SIGMICRO: G: A Mutual Information Accelerator for Autonomous Robot Exploration

Peter Zhi Xuan Li (Advisors: Sertac Karaman, Vivienne Sze)
peterli@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts

ABSTRACT

Exploration problems are fundamental to robotics, arising in various domains, ranging from search and rescue to space exploration. In these domains and beyond, exploration algorithms that allow the robot to rapidly create the map of the unknown environment can reduce the time and energy for the robot to complete its mission. Many effective exploration algorithms rely on the computation of Shannon mutual information (MI) which allow the robot to select the best location to explore in order to gain the most information about the unknown environment. Unfortunately, computing MI metrics is computationally challenging. While the computation of MI can be parallelized and thus computed by many parallel cores, the main challenge that limits throughput is the delivery of data to these cores. As such, in this work we propose a MI hardware accelerator that has a novel memory banking pattern and an arbiter that ensure effective utilization of all MI cores to maximize throughput. In addition, our rigorous analysis of the banking pattern and arbiter introduces a new paradigm for theoretically evaluating hardware design decisions. Finally, the proposed architecture is validated on an FPGA and implemented on an ASIC in a commercial 65nm technology. Our ASIC implementation computes the MI for an entire map from a real-world experiment of $10.05m \times 10.05m$ at $0.05m$ resolution in real time at $11Hz$, which is 83× and 12× faster than the ARM Cortex-A57 CPU and NVIDIA Pascal GPU on the Jetson TX2 board respectively. Furthermore, the ASIC implementation consumes $162mW$, which is 21× lower and 19× lower than the ARM Cortex-A57 CPU and NVIDIA Pascal GPU on the Jetson TX2 board respectively.

1 PROBLEM AND MOTIVATION

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 allow the robot to rapidly create the map of the unknown environment can reduce the time and energy for the robot to complete its mission. Shannon mutual information (MI) at a given location is a measure of how much *new* information of the unknown environment the robot will obtain given what the robot already know from its incomplete understanding of the environment. A typical exploration pipeline is shown in Figure 1 where robot starts with an incomplete map of the environment. At every step, the robot computes the MI across the *entire map*. Then, the robot can select the location with the highest mutual information for exploration in order to gain the most information about the unknown environment [1].

However, on the embedded Central Processing Units (eCPUs) and Graphical Processing Units (eGPUs) typically found on mobile robotic platforms, computing MI using the state-of-the-art Fast Shannon Mutual Information (FSMI) algorithm [2] across the entire map takes more than one second, which is too slow for enabling fast autonomous exploration. As a result, the emerging literature considers approximation techniques, and many practitioners rely on heuristics or computing MI across only a *subset of locations* within the map, which often fail to provide any theoretical guarantees [3–5]. To eliminate the bottleneck associated with the computation of MI across the entire map, we propose a specialized multi-core hardware that accelerates the FSMI algorithm so that the throughput for computing MI can be significantly increased while the power consumption is at a fraction of that of the embedded CPUs and GPUs. In summary, we made the following contributions.

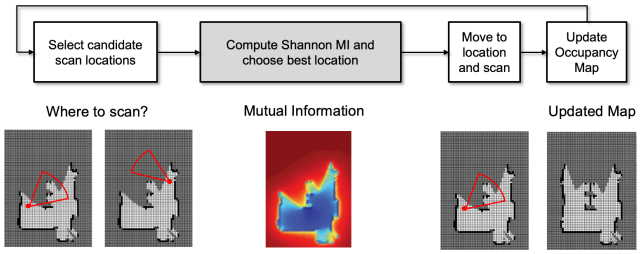


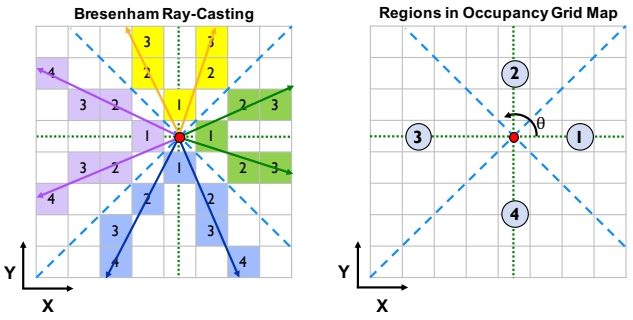
Figure 1: A typical pipeline used for autonomous exploration. To minimize the amount of exploration time and trajectory, the robot computes the mutual information for the entire map in order to choose the location with the highest mutual information to explore. See video in [2].

