Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

High-throughput Computation of Shannon Mutual Information on Chip

Peter Zhi Xuan Li*, Zhengdong Zhang*, Sertac Karaman, Vivienne Sze
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
Emails: {peterli,zhangzd,sertac,sze}@mit.edu
These authors contributed equally to this work

***Abstract*—Exploration problems are fundamental to robotics, arising in various domains, ranging from search and rescue to space exploration. Many effective exploration algorithms rely on the computation of mutual information between the current map and potential future measurements in order to make planning decisions. Unfortunately, computing mutual information metrics is computationally challenging. In fact, a large fraction of the current literature focuses on approximation techniques to devise computationally-efficient algorithms. In this paper, we propose a novel computing hardware architecture to efficiently compute Shannon mutual information. The proposed architecture consists of multiple mutual information computation cores, each evaluating the mutual information between a single sensor beam and the occupancy grid map. The key challenge is to ensure that each core is supplied with data when requested, so that all cores are maximally utilized. Our key contribution consists of a novel memory architecture and data delivery method that ensures effective utilization of all mutual information computation cores. This architecture was optimized for 16 mutual information computation cores, and was implemented on an FPGA. We show that it computes the mutual information metric for an entire map of $20m \times 20m$ at $0.1m$ resolution in near real time, at 2 frames per second, which is approximately two orders of magnitude faster, while consuming an order of magnitude less power, when compared to an equivalent implementation on a Xeon CPU.**

I. Introduction

Robotic exploration problems arise in various contexts, ranging from search and rescue missions to underwater and space exploration. In these domains and beyond, exploration algorithms that can rapidly reduce uncertainty can provide significant benefits, for instance, by shortening time and reducing resources required for exploration. Unfortunately, principled algorithms based on rigorous information-theoretic metrics, such as, maximizing Shannon mutual information along the exploration path, are computationally extremely demanding [1]. As a result, the emerging literature considers approximation algorithms, and many practitioners rely on heuristics that often fail to provide any guarantees [2–4].

Dedicated hardware accelerators have been presented for motion planning [5–8], visual inertial odometry [9, 10], deep neural networks [11, 12], object detection for semantic understanding [13, 14] and model predictive control [15]. To the best of our knowledge, custom hardware accelerators for information-theoretic mapping has not yet been considered.

In this paper, we focus on the design of a hardware accelerator for efficiently computing the mutual information between the map and future measurements for a robot equipped with a range measurement sensor. In particular, we propose a novel multi-core hardware architecture capable of high-throughput computation of Shannon mutual information between an occupancy grid map and potential future measurements. A Field Programmable Gate Array (FPGA) implementation of the new hardware architecture leads to significant improvements in computation time and power consumption, for instance, when compared to state-of-the-art implementations for powerful Central Processing Units (CPUs).

Throughout the paper, we show that the throughput of the multi-core hardware is dictated by its *memory architecture* and *data delivery* method. In other words, we find that parallelization alone is not sufficient for high-throughput computation. In addition, it is critical to consider *(i)* memory management, e.g., how data is placed and organized in memory, *(ii)* data delivery, e.g., how data is accessed and delivered to parallel cores, so that throughput scales well with increasing parallelization. In fact, we argue that the effective co-design of computing hardware and algorithms for robotics applications will be enabled by novel methods in *data flow on chip*, for instance, rather than counting the number of operations or amount of memory required that has been essential to developing robotics algorithms for CPUs.

Occupancy grid map [16] is the standard probabilistic representation for a 2D environment and serves as a building block for many well-known mapping system, including the frontier-exploration algorithm [17] and a Shannon mutual-information-based mapping algorithm [1, 18–21]. Aside from Shannon mutual information, alternative information metrics such as Cauchy-Schwarz Quadratic Mutual Information (CSQMI) [22, 23] also proves to be helpful and efficient. Although the computation cores of the hardware presented in this paper is based on the Fast Shannon Mutual Information (FSMI) algorithm [24], the memory architecture and data delivery method to these cores naturally generalizes to potential hardware implementations of other mutual information algorithms, such as the CSQMI.

Thus, the main contribution of this paper is a multi-core hardware architecture that contains a novel memory management and data delivery method for feeding data into multiple high-throughput FSMI computation cores which split mutual information computation across range sensing measurements. The proposed memory management method stores the occupancy grid map into multiple memory banks under a special access pattern, and the proposed data delivery method involves a fair arbiter that quickly manages memory requests from all

computation cores and resolves memory access conflicts.

We implement the proposed architecture on an FPGA clocked at 62.5 MHz and demonstrate that the resulting system is able to compute the mutual information metric for a 20m by 20m map at 0.1m resolution (i.e., 200-by-200 grid map) in near real time, at 2 Hz, resulting in two orders of magnitude improvement in computation time, while lowering the power consumption by an order magnitude to under 2 Watts, when compared to an implementation for an Intel Xeon CPU, which consumes more than 20 Watts extra when running the implementation.

This paper is organized as follows. Section II briefly summarizes FSMI algorithm. Section III is devoted to the description of the proposed architecture. Section IV discusses the results of an FPGA implementation. Section V concludes the paper with a summary and remarks.

II. Preliminaries

An information theoretic mapping problem involves the construction of an occupancy grid map, represented by a vector of independent occupancy cells, each with an occupancy value o_i that represents the probability for the cell to be occupied. As a convention, no prior information is assumed on o_i; therefore, initially for all cells $o_i = 0.5$. The odds-ratio of a cell relates to its occupancy value by $r_i = o_i/(1-o_i) = o_i/\bar{o_i}$.

We use a Bayesian filter to update the map from the range measurement:

$$r_i^{t+1} = r_i^t \delta^i(z), \tag{1}$$

where t is the time of the scan, r_i^t is the odds-ratio of the occupancy value for the i-th cell before the scan, r_i^{t+1} is the odds-ratio after the scan, z is the range measurement, $\delta^i(z)$ is the standard inverse sensing model [1]:

$$\delta^i(z) = \begin{cases} \delta_{emp} & \text{The beam indicates the cell to be empty} \\ \delta_{occ} & \text{The beam indicates the cell to be occupied} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\delta_{emp} < 1$ and $\delta_{occ} > 1$ are hyper-parameters fixed throughout an exploration trial.

