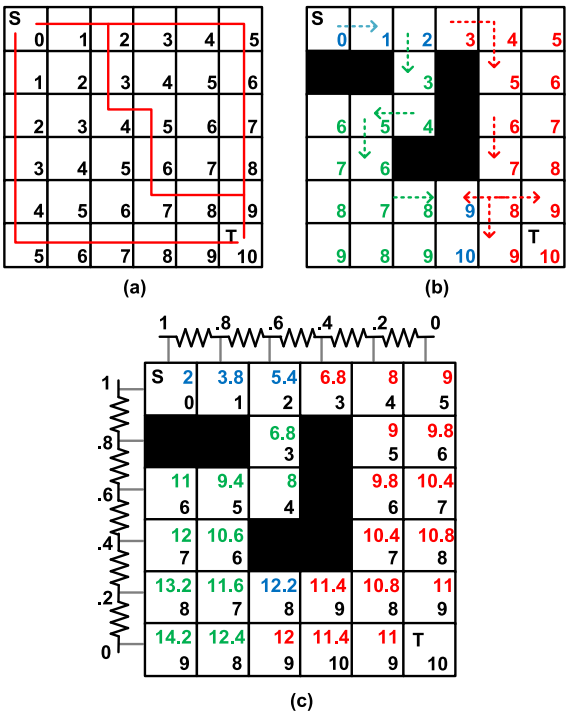


Fig. 17. Path planning on grids (a) costs from Dijkstra's algorithm, (b) Dijkstra's algorithm with blockages, and (c) A∗ via gradient.

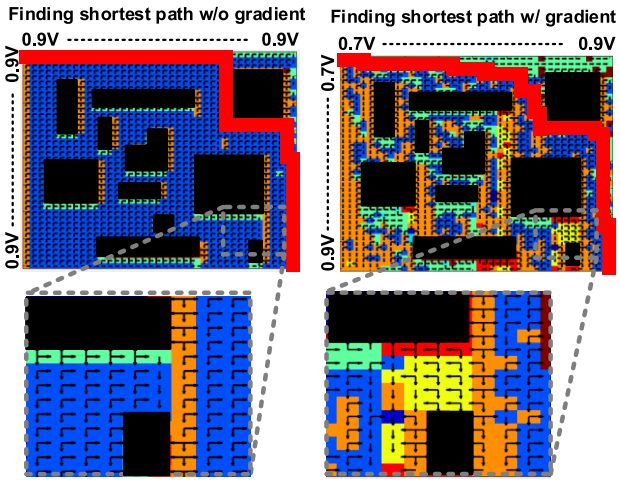


Fig. 18. Measured results from path planning application without (left) and with (right) the gradient. Modified from [5].

regular pattern as it traverses the map. Fig. 18 (right) has a voltage gradient applied that is weakest in the top-left corner and strongest in the bottom-right corner. The key difference between the two outputs is shown in lower callouts. Without the gradient, paths above and to the right of the blockage supersede any paths under as shown by the orange strip and the absence of arrows crossing into those paths. With the gradient in Fig. 18 (right), the orange path is still present, but the "above WF" is significantly faster than the "under WF" so much so that it begins to wrap underneath the blockage shown by the yellow cells. This is precisely the same observation seen in the analytical example in Fig. 17. Further study could be applied to modify the architecture so that data could be shifted in each

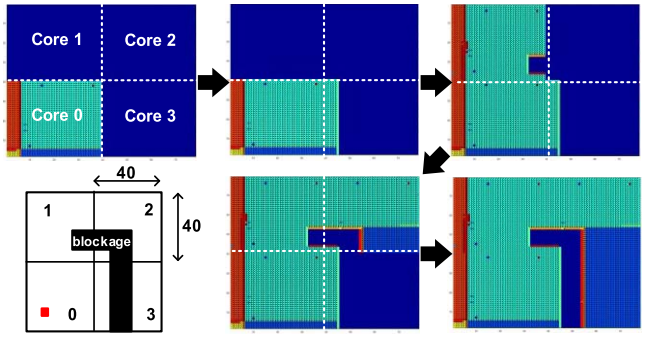


Fig. 19. Four-core example with time-multiplexed outputs interleaved to shown full map. Modified from [5].

direction to act like a streaming processor to update the map in real time.

