FPGA based Hierarchical Architecture for Parallelizing RRT

Gurshaant Singh Malik Krishna Gupta K Madhava Krishna
Shubhajit Roy Chowdhury

ABSTRACT

This paper presents a new hierarchical architecture for parallelizing the computation intensive rapidly exploring random tree problem. The architecture resembles a tree like structure that agglutinates minimal inter-module communication of a shared memory with data integrity of a distributed memory. Another novelty of this research has been in quantitatively analysing the performance metrics of the RRT algorithm across numerous embedded hardware solutions and ultimately implementing this algorithm on an FPGA to achieve hardware level optimization that offers real time performance and economical power consumption levels. We then analyse our implementation against hardware implementation of other scalable parallel RRT methods for motion planning.

CCS Concepts

•**Computer systems organization** → **Robotic control;** •**Hardware** → **Programmable logic elements;** *Hardware accelerators; Reconfigurable logic applications;*

Keywords

Rapidly Exploring Random Tree; Field Programmable Gate Arrays;

1. INTRODUCTION

Sampling based methods for motion and path planning have gained more interest during the last decade, as computer power has increased. In the field of robotics, the Rapidly exploring Random tree (RRT) has become the customary algorithm for solving mathematically complex single-query motion planning problems [1] involving Kinematic, Holonomic, Non-Holonomic or Kino-dynamic closure constraints [1],[2],[3]. For improving the performance of RRT, techniques like biased exploration [4], [5], controlled sampling [6], and faster nearest neighbor search [7] have been employed. Considerable research effort has also gone into achieving speedup of sampling based motion planning by parallelizing it [8], [9]. Our work centers around multiple modules of RRT working on a single map for single query motion planning of Mobile Robotics.

Amongst the scalable parallel RRT methods proposed so far, the design of distributed RRT [8] proposes the use of distributed memory architecture and updates all the local road-maps by the use of MPI (Message Passing Interface). Another implementation, K-distributed RRT [9], aims to minimize inter-process communication by using STAPL framework to create a globally shared road-map.

In the field of robotics, FPGAs are capable of delivering tightly packed, energy efficient infrastructures adept in fast real time performance. An FPGA allows gate level control of system architecture. This allows the designer to tap the potential of hardware design, allowing control over minute details of arithmetic design, real time parallelization, pipelining of sequential processes. The hardware level flexibility afforded by FPGAs results in designs that are not only fast, but also small and power efficient. FPGA boards generally consume small physical area, making it an ideal solution for robotic systems with constrained dimensions. To confirm our hypothesis, a quantitative analysis of numerous embedded hardware solutions, including FPGA, is presented in the results section.

2. ARCHITECTURE FOR PARALLELIZING RRT

2.1 Current challenges

Since RRT involves randomized exploration of the sample space, ours and many proposed algorithms [8], [9], use the principle of exploratory decomposition as their foundation. In other words, each instantiated RRT module produces its own output and through different write mechanisms, the outputs are integrated to build the road-map. Fig. 1 provides an overview of this design philosophy.

In a multi-module RRT design, an important issue is to decide the write access mechanism that integrates the data from multiple modules and then updates the road-map. There are 2 general philosophies: 1) Distributed 2) Shared. The distributed philosophy employs a scheme by which each RRT module will have its own local road-map. As a result, changes made by it to its local road-map will have no effect on other road-maps. Hence, as shown in Fig. 2, we need a 'mediatorâĂŹ system that updates each RRT-local road-map to changes made by other RRT modules. This will incur significant inter-module communication time in case of large

AIR '15, July 02 - 04, 2015, Goa, India

DOI: http://dx.doi.org/10.1145/2783449.2783461

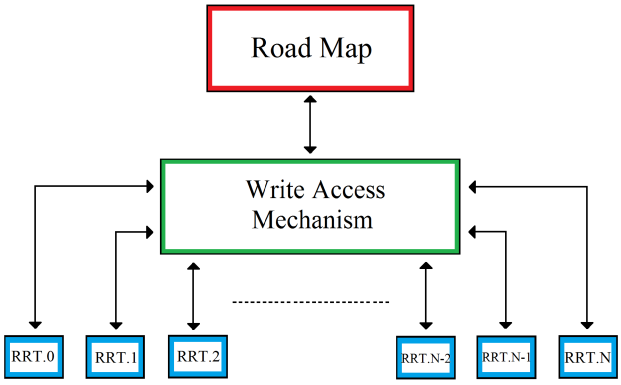


Figure 1: Exploratory decomposition

scale parallelization.