(1) We propose a novel multi-core hardware architecture with a memory subsystem that efficiently organizes the storage of the occupancy grid map and an arbiter that effectively resolves memory access conflicts among MI cores so that the entire system achieves high throughput.
(2) We provide rigorous analysis of memory subsystem and arbiter in order to justify our design decisions and provide provable performance guarantees.
(3) We thoroughly validated the entire hardware architecture by implementing it using a commercial 65nm ASIC technology.

2 APPROACH AND UNIQUENESS

2.1 Challenges & Proposed Architecture

Occupancy grid map [6] is the probabilistic representation for the 2D environment stored in memory. At any scan location in the occupancy grid map, the robot makes several range measurements which are represented by sensor beams emanating from the scan location (red dot) as shown in Figure 2(a). Computing the MI at the scan location requires the summation of the MI along cells within the



(a) Sequence of occupancy cells along every sensor beam (ray casted by the Bresenham algorithm) that needs to be accessed by the MI core every cycle, marked by numbers.

(b) Labelled regions partitioned by diagonal axes (blue lines) with origin at the scan location (red dot).

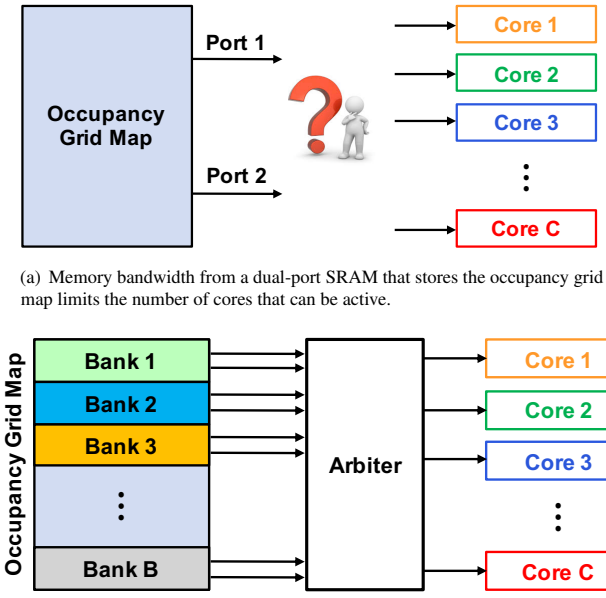
Figure 2: Properties of sensor beams emanating from scan location (red dot) after ray casting using the Bresenham algorithm [7] on an 8×8 occupancy grid map.

occupancy grid map that intersect each sensor beam. To increase throughput, MI computation for cells along several beams can be performed in parallel with multiple cores such that each core computes MI for one beam. However, the system throughput becomes memory bounded when the occupancy grid map is stored in a single dual-port SRAM because each core requires one memory access from the map every cycle to remain active as shown in Figure 3(a). To increase memory bandwidth to the cores, we propose a hardware architecture shown in Figure 3(b). The proposed architecture contains a memory subsystem that stores the occupancy grid map in multiple smaller dual-port banks so that there are more read ports for the cores to access the map. Recall that *memory access collision* occurs when more than two cores access distinct locations of the map stored within the same bank. Thus, in Section 2.2, we propose a *banking pattern* that assigns the occupancy grid map to the banks such that the number of memory read conflicts among cores is minimized. If the memory access collision do occur, the conflicting memory access requests should be resolved quickly. Thus, in Section 2.3, we propose a priority arbiter that quickly resolves potential memory access conflicts. In addition, we rigorously characterize the area (occupied by digital logic gates) and critical path of the arbiter to ensure that they scale well with increasing number of cores.

2.2 Banking Pattern

We assign the occupancy grid map into several dual-port memory banks to increase number of memory read ports / memory bandwidth to the MI compute cores as shown in Figure 3(b). Since ray casting for the sensor beams is performed using the Bresenham algorithm [7], every core concurrently accesses the same column or row of the occupancy grid map at every cycle as shown in Figure 2(a). To minimize the number of memory access collisions among the cores, the banking pattern should assign the cells in every column and row of the occupancy grid map into different banks.