C. Multicore Scalability

This ASIC is not limited to problems that map to a 40 × 40 grid. Fig. 19 highlights the scalability of this core via a four-core measured example with a single blockage spanning three cores. A simple rule to pick the vertex cells that have a single input pulse whose direction is opposite the boundary is used to determine the next start points in the following core. In this example, Core0 starts the pulse, Core3 is partially searched, and then Core1 is explored. It finds a new starting point in Core2, which causes the remainder of Core3 to be searched. The preceding analysis only is valid when all cells have uniform delay. It is important to note that multicore still solves the SSSP problem in linear time as the grid size expands. Any delay differences would require a more complex framework to timestamp pulse arrivals. It can be safely assumed that any inter-chip communication will incur a large overhead relative to the WF propagation in the single core. One option to mitigate this could be to time-multiplex a single core. The difference in the time to load a new map into the array and evaluate it could be compared with the energy and overhead from communicating with another die to determine optimal use case. In addition, optimizing the array size including using fewer elements in the array could control variation impacts at a cost of higher controller/communication overhead. This is a study well suited for future exploration including: a simple processor to determine candidate start nodes, a router framework for communicating between cores, and system design to ensure intelligent allocation of board-level resources.

D. Optics Experiment

Motivated by [3], this ASIC can also model optics experiments in Manhattan geometries to illustrate the utility of our hardware architecture. The primary limitation is that natural light follows Euclidean geometry. Fig. 20 shows a sampling of the core readout at different time points during a single evaluation of a two-slit experiment. The colors encode the time at which the cell has been activated. The map of the programed blockages is shown on the left. Note that in the top-middle frame, no cells are searched below the first boundary. The bottom-left frame is interesting in that the WF is split into two

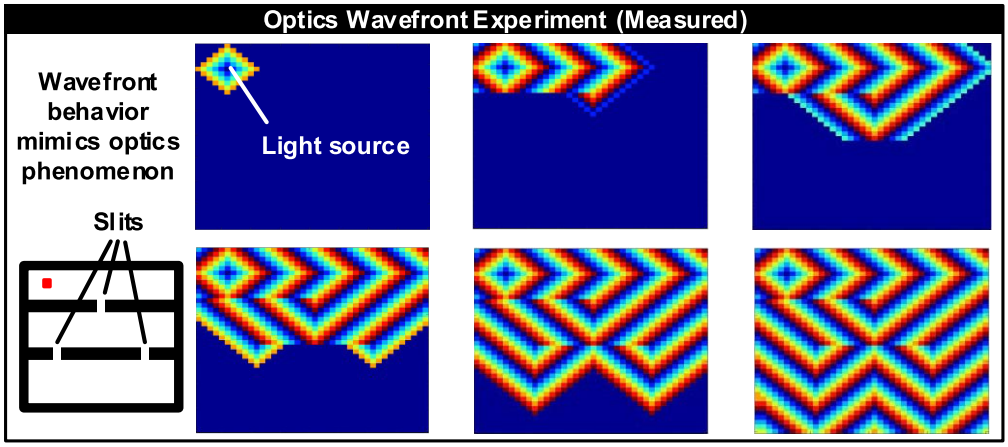


Fig. 20. Measured optics WF experiment. Evaluation at intermediate time points is used to visualize propagation. [5].

leading points. The resulting optical interference or collisions is easily modeled in this ASIC. This behavior mimics what is reported in [3] and has interesting consequences for future physical explorations of novel applications with low-power CMOS.