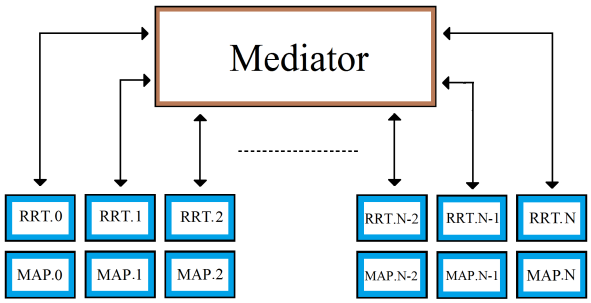


Figure 2: Distributed design philosophy

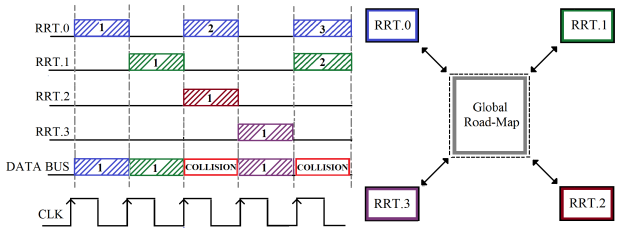


Figure 3: Shared design philosophy

The shared design philosophy, shown in Fig. 3, allows all the RRT modules to have access to the same global road-map. Hence, there is virtually no inter-module communication. But it adds the complexity of data integrity. Since all RRT modules have access to the same global road-map, large scale parallelization, without scheduling, can geometrically increase traffic on global address space, leading to data collisions. Scheduling in a large scale parallel system, on the other hand, will incur significant waiting time, resulting in a slower system overall.

The aim of our research has been to come up with a scalable write access mechanism that combines data integrity (distributed) with minimum inter-module communication (shared) and optimize it further by creating a hardware design of the same on an FPGA.

2.2 Proposal

A very brief overview of our proposal is shown in Fig. 4. Due to the scale of the algorithm, in order to clearly delineate, we subsequently explain each of the highlighted modules in the below image individually and finally present the integrated design. It should be stressed that using an FPGA as our ecosystem allows all the modules to run in parallel in real time. Hence, module level bottlenecks do not transmute into system level bottlenecks.

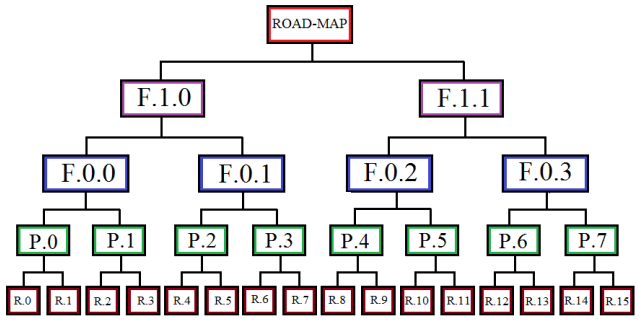


Figure 4: One of the possible designs for 16 RRT Modules

2.2.1 RRT

Multiple instantiations of the RRT module work to achieve multifarious, indiscriminate exploration on a single configuration space.