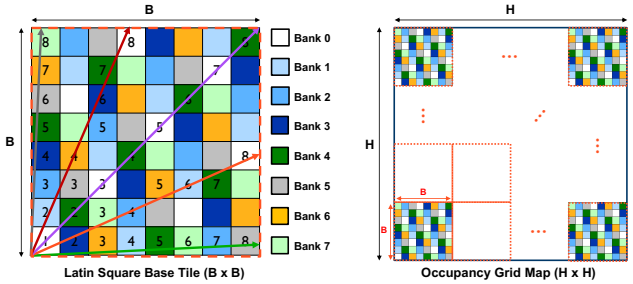
Let B denote the number of banks in the memory subsystem that stores the occupancy grid map. A Latin square pattern of size $B \times B$



(a) Memory bandwidth from a dual-port SRAM that stores the occupancy grid map limits the number of cores that can be active.

(b) Proposed multi-core hardware architecture that provides sufficient memory bandwidth so that the computation cores are active.

Figure 3: Proposed architecture resolves memory bandwidth limitation from the memory to the computation cores.

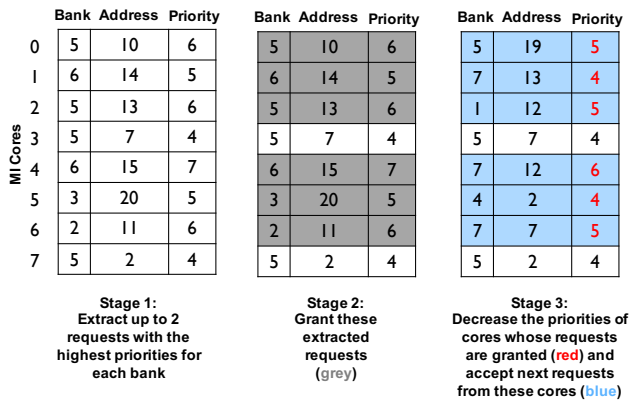


(a) An example of Latin-square base tile with $B = 8$ banks. Cells in every row and column of the square are assigned to distinct banks.

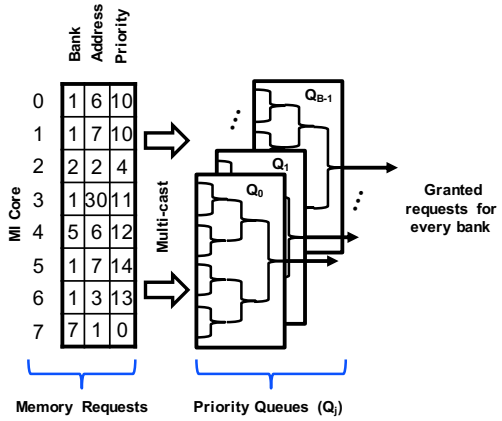
(b) Construction of the Latin-square banking pattern by replicating the Latin-square base tile.

Figure 4: Latin-square banking pattern when the total number of banks is $B = 8$.

contains distinct numbers in each row and column of the square [8]. To minimize the number of memory access conflicts, we assign the occupancy grid map into B banks using the Latin-square banking pattern as shown in Figure 4, which can be constructed by replicating the Latin-square base tile of size $B \times B$ in Figure 4(a) across the entire dimension of the occupancy grid map in Figure 4(b). We rigorously proved that the Latin-square banking pattern is near-optimal in the sense that it allows the cores to achieve the minimum number of memory access conflicts (which scales in $O(1/B^2)$) as the number of sensor beams emanating from the same scan location and lying within the same region (as defined in Figure. 2(b)) goes to infinity.



(a) Operations of the the priority arbiter during every cycle.



(b) Overview of the hardware architecture for the priority arbiter. A tree-structured priority queue for every bank is used identify up to two requests with the highest priorities.

Figure 5: Operation of the priority arbiter and its hardware architecture.

2.3 Priority Arbiter

To ensure most cores are active, the arbiter should quickly resolve memory access conflicts. Determining the optimal arbitration strategy across all cores can be formulated as a Job Shop Scheduling Problem, which is NP-hard for more than two cores [9]. Thus, we propose a greedy arbitration heuristic, called the *priority arbiter*, that makes a set of arbitration decisions as the memory access requests are generated.

The operation of the priority arbiter is shown in Figure 5(a). Each core has its own priority counter, which is initialized to its total number of memory access requests at the beginning of MI computation. At every cycle, the arbiter receives one memory access request (i.e., bank and its address) from every core. Since each bank has two ports, it can service up to two requests. In order to minimize the number of remaining requests for every core, the priority arbiter grants up to two requests with the highest priorities for every bank. For the cores whose requests are granted, their next memory access requests are sent to the arbiter and their priority counters decrease by one. For the cores whose requests are not granted, their memory access requests and priority counters remain unchanged. The priority arbiter continues to operate until every request is serviced (i.e., priority counter for every core becomes zero).