In information-theoretic mapping problems, we aim to select a path that maximizes an information-theoretic measure, which typically leads to maximizing the amount of information collected along the path. A prominent metric is the mutual information between the current occupancy grid map and future measurements. Below we summarize the computation of the mutual information metric for a robot equipped with a range measurement sensor. The sensor has multiple beams, each of which measures distance in a different direction. The computation utilizes the Fast Shannon Mutual Information (FSMI) algorithm [24].

Suppose a sensor beam intersects with n occupancy grid cells, namely $M = [M_1, \ldots, M_n]$, where each cell M_i has an occupancy value o_i. Suppose also that the corresponding range measurement is Z. Then, the Shannon mutual information (MI) between the sensor beam and the occupancy grid map, which we denote by $I(M; Z)$, can be computed in three steps:

1) $P(e_j)$: The probability of the j-th cell being the first non-empty cell on the beam: For all $j \in \{1, 2, \ldots, n\}$,
$$P(e_j) = \prod_{i<j} \bar{o_i} o_j. \tag{3}$$
2) C_k: The mutual information contribution of a beam that hits cell k: For all for all $k \in \{1, 2, \ldots, n\}$,
$$C_k = \sum_{i<k} f(\delta_{emp}, r_i) + f(\delta_{occ}, r_k), \tag{4}$$
where $f(\delta, r)$ is the contribution of mutual information from of a depth measurement to a cell indicating whether the cell is empty or occupied:
$$f(\delta, r) = \log\left(\frac{r+1}{r+\delta^{-1}}\right) - \frac{\log \delta}{r\delta + 1}. \tag{5}$$
3) $I(M; Z)$: The approximate Shannon mutual information based on $P(e_j)$ and C_k:
$$I(M; Z) = \sum_{j=1}^{n} \sum_{k=j-\Delta}^{j+\Delta} P(e_j) C_k G_{|k-j|}, \tag{6}$$

where $G_{|k-j|}$ is the area under the sensor's Gaussian noise curve in the k-th cell if j-th cell is the first non-empty cell hit by the sensor beam, and n is the number of occupancy cells that intersect with the sensor beam. The hyper-parameter Δ is the width of the Gaussian truncation, and is typically as small as five[1] [1, 22, 24].

In the sequel, we use MI as a shorthand for the Shannon mutual information $I(M; Z)$.

III. Hardware Architecture

A. Hardware Architecture Overview

This paper presents a novel hardware architecture for FSMI computation shown in Fig. 1. The input to this architecture is a scan location and its associated sensor beams. The output of this architecture is the Shannon MI for the requested scan location in 32-bit floating point precision. This architecture consists of 16 parallel FSMI computation cores that can concurrently compute MI for a set of 16 distinct sensor beams. The occupancy grid map (stored in a specialized memory architecture) contains 8-bit quantized occupancy values b_i, which can be used to generate o_i, $\bar{o_i}$, $f(\delta_{emp}, r_i)$ and $f(\delta_{occ}, r_i)$ used by FSMI computation cores via look up tables. The hardware also contains three data delivery modules (memory request generator, arbiter and workload balance controller) to efficiently delivery these quantized occupancy values b_i from the shared memory to all the FSMI cores. The design goals to enable high-throughput hardware architecture are

1) Fully utilize all the FSMI cores and avoid idle time.
2) Reduce number of cycles required by each FSMI core to compute MI.
3) Increase the clock frequency of the entire system by minimizing the worst case delay between two clocked registers (referred to as the *critical path delay*).

[1]In a typical occupancy grid map, the resolution of the cell is set to match the sensor noise level; hence the Gaussian truncation width Δ should only span a small number of cells (i.e., $\Delta = 5$ works well).

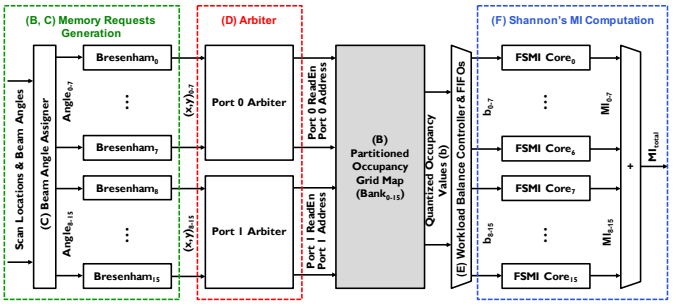
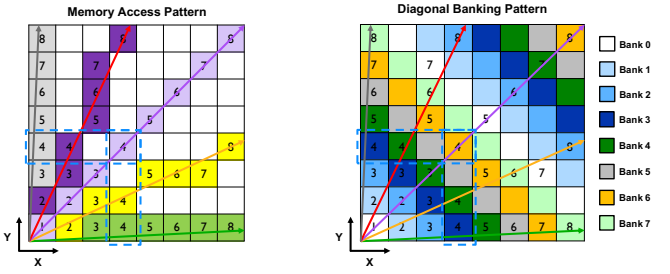


Fig. 1: Overview of the top-level architecture. The Section III's subsection letter that corresponds to each module is annotated before the module name.



(a) Memory access pattern of Bresenham algorithm for multiple beams at every cycle

(b) Diagonal partitioning of the occupancy grid map

Fig. 2: Diagonal partitioned occupancy grid map minimizes memory read conflicts at each cycle, indexed by the numbers.

B. Partitioned Occupancy Grid Map

The design of the specialized memory architecture for storing the occupancy grid map has three target requirements: (1) sufficient memory bandwidth to keep cores busy, (2) minimize read conflicts among cores, and (3) minimize resource utilization.