Algorithm 1: Rapidly Exploring Random Tree

input : configuration space C, root q_{init}
output: new node q_{new}

while ***not*** $RRTdone$ **do**
  $q_{rand} \longleftarrow sampleRandomStateC$;
  $q_{near} \longleftarrow nearestNode(map, q_{rand})$;
  $q_{new} \longleftarrow kinematicExtend(q_{near}, q_{rand})$;
  if $\mid q_{new} - q_{goal} \mid < tol.Error$ **then**
    $RRTdone$;
  $wait.poll()$;
  $transferNewNode(T)$;
  $acknowledge()$;

As shown in Algorithm 1, the first step is to randomly sample the configuration space. The next step is to find the nearest node to this randomly sampled point. For the mobile robot in question, a kinematically constrained, collision free path is laid from the nearest node in the direction of the random node. The RRT module then waits to be polled by its parent to transfer the generated nodes before running another iteration of the exploratory algorithm.

2.2.2 Polling Module

The polling module is a custom buffer stack designed to periodically sample the RRT modules ascribed to it. In the event of completion of a run of RRT, the RRT module waits to be polled by the polling module to transfer the generated data before running another iteration of RRT. The data is then transferred through a write-acknowledge mechanism to preserve data integrity.

As shown in Fig. 5, the POLL periodically polls the RRT modules. The first 4 rising edges of clock do not lead to capture of the data bus since none of the 4 RRT modules are ready to transfer data when they are polled. At the 5th

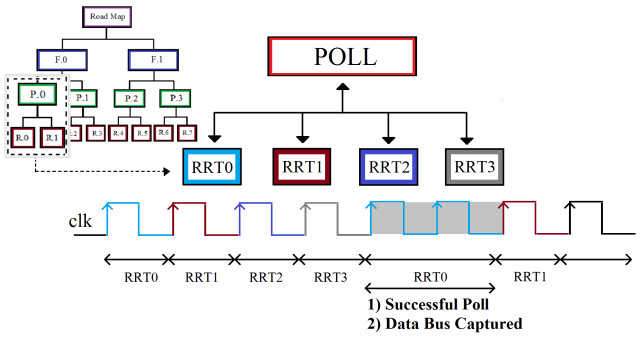


Figure 5: The illustrative inset is only for showcasing the position of POLL

rising edge of clock, RRT0 is polled successfully and hence the data bus is captured by it. RRT0 then transfers all the data to the custom stack before freeing the data bus for polling to resume.

2.2.3 *FIFO*

As the name suggests, FIFO is a first in-first out stack. It polls FIFOs, other polling modules, but not RRT modules. Periodic polling for write access to module underneath preserves data integrity while instant read access upon request by higher level modules ensures minimum data flow latency. The First in-First out nature of the stack ensures chronological queuing up of data. Fig. 6 shows the working of FIFO in detail.

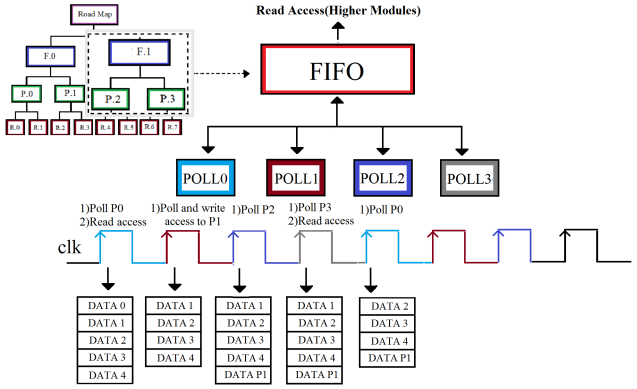


Figure 6: The illustrative inset is only for showcasing the position of FIFO

The data stack shown belongs to the FIFO. At the first rising edge of clock, POLL0 is polled for data and the parent module of FIFO requests read access of FIFO. Hence, DATA0 is transferred to the parent module. At the second rising edge of clock, the POLL1 is successfully polled and captured. Hence, POLL1 transfers DATAP1 to the FIFO. At the third rising edge, POLL2 is unsuccessfully polled. At the 4th rising edge, the parent module again requests a read access. As a result, DATA1 is transferred to it. Hence the FIFO module continues to periodically poll its child modules and grants instant read access to the parent module upon request.