An overview of the hardware architecture for the priority arbiter is shown in Figure 5(b). The memory request and priority counter for every core is broadcasted to several priority queues (one for each bank) which identifies up to two requests with the highest priorities. The priority queues are implemented using a tree-structured architecture such that its critical path scales in $O(\log(C))$ and its area scales in $O(C)$, where C is the number of cores. Let B denote the number of banks used to store the occupancy grid map. Since the priority arbiter is constructed using B priority queues (one for each bank) that operate concurrently, the critical path and area of the priority arbiter scales in $O(\log(C))$ and $O(BC)$, respectively. Since the critical path the arbiter increases sub-linearly with the number of cores C, the latency of the priority arbiter scales well with the number of cores. To minimize the area of the arbiter given the number of cores C, we need to minimize the number of banks B used to store the occupancy grid map while ensuring that these cores receive sufficient memory bandwidth to remain active.

In our work, we rigorously proved that the arbiter greedily grants the maximum number of requests every cycle without introducing any deadlock. When the cores access the occupancy grid map that is assigned to the banks using the Latin-square banking pattern, we derived a guaranteed lower bound on the average percentage of memory requests granted every cycle by the priority arbiter, which is referred as the *average core utilization*. Since our proposed architecture contains 16 cores, we plotted the average utilization of these cores vs. the number of banks B used to store the occupancy grid map (via the Latin-square banking pattern) as shown in Figure 6. As the number of banks increases, the priority arbiter needs to resolve less memory access conflicts, which leads to an increasing average core utilization towards 100% (red line). In addition, the average core utilization achieved by the priority arbiter lies above the theoretical lower bound (blue line) and slightly below the state-of-the-art arbitration strategy determined using the IBM CPLEX CP optimizer [10] (green line). Thus, the performance of the priority arbiter is comparable with the state-of-the-art arbitration strategy. In order to minimize the area of the priority arbiter while ensuring that more than 90% of the cores are utilized (i.e., more than 90% of the memory requests are granted) every cycle, we need to use at least 16 banks (i.e., $B = 16$) to store the occupancy grid map via the Latin-square banking pattern.

3 RESULTS AND CONTRIBUTIONS

We implemented the proposed architecture with 16 MI cores and partitioned the storage of a occupancy grid map with a maximum size of 512×512 into 16 banks using the Latin-square banking pattern. We validated the architecture using Xilinx Zynq-7000(XC7Z045) FPGA and performed post-layout simulations using a commercial 65nm technology. In the following sections, we present the accuracy, throughput, latency and power consumption of the proposed architecture.

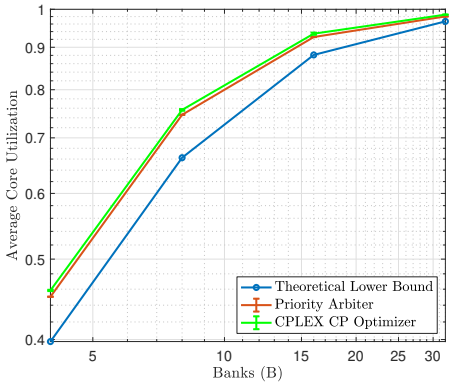
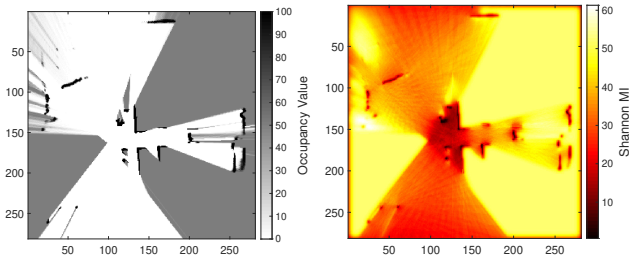


Figure 6: Average core utilization of the priority arbiter vs. number of banks B in the Latin-square banking pattern when the proposed architecture contains 16 cores.



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

Figure 7: Occupancy grid map and its corresponding MI map computed in half-precision floating point arithmetic on an FPGA.