VI. Conclusion

In this article, an intra-memory computing ASIC graph processor was detailed. The 40 × 40 array of asynchronous processor vertices was used to solve SSSP. Each vertex operates asynchronously, which enables this chip to break the von Neumann bottleneck and solve SSSP in linear time as grid size increases. The circuit design details were discussed for the key blocks. Measured edge delay was presented, as well as a comparison table with conventional platforms used to solve the SSSP. A diverse stable of applications was demonstrated to highlight the utility of the core. Scalability concerns were addressed with a multicore experiment or discussion of time-multiplexing.

References

[1] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[3] C. Y. Lee, "An algorithm for path connections and its applications," *IEEE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 346–365, Sep. 1961.
[4] V. Honkote *et al.*, "A distributed autonomous and collaborative multi-robot system featuring a low-power robot SoC in 22 nm CMOS for integrated battery-powered minibots," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2019, pp. 48–50.
[5] L. R. Everson, S. S. Sapatnekar, and C. H. Kim, "A 40 × 40 four-neighbor time-based in-memory computing graph ASIC chip featuring wavefront expansion and 2-dimensional gradient control," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2019, pp. 1–11.
[6] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Taipei, Taiwan, Oct. 2016, pp. 1–13.
[7] M. Zhang *et al.*, "GraphP: Reducing communication for PIM-based graph processing with efficient data partition," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Vienna, Austria, Feb. 2018, pp. 544–557.
[8] G. Lei, Y. Dou, R. Li, and F. Xia, "An FPGA implementation for solving the large single-source-shortest-path problem," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 5, pp. 473–477, May 2016.
[9] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on FPGA," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2016, pp. 103–110.
[10] Y. Zhou and J. Zeng, "Massively paralllel A* search on a GPU," in *Proc. Conf. Artif. Intell. (AAAI)*, 2015, pp. 1248–1254.
[11] O. Takahashi and R. J. Schilling, "Motion planning in a plane using generalized Voronoi diagrams," *IEEE Trans. Robot. Autom.*, vol. 5, no. 2, pp. 143–150, Apr. 1989.

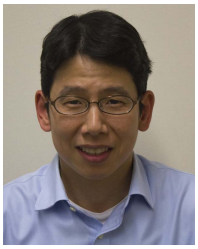
Luke R. Everson (Member, IEEE) received the B.S.E.E., M.S.E.E., and Ph.D. degrees from the University of Minnesota, Minneapolis, MN, USA, in 2015, 2016, and 2019, respectively.

He is currently a Memory Designer with the Central Engineering Group, Broadcom Inc., San Jose, CA, USA. He has coauthored over 20 articles in a diverse range of circuit topics ranging from machine learning, neural recording, radiation effects, and architecture.

Sachin S. Sapatnekar (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, the M.S. degree from Syracuse University, Syracuse, NY, USA, and the Ph.D. degree from the University of Illinois, Champaign, IL, USA.

He taught at Iowa State University from 1992 to 1997 and has been at the University of Minnesota since 1997, where he holds a Distinguished McKnight University Professorship and the Robert and Marjorie Henle Chair.

Dr. Sapatnekar was a recipient of the eight conference Best Paper awards, a Best Poster Award, two ICCAD ten-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence award, and the SIA University Researcher Award. He is a fellow of the ACM.



Chris H. Kim (Fellow, IEEE) is currently a Professor with the University of Minnesota, Minneapolis, MN, USA.

Dr. Kim was a recipient of the Taylor Award for Distinguished Research, SRC Technical Excellence Award, Council of Graduate Students Outstanding Faculty Award, NSF CAREER Award, Mcknight Foundation Land-Grant Professorship, 3M Non-Tenured Faculty Award, DAC/ISSCC Student Design Contest Awards, IBM Faculty Partnership Awards, IEEE Circuits and Systems Society Outstanding Young Author Award, ISLPED Low Power Design Contest Awards, and ISLPED Best Paper Awards. His group has expertise in digital, mixed-signal, and memory IC design, with an emphasis on circuit reliability, hardware security, memory circuits, radiation effects, time-based circuits, beyond-CMOS technologies, and machine learning hardware design..