2.2.4 *Integrated Design*

As shown in Fig. 7, the data stems from the RRT modules and flows through higher levels of hierarchy to reach the global map. At the deepest level, P.0 chronologically polls (RRT.0, RRT.1), P.1 polls (RRT.2, RRT.3) and so on.Going up, F.0.0 polls (P.0, P.1), F.0.1 polls (P.2, P.3) and so on. Going up a level, F.1.0 polls (F.0.0, F.0.1) and F.1.1 polls (F.0.2, F.0.3). Finally, at the highest level, the global road-map is updated by F.1.0 and F.1.1.

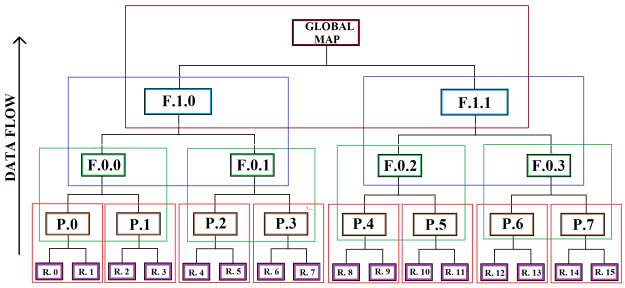


Figure 7: 16-RRT binary tree design.

At all levels, chronological polling for data by parent module preserves data integrity (distributed). Since scheduling has to happen only amongst child modules of the same parent (siblings), the waiting time before data from a child module starts to flow is significantly reduced (shared). This is a pliant architecture since the designer has control over the number of module instantiations at any level and the overall depth of the tree. Since any child module only communicates with its parent, the designer can achieve zero intra-level communication. Polling only amongst siblings ensures zero data collision and fast data flow.

3. FPGA IMPLEMENTATION

The design test platform, Virtex^{TM}-6XC6VLX75T FPGA delivers 6.6Gbps GTX trans receivers and built-in PCIe and tri mode Ethernet MAC blocks that help meet higher bandwidth and performance demands with less power.

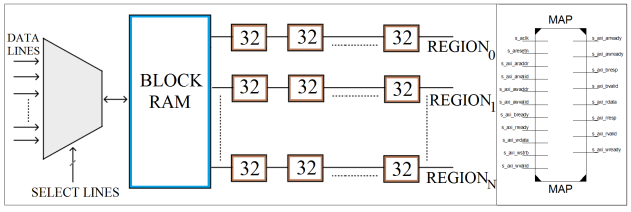


Figure 8: Technology Schematic of the Map

The global road-map has been designed as a BLOCK RAM with single port write channel. The output lines map to the 2D configuration space as shown in the technology schematic of the map in Fig. 8 . Each output string details information about a specific region of the map and the child 32 bit strings represent the Cartesian co-ordinates the robot has traversed. The multiplexer at the input allows one of the multiple modules to access the single write port.

The native FIFO can be customized to utilize BLOCK RAM or DISTRIBUTED RAM or built in FIFO FPGA resources.As the RTL and technology schematics in Fig. 9 show, we use built in FIFO resources to create high performance , area optimized FIFO module. The First Word Fall

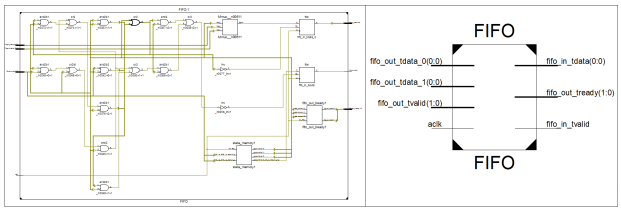


Figure 9: The RTL and Technology Schematic of FIFO

Through is chosen as the mode of operation for the FIFO interface.