3.1 Accuracy

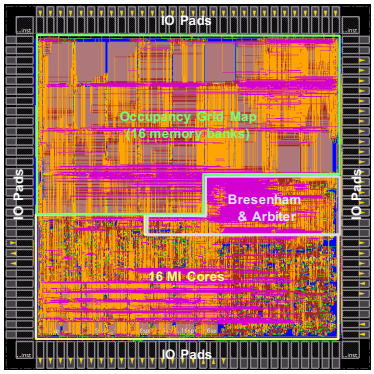
We validated the MI computation for an occupancy grid map of size 281×281 in half-precision floating point arithmetic on the Xilinx FPGA as shown in Figure 7(b). The resulting MI map has a loss of accuracy less than 1.5% compared with the one computed in single precision and is sufficiently accurate for autonomous exploration.

3.2 ASIC Post-layout Verification

The layout of the proposed architecture using a commercial 65nm technology is shown in Figure 8. The post-layout power and chip area of every module within the proposed architecture are shown in Table 1. In summary, the ASIC implementation can be clocked at a maximum frequency of $116.5MHz$ and consumes $162mW$.

3.3 Throughput, Latency & Power Consumption

Figure 9(a) shows the MI throughput vs. number of MI cores for a variety of banking patterns and arbiters using cycle-accurate simulators. If the occupancy grid map is stored in a single dual-port SRAM, the throughput is memory bounded and does not increase with the number of cores as shown with the blue line (baseline). On the other hand, if the memory that stores the occupancy grid map contains unlimited number of ports, the MI throughput should



Technology	Commercial 65nm
Chip Size	2550x2550um
Core Size	2100x2100um
Core Clock	116.5MHz
Total Gates	2981291
Total Power	162mW

Figure 8: Layout of the proposed hardware architecture with 16 cores and 16 banks using a commercial 65nm technology.

increase linearly with the number of cores as shown with the dotted black line (theoretical limit). Our proposed architecture that uses the Latin-square banking pattern and the priority arbiter achieves 91% of the theoretical limit as shown with the purple line.

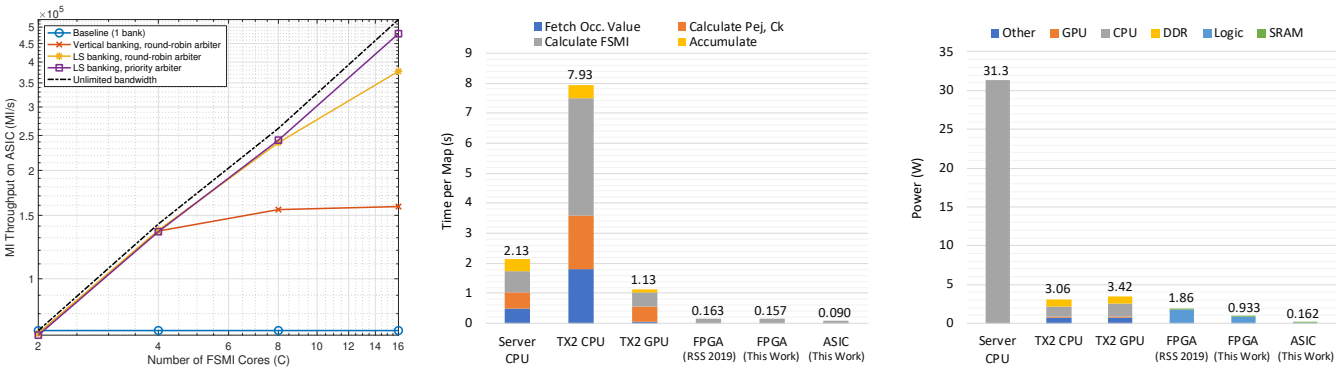
We also validated the MI computation time and power consumption on the following hardware platforms: Server CPU (Intel Xeon E5-4627), TX2 CPU (ARM Cortex-A57), TX2 GPU (256-core NVIDIA Pascal), FPGA (Xilinx XC7Z045) and ASIC (a commercial 65nm technology). Using an occupancy grid map of size 201×201 from a real-world exploration experiment, the ASIC implementation takes $90ms$ to compute MI for the entire map, which is 23× faster than the Server CPU, 83× faster than the TX2 CPU, and 12× faster than the TX2 GPU as shown in Figure 9(b). The ASIC implementation consumes around $162mW$, which is 193× lower than Server CPU, 19× lower than the TX2 CPU, and 21× lower than the TX2 GPU as shown in Figure 9(c).

4 RELATED WORKS

To the best of our knowledge, we proposed the first hardware accelerator for information-theoretic mapping in an earlier version of this

Table 1: Post-layout area and power for each hardware module in the proposed architecture clocked at 116.5MHz using a commercial 65nm technology.