FPGA memories are composed of blocks of standard random access memory (BRAM) which contains only two read ports (dual-port), meaning that *only two independent reads* (i.e., two reads from different addresses) can be performed concurrently (i.e., in the same clock cycle). Therefore, if the entire occupancy grid map is stored in a single memory bank, a maximum of two FSMI cores can operate in parallel at full utilization because each core requires one independent memory read from the occupancy grid map every cycle.

Enabling 16 FSMI cores to operate at full utilization would require that we replicate the memory bank that stores the entire occupancy grid map by 8 times so that each core has its own read port, and each port can access any occupancy value in the entire map, thus avoiding any read conflicts across cores during every cycle. However, this is very wasteful in terms of the memory storage resources on the FPGA. In fact, this approach does not fit our target mid-tier FPGA platform.

Instead, we use another approach called *memory banking*, which involves partitioning the occupancy grid map into distinct parts and storing each part in a separate memory bank. In this work, we partition the occupancy grid map into 16 banks. The benefit of banking is that multiple cores can operate in parallel at full utilization if they are accessing different parts of the map that are stored in different banks. Due to the utilization of dual-port memory banks, only two cores can access the same bank at the same time. Thus, if more than two cores want to access different addresses of the same bank, a read conflict will occur and some cores will have to wait for their turn and idle (i.e., operate below full utilization). Therefore, one of the key challenges with the memory banking technique is the allocation of different regions of the occupancy grid map to different memory banks so that read conflicts among multiple FSMI cores are minimized during every cycle.

Since each FSMI core has an associated Bresenham module for ray-casting each sensor beam [25], the memory allocation of the occupancy grid map can be optimized based on the pattern of memory access requests generated by the Bresenham module for each FSMI core (green box in Fig. 1). The Bresenham algorithm has a special property that the output coordinate along the major axis of the beam angle always increments by exactly one cell every iteration (execution) of the algorithm as shown in Fig. 2(a). Using the first quadrant as an example, when cores are concurrently processing multiple sensor beams in the same quadrant, these cores will read cells either in the same column (when the beam is between 0 to 45 degrees) or same row (when the beam is between 45 to 90 degrees) of the occupancy grid map. This means that vertically neighboring cells in the same column (when the beam is between 0 to 45 degrees), and horizontally neighboring cells in the same row (when the beam is between 45 to 90 degrees) of the map should be stored in different memory banks. As a result, we propose to partition the occupancy grid map to different banks using a diagonal pattern as shown in Fig. 2(b).

Finally, each memory port can only service one address per clock cycle. To further increase the amount of data that the cores can read in parallel, we pack 2×2 occupancy cells into the same address which increases the amount of occupancy cells read per cycle by $4\times$. However, we do not pack more than this. Too many occupancy cells per address can result in wasted reads as there is no guarantee that all occupancy cells in the address will be used by the cores. An example of 2×2 packing for an occupancy grid map of 8×8 is shown in Fig. 3.

Using the technique of diagonal partitioning and multiple cell packing, we partition an occupancy grid map of size 512×512 into 16 dual-port memory banks with 2×2 packing in our hardware accelerator. The specialized memory architecture for storing occupancy grid map can support the concurrent and continuous operation of 16 FSMI cores.

C. Beam Angle Assignment

Another key design decision is to determine the order with which to assign the beams to the Bresenham ray-casting modules associated with each FSMI core for memory request generation (Fig. 1). We observe that if neighboring beams intersect the same cell at the same time, their corresponding cores can issue and share a single memory read of the

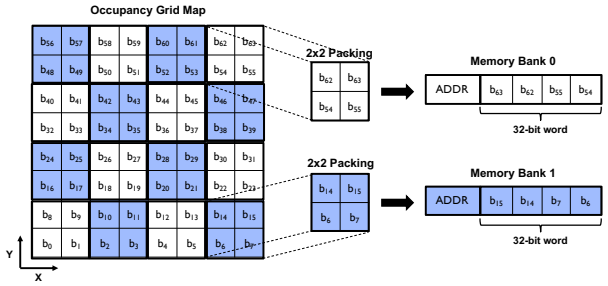
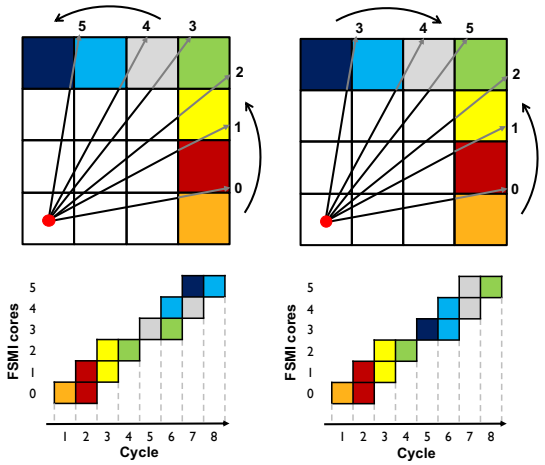


Fig. 3: An example of 2×2 occupancy cell packing for an 8×8 occupancy grid map partitioned into two memory banks.



(a) Naïve assignment (b) Proposed assignment

Fig. 4: Comparison of memory access pattern between two beam angle assignment strategies. Note for beams 3 to 5, the proposed strategy shares the data read from memory at cycle 6 to 7.

cell so that the total number of memory reads are reduced to save energy. Therefore rather than consistently assigning the beams to be processed by each core in a clockwise or counterclockwise manner, we assign the beams to go from major axes (i.e., 0, 90, 180, 270 degrees) to diagonal axes (i.e., 45, 135, 225, 315 degrees). Since each core processes stream of occupancy value starting from the scan location, this beam angle assignment order increases the number of shared reads at every cycle among neighboring beams as shown in Fig. 4.

D. Fast and Fair Memory Arbiter

While the specialized memory architecture for storing the occupancy grid map in Section III-B minimizes the memory read request conflicts from all FSMI cores, the memory arbiter is required to identify and resolve these read conflicts. Thus, the arbiter has two key requirements:

- **Complete in a single clock cycle:** The arbiter must complete all collision checking in one clock cycle. This constraint ensures that the arbiter does not introduce additional delay for memory access, which would reduce the memory bandwidth from the occupancy grid map to the FSMI cores.
- **Fairness:** When a collision arises, the arbiter should not be biased towards memory request from one core over another so that a constant data delivery rate from the memory to all cores is maintained. This ensures that all FSMI cores are equally active.