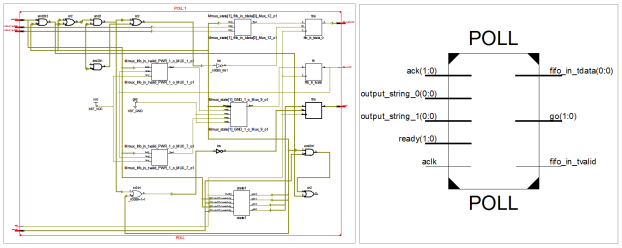


Figure 10: The RTL and Technology Schematic of POLL

The RTL and technology schematics in Fig. 10 show the hardware level implementation of the polling module. The POLL is implemented as a sequential Finite State Machine (FSM). Isochronal cyclic polling of child RRT modules germinated by rising edge of clock leads to capture of data bus by one of the children, which then transfers its generated nodes via write-acknowledge mechanism.

A pseudo-random number generator generates a random state for the mobile robot in use. We use the box [7] method to find the nearest node. Deployment of DSP48E1 slices minimizes the time complexity of distance computation. CORDIC cores are used for computation of trigonometric functions. DSP slices are then used for path laying. Fig. 11 shows the design overview of RRT.

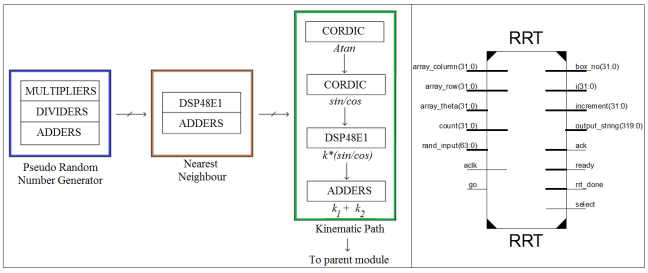


Figure 11: Design overview of RRT

4. RESULTS

4.1 Platform analysis

The platforms that were analysed included an ARM® Cortex™-A7, Intel® core™-i5 2430M, Intel® Atom™ N455, Arduino® Duemilanove™-328 and a Virtex™-6 FPGA.

The test environment consisted of 3 different configuration spaces measuring 200 × 200 each as shown in Fig. 12. The tests were run on a ground differential drive robot with 3 degrees of freedom.

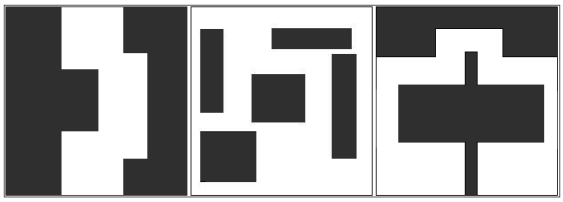


Figure 12: Test Environment

Performance analysis consisted of adding 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024 collision free nodes to the test environment. Since RRT is probabilistic by nature, each test case was run for 8000 iterations and mean over the whole resultant sample space of the 3 configuration spaces was computed. Power consumption was measured by running the algorithm with no exit condition for 15 minutes. The average of power over the complete time-line was then computed.

Let $T_N(p)$ be the time taken by platform p to add N nodes to the test environment. Let $P(p)$ be the average power consumed by platform p. We define:

$$Relative.Performance_N(p) = \frac{T_N(FPGA)}{T_N(p)}$$

$$Relative.Throughput_N(p) = \frac{T_N(FPGA) \times P(FPGA)}{T_N(p) \times P(p)}$$

As shown in Fig. 13, the hardware implementation of the algorithm on Virtex™-6 FPGA performs faster compared to its software implementation across all the platforms. The Intel® i5 performs the fastest among the software implementations with a best relative performance of 0.38. The Intel® Atom™ comes in a distant third with a best relative performance of 0.08, followed by Cortex® A7. The Arduino® is only able to add a maximum of upto 64 nodes before it runs out of SRAM. Trigonometric and fixed point computations, heavily used in RRT, combined with high I/O latency, tend to bottleneck the performance of the software implementation. Through the use of dedicated hardware like DSP48E1 slices and relatively zero I/O latency, such bottlenecks do not arise in hardware.