Modules	Area (μm^2)	Area Percentage	Power (mW)	Power Percentage
Workload Allocator	7,037	0.16%	0.2	0.1%
Ray-casting Subsystem	51,013	1.19%	3.1	1.9%
Arbiter	137,917	3.21%	14.6	9.0%
Occupancy Grid Map	2,000,086	46.59%	30.8	19.0%
MI Core (16×)	870,082	20.27%	58.6	36.2%
LUTs for MI Cores	293,523	6.84%	35.3	21.8%
Buffers	196,965	4.59%	2.7	1.7%
IO Drivers & Pads	720,000	16.77%	13.9	8.6%
Auxiliary	16,435	0.38%	2.8	1.8%
Total	**4,293,058**	**100%**	**161.9**	**100%**



(a) MI computation throughput vs. number of cores for a variety of banking patterns and arbiters.

(b) Time for computing MI across an entire occupancy grid map of size 201 × 201.

(c) Power consumption during MI computation.

Figure 9: Throughput, latency and power of MI computation for the proposed architecture.

work at RSS 2019 [11]. In the earlier version, we used a round-robin arbiter to grant memory access requests from the cores. Since the round-robin arbiter cannot grant enough requests at every cycle, we need to prefetch and buffer multiple cells in the map for each MI core during every granted request in order to keep cores utilized. To achieve this, we increased the memory bandwidth by storing four cells per memory address; however, only a maximum of two of these cells are actually used by the core to compute MI, which wastes over 50% of the memory bandwidth. In the current version of this work, we proposed the priority arbiter that is guaranteed to grant the maximum number of requests every cycle. As a result, there is no need to prefetch any extra cells to buffer for each core, and we only need to store one cell per address which eliminates the wasted reads in the previous design. For the design of the MI cores, we reduced the precision of the floating-point arithmetic operations from single precision to half precision so that the power consumption and chip area are reduced with negligible decrease in computational accuracy as shown in Figure 7. Finally, we validated our proposed architecture using a commercial 65nm ASIC technology in addition to an FPGA. By comparing the performance of the hardware architecture in our previous work (RSS 2019) with the proposed architecture in the current work on the same FPGA, the proposed architecture in the current work maintains the same throughput but consumes around 2× less power as shown in Figure 9(c) and requires 2× less programmable logic.

5 CONCLUSION

In this work, we presented a new hardware architecture for high-throughput computation of Shannon mutual information. We argued that the key challenge is to design the memory organization and memory request arbitration subsystems that ensure the maximum utilization of parallel computation cores. The proposed hardware architecture achieves this goal by using a Latin-square banking pattern for memory organization that minimizes the number of memory access conflicts among the cores and a priority arbiter that quickly resolves these conflicts. We implemented the proposed architecture with 16 high-throughput MI cores using a commercial 65nm technology. Our ASIC implementation computes the mutual information for an entire map from a real-world experiment of $10.05m \times 10.05m$ at $0.05m$ resolution in real time at $11Hz$, which is 83× and 12× faster than the ARM Cortex-A57 CPU and NVIDIA Pascal GPU on the Jetson TX2 board respectively. Furthermore, the ASIC implementation consumes $162mW$, which is 21× lower and 19× lower than the ARM Cortex-A57 CPU and NVIDIA Pascal GPU on the Jetson TX2 board respectively. Thus, the proposed architecture eliminates the computational bottleneck for autonomous exploration and reduces the amount of exploration time and trajectory for time-critical missions. In addition, our rigorous analysis of the banking pattern and arbiter introduces a new paradigm for theoretically evaluating hardware design decisions.

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] 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. URL https://youtu.be/N7mi9uXKrP4?t=90.

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

[4] 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.

[5] 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.

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

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

[8] L. Budach. DÉnes, j., a. d. keedwell: Latin squares and their applications. akademiai kiadó, budapest 1974. 547 s., ft 320,-. *Biometrical Journal*, 23(7): 725–726, 1981. doi: 10.1002/bimj.4710230715. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.4710230715.

[9] Yu N Sotskov and Natalia V Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

[10] IBM. Ilog cplex optimization studio. URL https://www.ibm.com/products/ilog-cplex-optimization-studio.

[11] Peter Zhi Xuan Li, Zhengdong Zhang, Sertac Karaman, and Vivienne Sze. High-throughput Computation of Shannon Mutual Information on Chip. In *Robotics: Science and Systems (RSS)*, 2019.