The complexity and, consequently, the delay of the arbiter grows with the number of cores. Therefore rather than using a single arbiter to service memory requests from 16 cores using 16 dual-port memory banks, we separate 16 cores into two groups of 8, where each group has its own arbiter. As a result, each arbiter only need to service memory requests from its group of 8 cores using a single memory port from every bank. This is illustrated in the red box of Fig. 1.

Each arbiter uses a round robin scheme that keeps track of a priority pointer to ensure fairness in processing read requests among all cores so that they are equally active and the overall throughput can be improved. There are three steps to this approach as shown in Fig. 5:

- *Step 1*: Identify read conflicts among all cores (i.e., cores that want to read different addresses of the same bank).
- *Step 2*: Starting from the priority pointer, service read requests from all cores up to the first read conflict. The priority pointer is referred to as the beginning of the service window, while the core before the one with the first conflict is referred to as the end of the window.
- *Step 3*: Move the priority pointer to the location of the first conflict, and get the new read requests from the core that were serviced in Step 2.

These three steps are repeated until the MI computation is complete. One of the main challenges of the arbiter design is to complete all three steps in one clock cycle. In particular, the main computation bottleneck is to determine the size of the service window for a given position of the priority pointer.

In our design, the length of the critical path in the arbiter determines the maximum frequency of the clock. Hence, to maximize the clock frequency, we propose a tree structured circuit with a shorter critical path to determine the start and end positions of the service window for every bank in each arbiter. Fig. 6 illustrates the design. The location of the window is first determined for each bank. Starting at the leaf nodes, the arbiter computes the start and end location of the window for subsets of the cores in each bank and the windows are merged at the next level. The overall window, which is the intersection of the service windows across all 16 banks, is computed using another tree structure. By utilizing two tree-structured circuits, the critical path delay scales logarithmically in $O(log(B) + log(T))$, where B is the number of memory banks and T is the number of cores that the arbiter has to service. In our design, $B = T = 16$.

E. Workload Balance Controller

Since number of occupancy cells varies per beam (depending on the angle), the number of cycles required to process each beam varies for each FSMI core. Using the beam angle assignment strategy described in Section III-C, the core with

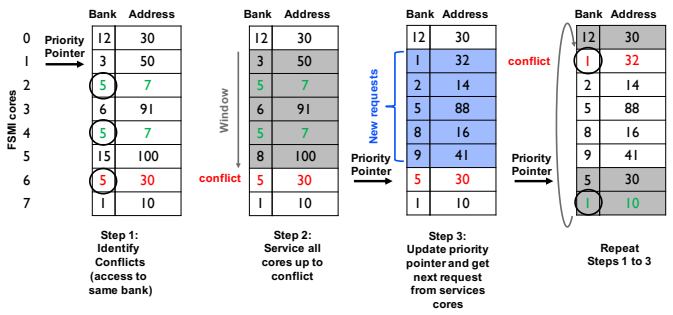


Fig. 5: Key steps in arbiter.

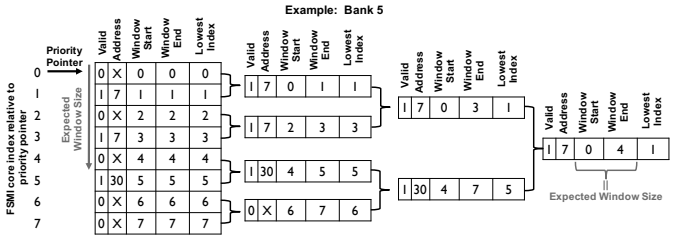


Fig. 6: Tree structure to determine start and end of service window in arbiter for each bank.

higher index (i.e., closer to the diagonal) tends to process less occupancy values because its associated sensor beam (of fixed length) crosses the occupancy cells in the map diagonally. However, achieving high throughput requires that the workloads across all FSMI cores are balanced as the overall speed up depends on the core that does the most work (i.e., takes the most clock cycles to complete). Since the hardware contains 16 cores, FSMI for a batch of 16 beams that intersect different number of cells are calculated every round. Thus, a workload balance controller is added before the FSMI cores so that for every round of 16 beams, the controller flips the order of beam allocation and Bresenham module association for each FSMI core as shown in Fig. 7.

F. FSMI Computation Cores

In the previous section, we present the memory and data delivery architecture that improve the utilization of the 16 parallel FSMI cores. In this section, we will illustrate how to make the cores themselves fast by performing parallel

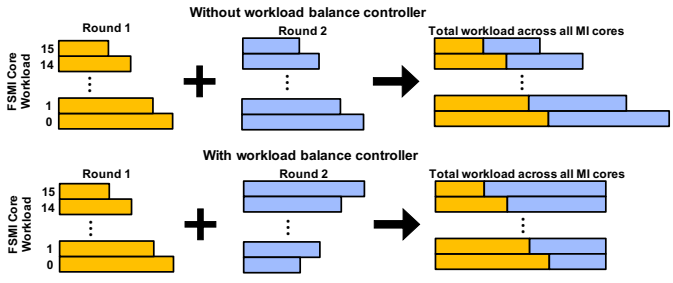
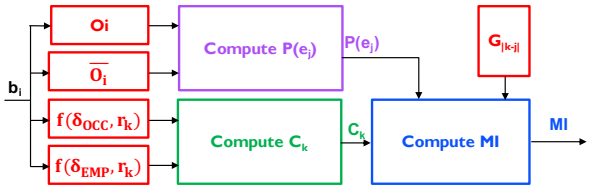
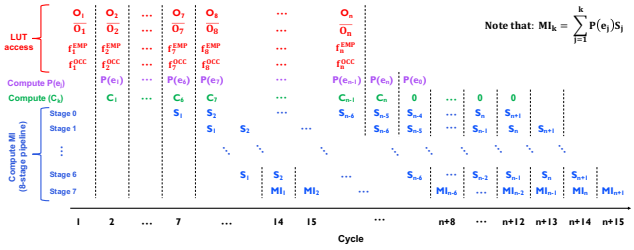


Fig. 7: Workload balance controller flips beam assigned to each FSMI core to balance workload for higher throughput.