As shown in Fig. 14, although Virtex™-6 FPGA consumes the second highest power at 2.5 W, it still offers the best combination of speed and power consumption. The Intel® Atom™ and Cortex® A7, with their low power consumption (518 mW, 397 mW), edge ahead of the fast but high power consuming (9.5 W) Intel® i5. Reconfigurable architecture of an FPGA allows us to fine-tune the design for lower power consumption or better performance in accordance to the specifications of the mobile robot in question.

4.2 Algorithm Analysis

This section presents a quantitative analysis of the 3 algorithms: 1)Hierarchical 2)Distributed 3)K-distributed. Consequently, distributed and K-distributed were also ported to Virtex™-6 FPGA. We believe that by harnessing the flexibility of FPGA systems at gate level hardware design, one can further reduce the inter process communication to achieve better speed up and efficiency. The current hierarchical architecture exploits this hardware level flexibility

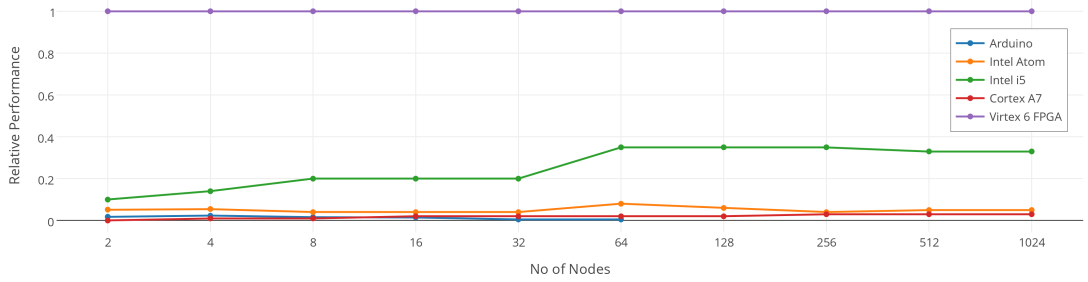
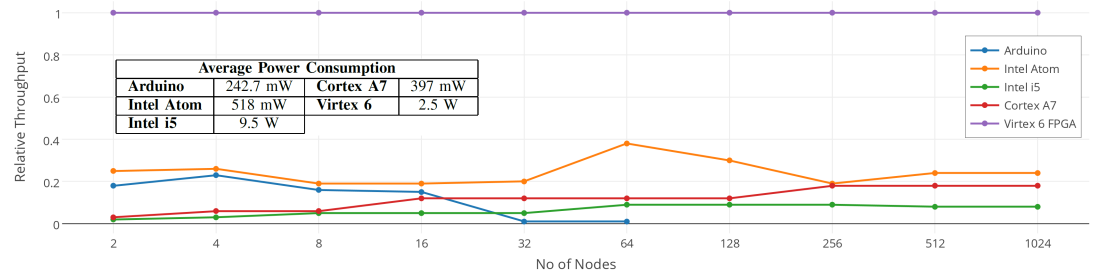


Figure 13: Relative Performance



Average Power Consumption			
Arduino	242.7 mW	**Cortex A7**	397 mW
Intel Atom	518 mW	**Virtex 6**	2.5 W
Intel i5	9.5 W		

Figure 14: Relative Throughput

afforded by an FPGA to improve upon the speedup and efficiency of the earlier methods. In the subsequent comparisons, K=1 is same as the distributed design, whereas K>1 refers to the K distributed algorithm.

Configuration spaces with varying degrees of geometrical constraints were used as test environment, as shown in Fig. 15. Inspired by the test environment of Distributed RRT, we used 2D planar projections of the same problem. The tests were run on a ground differential drive robot with 3 degrees of freedom. GAB, with the longest distance to cover, involves very weak geometrical constraints. BCL has a relatively short pathway but is geometrically constrained by several side chain blocks. CALB has the tightest geometrical constraints, with the pathway constrained by side chain blocks of varying length. The pathway is shorter than GAB but longer than BCL.