(a) Connections among the main modules of the FSMI core



(b) Timing diagram of interactions among main modules. (LUT = look up table)

Fig. 8: Hardware architecture overview of the FSMI core.

operations within each core, improving the utilization of its compute units (i.e., multipliers and adders), and reducing the critical path delay (increasing clock frequency) via pipelining.

Each FSMI core is used to evaluate the Shannon MI between a single sensor beam and the occupancy grid map. From Section II, the computation of MI is defined by Eq. (6), which requires the computation of $P(e_j)$ (defined by Eq. (3)) and C_k (defined by Eq. (4)). The proposed FSMI core, shown in Fig. 8(a), performs these computations in parallel, shown in Fig. 8(b), such that we can complete the MI computation within $n + 15$ clock cycles, where n is the beam length. Look up tables (LUT) are accessed for translating the 8-bit quantized output of the occupancy grid map b_i to pre-loaded floating-point parameters of o_i, $\bar{o}_i$, $f(\delta_{emp}, r_i)$ and $f(\delta_{occ}, r_i)$ needed for computing $P(e_j)$ and C_k (red in Fig. 8(b)). All computations in the FSMI core are performed in 32-bit floating point precision.

The micro-architecture for computing $P(e_j)$ is shown in Fig. 9(a) with its associated timing diagram shown in Fig. 9(b). It consists of two 32-bit floating point multipliers operating in parallel; one is used to compute the cumulative multiplication of $\bar{o}_i$ to generate E_{i-1}, while the other is used to update the value of $P(e_j)$ every clock cycle by computing the product of E_{i-1} and o_i. Muxes are inserted in the feedback loop and the output to handle initialization (sw_fb) and termination (sw_o) of the module. The critical path is highlighted in Fig. 9(a), and it is dictated by a multiplication and a mux.

The micro-architecture for computing C_k is shown in Fig. 10(a) with its associated timing diagram shown in Fig. 10(b). The module is almost identical to $P(e_j)$ except that the multipliers are replaced with adders.

The final computation of MI uses the output of $P(e_j)$ and C_k computations. From Section II, the width of the Gaussian noise truncation $\Delta = 5$. To increase the throughput, the double summation in Eq. (6) is broken into two operations:

$$S_j = \sum_{k=j-\Delta}^{j+\Delta} C_k G_{|k-j|} \quad (7) \qquad MI = \sum_{j=1}^{n} P(e_j) S_j \quad (8)$$

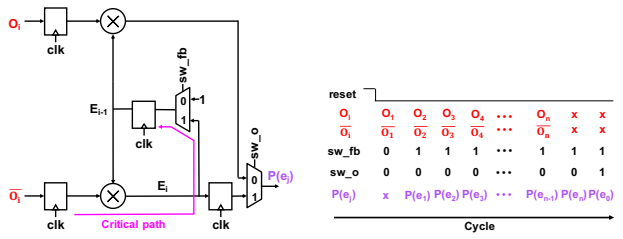
The operations in Eq. (7) is illustrated in Fig. 11(a). The inputs needed for calculating S_j is a shifted stream of 11 consecutive C_k within the range of the truncated Gaussian noise curve. To compute S_j, each C_k is first multiplied by a corresponding area under the Gaussian noise distribution ($G_{|k-j|}$) and then summed together. Since the Gaussian noise distribution is symmetric, hardware optimization can be performed by first summing the corresponding C_k under the symmetric regions of the Gaussian curve and then multiply the summation by $G_{|k-j|}$, reducing the required number of floating-point multipliers by almost half, as shown in Fig. 11(b).

Fig. 11(b) is directly mapped to the hardware shown in Fig. 12. The MI micro-architecture uses 8 pipeline stages in order to reduce the critical path delay and simultaneously process different data in each stage. In stage 0, a sliding window of consecutive C_k is created with the shift registers. This requires one C_k every clock cycle, matching the throughput of the hardware architecture for C_k computation. Stage 1 and 2 illustrate the optimization described in Fig. 11(b). The summation in Eq. (7) is performed with an adder tree in stages 3 to 5 so that S_j is computed at the output of stage 5. Stage 6 and 7 perform the accumulation in Eq. (8) over $n+1$ cycles.

IV. Results

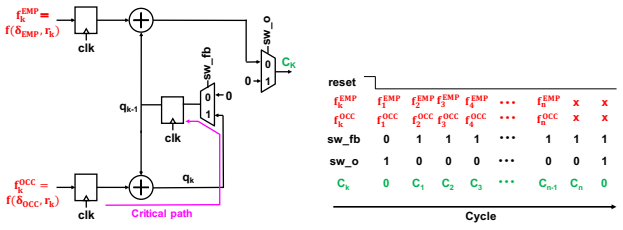
A. Hardware Accuracy

The FPGA implementation was verified against a reference MATLAB implementation of FSMI [24] by computing the MI for every potential scan location on a 512×512 occupancy grid map shown in Fig. 14(a). Fig. 14(b) shows the MI map computed by FPGA. The MI value computed by FPGA at every scan location has a relative error of less than $4 \times 10^{-5}\%$ compared with the one generated by the MATLAB reference. We believe this negligible error is due to the difference in the order of the operations for the floating point computations.



(a) Micro-architecture (b) Timing diagram

Fig. 9: Compute module for $P(e_j)$



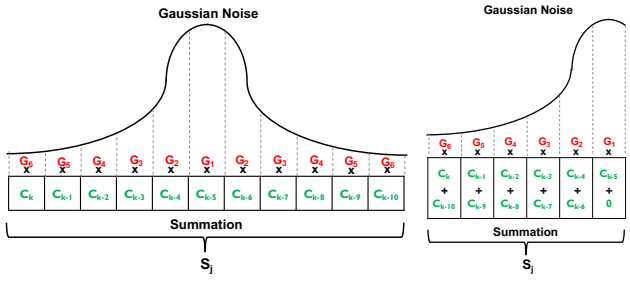
(a) Micro-architecture (b) Timing diagram

Fig. 10: Compute module for C_k

B. Computational Speed

1) Performance of Single FSMI Core: We compare the speed of processing a single beam on the proposed single FSMI core on running on a Xilinx Zynq-7000(XC7Z045) FPGA to a Intel Xeon E5-4627 v2 CPU core running a reference C++ implementation of FSMI with all optimizations discussed in [24]. The comparison is performed across different beam lengths (n). For each beam length, both implementations compute MI on randomly generated occupancy vectors, and we repeat this procedure 10000 times for each beam length to get more stable timing results. Fig. 15(a) shows that a single FSMI core on operating at 62.5MHz on an FPGA is more than an order of magnitude faster than a Xeon CPU core operating at 3.4GHz across all tested beam lengths.

2) Performance of 16 FSMI cores with proposed memory and data delivery architecture: As previously discussed, if the memory has infinite bandwidth, 16 FSMI cores running in parallel will always lead to 16 times acceleration compared against a single core. We use the latency of this ideal case as the reference to measure the quality of the proposed memory and data delivery architecture. Recall that the hardware accelerator uses 16 dual-port memory banks to partition the occupancy grid map in a diagonal fashion. Each entry of the memory stores 2×2 occupancy values. In our simulation, each MI is computed by summing the MIs for all sensor beams with a beam length of 200 in various scan resolution settings (1° to 22.5° between neighboring beams in a 360° scan). Fig. 15(b) shows the speed of computing MI on the hardware accelerator compared against the ideal reference that assumes infinite bandwidth. Across all scan resolutions with a fixed beam length of 200, the proposed design is less than 6% slower than the ideal reference. This validates the effectiveness of the proposed memory and data delivery architecture.



(a) Direct computation (b) Exploit symmetry

Fig. 11: Illustration of computation for S_j in Eq. (7)

C. Resource Usage and Power Consumption

Table I shows the resource utilization of the proposed hardware accelerator shown in Fig. 1 on the FPGA with 16 FSMI cores and 16 dual-port memory banks. On an FPGA, arithmetic operations are either performed using digital signal processing (DSP) hardware, which supports fixed-point multiplication and addition, or Look-Up Tables (LUT) that emulate logic gates (e.g., AND/OR); floating point operation uses both DSPs and LUTs. On-chip memory includes Block RAM (BRAM), LUT RAM and LUT registers (Reg). The entire FPGA implementation consumes around 2 Watts which is over an order of magnitude less than the Xeon CPU core which consumes 22 Watts when running FSMI. The breakdown shows that the 16 FSMI cores dominates the resource utilization as well as the power consumption. Thus, we drastically increased the memory bandwidth to all FSMI cores with only 22% logic and 8% power overhead (first four rows of Table I) compared with total amount of logic and power used by 16 FSMI cores (fifth row of Table I) while maintaining the same storage size of the occupancy grid map.

D. Impact of Hardware Design Parameters

We evaluate the impact of our design choices on throughput and resource utilization by exploring the design space.

1) Number of memory banks: For a fixed memory size, increasing the number of banks increases the memory bandwidth as well as the logic needed for the arbiter (larger tree structure) and the workload balance controller (manage data from more banks). However, for a fixed number of FSMI cores, increasing the memory bandwidth beyond a certain point has diminishing returns on the overall throughput improvement of the hardware system. Therefore, we balanced the trade-offs between hardware resource utilization and overall throughput when choosing the number of banks for a fixed number of FSMI cores so that the system fit on our target FPGA platform.

2) Number of b_i per memory address (packing factor L): Another design choice is the packing factor L that dictates the number of quantized occupancy values b_i ($L \times L$) packed in each memory address of every bank. While increasing L also increases memory bandwidth to FSMI cores, it introduces hardware complexity and resource overhead for the Bresenham modules (need to generate L requests every cycle), the arbiter (need to service L requests from every core every cycle) and the workload balance controller (need to control up to L occupancy values to each MI core every cycle). In addition, a larger L increases the energy needed for reading each useful b_i because at least $L(L-1)$ cells in every $L \times L$ block are not needed (a sensor beam can cross at most L occupancy cells in a $L \times L$ block under the access pattern ray-casted using the Bresenham algorithm).

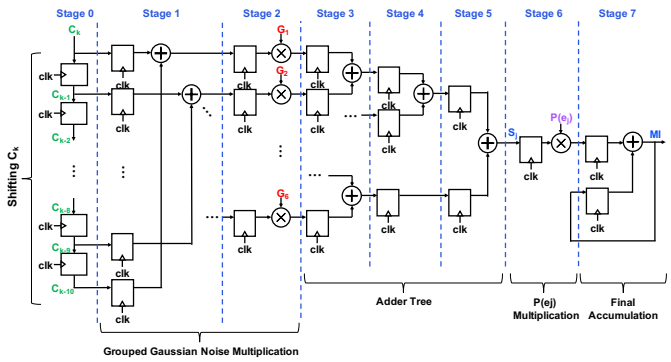


Fig. 12: Micro-architecture to compute MI

Fig. 13 shows several sweeps of hardware performance and design parameters. Fig. 13(a) suggests that packing 2×2 occupancy values ($L = 2$) is better than only packing a single entry ($L = 1$) if the extra hardware complexity and resource usage are tolerable. However, as long as the number of memory banks is as high as 16, increasing L from 2 to 4 leads to more hardware complexity (Fig. 13(b)) with little reduction to overall system latency (Fig. 13(a)). Thus, we limit $L = 2$ and choose the number of banks to be 16 when designing our system.