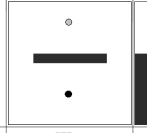
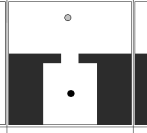
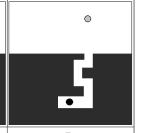
Problem Name	**GAB**	**BCL**	**CALB**
Configuration Space			
Geometrical Constraints	III	II	I
Distance	I	III	II

Figure 15: Test Environment

Let P be the total number of RRT modules instantiated. We define $T(P)$ as the time taken by an algorithm with P modules to reach the desired configuration. The RRT, as delineated in Algorithm 1, was used for this purpose.

Speedup $S(P) = \frac{T(1)}{T(P)}$

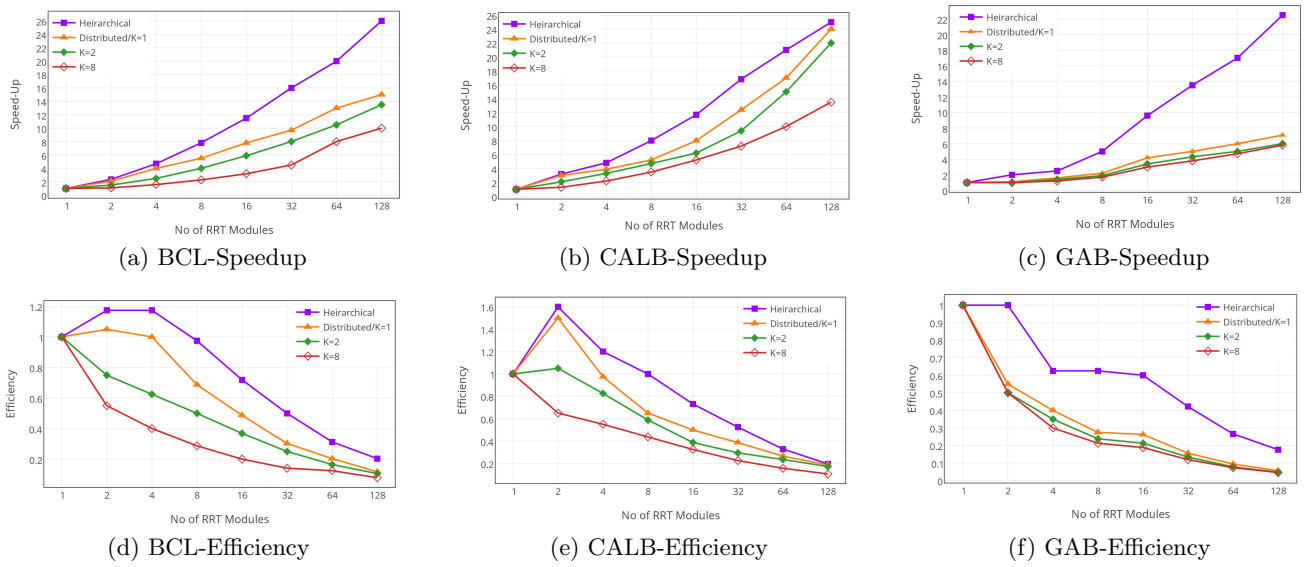
Efficiency $E(P) = \frac{S(P)}{P}$

As more and more modules are added in parallel, the speedup is a good reflection of the scalability of the algorithm. Theoretically, $S(P)$ is bounded by P but we observed super linear speedup during our tests ($S(P) > P$).

The test cases for the above mentioned analysis included estimation of $T(P)$, $S(P)$ and $E(P)$ parameters for P=1, 2, 4, 8, 16, 32, 64, 128. Owing to the probabilistic nature of the algorithms, the parameters were averaged out over 1000 iterations of each test case. The focus of the analysis was to map the scalability of the algorithm to module instantiations and physical constraints of the map.

As shown in Fig. 16, the hierarchical design scales much better than distributed and K-distributed for GAB with the best speedup of 22.5. The Distributed comes behind hierarchical with maximum speedup of 7.1. The K-distributed offers the lowest speedup out of all three, with K=2 offering a relatively higher speedup of 6 with respect to K=8. All 3 algorithms scale better with the tighter constrained configuration space of BCL. The hierarchical scales to a speedup of 26, followed by distributed that offers a maximum speedup of 15. K=8 for K-distributed comes in last with a speedup of 10. CALB, with the tightest constraints, scales the distributed design the best with respect to its scalability in other configuration spaces. Although hierarchical still showcases the best speedup of 25, it is closely followed by the distributed design with a speedup of 24. The K-distributed RRT offers a speedup of 22 and 13.5 for K=2 and K=8 respectively. We now qualitatively discuss the test results.

Hierarchical architecture has been designed to reduce inter-module communication by limiting scheduling to occur amongst siblings only. In contrast, the addition of modules in distributed RRT introduces a large inter-module communication overhead which weighs down the speedup. As the Fig. 16 shows, the hierarchical design generally scales better than the distributed design in view of this fact. The configuration space of GAB offers expansive, unconstrained areas. Hence a substantial numbers of nodes can be added in a small timeline. This introduces an exhaustive amount of communication load, causing the distributed design to scale poorly rel-



(a) BCL-Speedup (b) CALB-Speedup (c) GAB-Speedup

(d) BCL-Efficiency (e) CALB-Efficiency (f) GAB-Efficiency

Figure 16: Comparative analysis of Heirarchical, Distributed and K-Distributed (K=2,8) for different configuration spaces.The graphs have a smoothing factor of 0.2

ative to the hierarchical design. Relative to GAB, BCL has tighter physical constraints. Slower addition of nodes result in a smaller communication/scheduling overhead, resulting in both algorithms to scale better. Out of the 3 configuration spaces, CALB offers the most unyielding physical constraints. The resultant near zero communication/scheduling load allows both the algorithms to scale really well and distributed to perform nearly as well as heirarchical.

To reduce inter-module communication, K-distributed RRT introduces a parameter K that can be adjusted to vary how often the global graph is updated by one individual processor. As a result, a higher K value results in trees more localized to a region of the configuration space. This results in localized propagation of trees based on the local road-map affecting performance. Unsurprisingly, K-distributed RRT scales very poorly to increase in K values.

5. CONCLUSIONS

This paper introduced an FPGA implementation of hierarchical design for parallelizing RRT. We backed our choice of platform by quantitatively analyzing performance, power and throughput parameters on various embedded solutions. FPGA implementation shows a tangible increase in performance and throughput, attributed to relatively low I/O latency and custom hardware design of the algorithm.

A major portion of this paper focused on a new hierarchical design of parallelizing RRT that combined distributed memory's data integrity with shared memory's fast data flow. We presented a tree like structure that reduced scheduling to occur amongst siblings only. Quantitative analysis of this design over 3 different configuration spaces revealed that the hierarchical design scaled better relative to the distributive and K-distributive design. We attribute this behaviour to the design being relatively immune to increase in inter-module communication load.

6. REFERENCES

[1] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[2] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[3] Juan Cortés and Thierry Siméon. Sampling-based motion planning under kinematic loop-closure constraints. In *Algorithmic Foundations of Robotics VI*, pages 75–90. Springer, 2005.

[4] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2859–2865. IEEE, 2009.

[5] Russ Tedrake. Lqr-trees: Feedback motion planning on sparse randomized trees. 2009.

[6] Léonard Jaillet, Anna Yershova, Steven M La Valle, and Thierry Siméon. Adaptive tuning of the sampling domain for dynamic-domain rrts. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2851–2856. IEEE, 2005.

[7] Mikael Svenstrup, Thomas Bak, and Hans Jørgen Andersen. Minimising computational complexity of the rrt algorithm a practical approach. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5602–5607. IEEE, 2011.

[8] Didier Devaurs, Thierry Siméon, and Juan Cortés. Parallelizing rrt on distributed-memory architectures. In *Proc. IEEE ICRA'11*, pages pp–2261, 2011.

[9] Nick Stradford, Sam Ade Jacobs, and Nancy M Amato. Scalable parallel rrt method for motion planning.