Finally, Fig. 13(c) shows that increasing the number of cores alone is not sufficient for increasing throughput. The proposed memory and data delivery architecture enables significant increase in throughput which scales well with number of FSMI cores. Table II summarizes the impact of our design decisions for 16 FSMI cores with 16 dual-port memory banks compared with a baseline system that has 1 dual-port memory bank.

E. Impact of Faster MI Evaluation

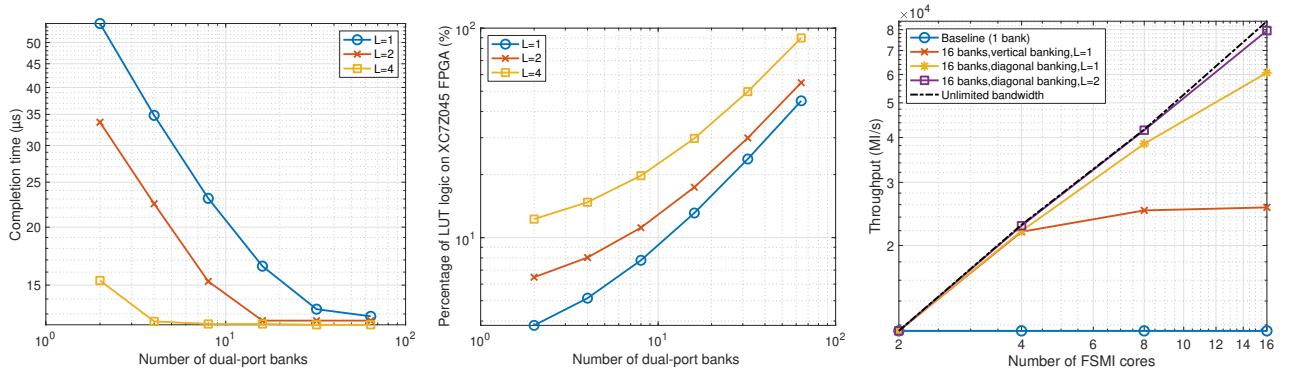
In an information theoretic mapping framework, faster computation of MI enables higher planning frequency. Previous work [26] has demonstrated that up to the mechanical limit of a vehicle, the safe mapping speed of a vehicle scales almost linearly with respect to the planning frequency. Hence, the two orders of magnitude acceleration of the FPGA system compared against a powerful Xeon CPU can potentially lead a vehicle to explore at much faster speed.

TABLE I: Breakdown of resources utilization of different modules on the Xilinx Zynq-7000(XC7Z045) FPGA.

Module	LUT (Logic)	LUT (Reg)	BRAM	LUT (RAM)	DSP	Dynamic Power
Angle Assigner	131	437	0	0	0	0.001W
Bresenham (16x)	10602	3017	0	1408	32	0.042W
Arbiter	25226	2055	0	662	0	0.107W
Occupancy Grid Map	0	0	64	0	0	
FSMI Core (16x)	163141	36856	40	49	288	1.827W
Total	**199101**	**42365**	**104**	**2119**	**320**	**1.977W**
Utilization of FPGA	**91.1%**	**9.7%**	**19.1%**	**3.0%**	**35.6%**	**NA**

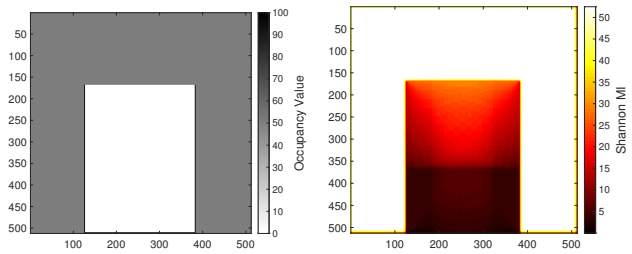
TABLE II: MI computation time for 60 beams of length 200 using 16 FSMI cores with different memory configurations.

	Baseline	Vertical banking & L = 1	Diagonal banking & L = 1	Diagonal banking & L = 2	Unlimited Bandwidth
Latency	$86.53\mu s$	$39.15\mu s$	$16.48\mu s$	$12.58\mu s$	$11.84\mu s$
Speed up	$1\times$	$2.21\times$	$5.25\times$	$6.88\times$	$7.31\times$



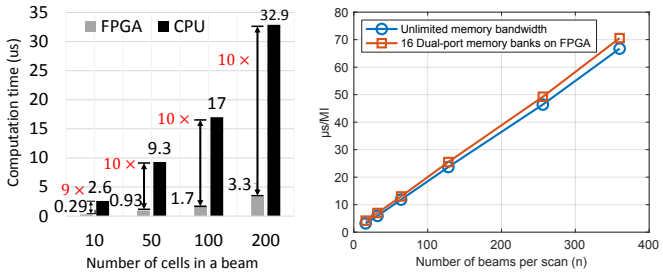
(a) Impact on compute latency (b) Impact on resource usage (c) Impact of memory bandwidth on throughput

Fig. 13: Exploration of hardware design parameters.



(a) Occupancy grid map input (b) MI map from FPGA

Fig. 14: Input occupancy grid map and the MI map generated on FPGA, which has a relative error of less than $4 \times 10^{-5}\%$ compared with one generated by the MATLAB reference.



(a) Computational speed-up compared with Xeon CPU (b) MI computation time for different scan resolutions

Fig. 15: Computation speed evaluation

We also validate the benefit of a faster MI evaluation in a 2D synthetic exploration experiment. We will show that by evaluating more MI at every planning step can lead to faster exploration of a small $20m \times 20m$ environment. The map is $0.1m$ resolution. Hence it uses 200×200 portion of the occupancy grid map, although the hardware is capable of 512×512 occupancy grid map size. Fig. 16 shows the average entropy reduction curve with respect to travel distance for two configurations. In one configuration, the number of candidate scan locations for FSMI is 25 times larger than the other configuration. Five different trials are repeated for each configuration. From Fig. 16, we measure that 25 times more

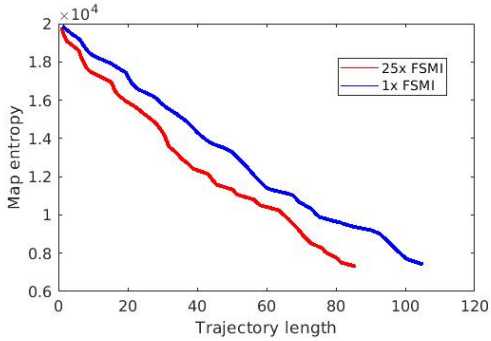


Fig. 16: Being able to evaluate FSMI at 25 times more locations lead to shorter exploration trajectories.

FSMI evaluation leads to around 19% shorter exploration paths for the entropy to drop by 80%. This benefit is significant for this 2D synthetic exploration experiment.

V. Conclusions

In this paper, we presented a new hardware architecture for low-latency, high-throughput computation of Shannon mutual information. We argued that the key challenge is to design the memory management and data delivery components that ensure maximal utilization of parallel computation cores. We have shown that the proposed hardware architecture achieves this goal by containing a diagonally partitioned occupancy grid map that minimizes the cores' memory access conflicts with a fair arbiter that quickly identifies and resolves these conflicts. We implemented the proposed memory and data delivery architecture with 16 high-throughput FSMI computation cores on an FPGA platform. The entire system is able to compute the mutual information for a 20m by 20m map at 0.1m resolution (i.e., 200-by-200 occupancy grid map) in near real time in 2Hz, while consuming less than 2 Watts. An implementation of the same FSMI algorithm for an Intel Xeon CPU runs two orders of magnitude slower, while consuming more than 20 Watts of power in addition to the idle power of the CPU.

Acknowledgements. This work was partially funded by the AFOSR YIP FA9550-16-1-0228, by the NSF CAREER 1350685 and NSF CPS 1837212.

REFERENCES

[1] Brian J Julian, Sertac Karaman, and Daniela Rus. On mutual information-based control of range sensing robots for mapping applications. *The International Journal of Robotics Research*, 33(10):1375–1392, 2014.

[2] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. Coordinated multi-robot exploration. *IEEE Transactions on robotics*, 21(3):376–386, 2005.

[3] Héctor H González-Banos and Jean-Claude Latombe. Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research*, 21(10-11):829–848, 2002.

[4] Dirk Holz, Nicola Basilico, Francesco Amigoni, Sven Behnke, et al. A Comparative Evaluation of Exploration Strategies and Heuristics to Improve Them. In *ECMR*, pages 25–30, 2011.

[5] Nuzhet Atay and Burchan Bayazit. A motion planning processor on reconfigurable hardware. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 125–132, 2006.

[6] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel J Sorin, and George Konidaris. Robot Motion Planning on a Chip. In *Robotics: Science and Systems*, 2016.

[7] Youchang Kim, Dongjoo Shin, Jinsu Lee, and Hoi-Jun Yoo. BRAIN: A Low-Power Deep Search Engine for Autonomous Robots. *IEEE Micro*, 37(5):11–19, 2017.

[8] Size Xiao, Neil Bergmann, and Adam Postula. Parallel RRT architecture design for motion planning. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–4, 2017.

[9] Zhengdong Zhang, Amr AbdulZahir Suleiman, Luca Carlone, Vivienne Sze, and Sertac Karaman. Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach. In *Robotics: Science and Systems*, 2017.

[10] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. Navion: A Fully Integrated Energy-Efficient Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones. In *2018 IEEE Symposium on VLSI Circuits*, pages 133–134, 2018.

[11] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52:127–138, 2016.

[12] Bert Moons and Marian Verhelst. A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets. In *Symp. on VLSI*, 2016.

[13] Amr Suleiman, Zhengdong Zhang, and Vivienne Sze. A 58.6 mW real-time programmable object detector with multi-scale multi-object support using deformable parts model on 1920× 1080 video at 30fps. In *Symp. on VLSI*, 2016.

[14] Kenta Takagi, Kotaro Tanaka, Shintaro Izumi, Hiroshi Kawaguchi, and Masahiko Yoshimoto. A real-time scalable object detection system using low-power hog accelerator vlsi. *Journal of Signal Processing Systems*, 76(3):261–274, 2014.

[15] Jacob Sacks, Divya Mahajan, Richard C Lawson, and Hadi Esmaeilzadeh. RoboX: An end-to-end solution to accelerate autonomous control in robotics. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 479–490, 2018.

[16] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, (6):46–57, 1989.

[17] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151, 1997.

[18] Frederic Bourgault, Alexei A Makarenko, Stefan B Williams, Ben Grocholsky, and Hugh F Durrant-Whyte. Information based adaptive robotic exploration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 540–545. IEEE, 2002.

[19] Gabriel M Hoffmann and Claire J Tomlin. Mobile sensor network control using mutual information methods and particle filters. *IEEE Transactions on Automatic Control*, 55(1):32–47, 2010.

[20] Thomas Kollar and Nicholas Roy. Efficient Optimization of Information-Theoretic Exploration in SLAM. In *AAAI*, volume 8, pages 1369–1375, 2008.

[21] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Robotics: Science and Systems*, volume 2, pages 65–72, 2005.

[22] Benjamin Charrow, Sikang Liu, Vijay Kumar, and Nathan Michael. Information-theoretic mapping using Cauchy-Schwarz Quadratic Mutual Information. In *IEEE International Conference on Robotics and Automation*, 2015.

[23] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping. In *Robotics: Science and Systems*, volume 6, 2015.

[24] Zhengdong Zhang, Trevor Henderson, Vivienne Sze, and Sertac Karaman. FSMI: Fast computation of Shannon Mutual Information for Information Theoretic Mapping. In *IEEE International Conference on Robotics and Automation*, 2019.

[25] Jack Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100–106, 1977.

[26] Erik Nelson and Nathan Michael. Information-theoretic occupancy grid compression for high-speed information-based exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4976–4982, 2015.