Optimized Local Path Planner implementation for GPU-accelerated embedded systems

Hidden Authors - Double Blind

***Abstract*—Autonomous vehicles are latency-sensitive systems. The planning phase is a critical component of such systems, during which the in-vehicle compute platform is responsible for determining the future maneuvers that the vehicle will follow. In this paper, we present a GPU-accelerated optimized implementation of the Frenet Path Planner, a widely known path planning algorithm. Unlike the current state-of-the-art, our implementation accelerates the entire algorithm, including the path generation and collision avoidance phases. We measure the execution time of our implementation and demonstrate dramatic speedups compared to the CPU baseline implementation. Additionally, we evaluate the impact of different precision types (double, float, half) on trajectory errors to investigate the trade-off between completion latencies and computation precision.**

***Index Terms*—Planning, Autonomous vehicle, Parallel, GPU, Racing**

I. Introduction

Autonomous vehicles (AVs) are nowadays adopted in complex and safety-critical scenarios, such as urban driving and race competitions. AVs are composed of different phases such as path planning and collision avoidance. These phases are latency-sensitive since their execution time must be proportional to the vehicle speed [1], [2], [3]. For this reason, AVs require to efficiently process a significant amount of data within the onboard computing systems in a timely fashion. Therefore, high-performance embedded computers featuring highly parallel accelerators such as GPUs, are key enabling technologies. These platforms are amenable to accelerating the complex algorithms of autonomous vehicles. In this work, we focus on the *Local Planner* phase of the autonomous vehicles pipeline and we propose a novel implementation of the well-known *Frenet Path Planner* algorithm that exploit the GPU acceleration. We can summarize the contribution of this paper as follows: 1) We propose a novel GPU implementation for the *Frenet Path Planner* algorithm. To the best of our knowledge, the proposed implementation is the only one that ports on GPU the entire algorithm pipeline. 2) We test our proposal using different precision types (*double*, *float*, *half*), aiming at understanding the impact on execution time and on the precision of the generated trajectories. 3) We measure the impact of the proposed implementation both in terms of execution time and precision compared to a baseline CPU implementation. 4) We made the source code of our implementation publicly available[1] and, to the best of our knowledge, this represents the first publicly available GPU-based *Frenet Path Planner* implementation.

[1]https://github.com/HiPeRT/FrenetTenth

II. Related Work

The *Planning* problem has been extensively studied in previous literature. In [4] and [5], for example, a path is generated considering kinematic and dynamic constraints. The dynamic model is also considered in [6], in which a spatiotemporal lattice with vehicle feasible states is proposed. Similarly in [7] and [8] the kinematics quantities are optimized to find the path. In the race context, a fully reactive planner called the Follow The Gap method (FTG) [9] is used. It avoids collisions by aiming at the center of the maximum gap among obstacles. The model predictive control is used in [10], [11], [12]. In [13] the problem is formulated as an optimization problem and it is solved using the gradient descent method. Another approach is to generate a single path and iteratively improve it [14]. Lastly, some contributions are based on the generation of different paths to then choose the best one [15], [16] according to predefined metrics. The Rapid Exploring Random Tree algorithm [17] is used in [18] and [19]. In [16] the authors use *Frenet Coordinates* to split the generation of lateral and longitudinal movements and in [20] the authors consider dynamic objects. Our work exploits *Frenet Coordinates* and will focus on an optimized GPU implementation in which the performance analysis will also account for different settings w.r.t. data type precision. In [21] the authors accelerate on GPU only the path generation on *Frenet Coordinates* omitting to port on GPU other phases such as obstacle avoidance. Differently, we also ported both obstacle avoidance and the best path selection on the GPU. We use NVIDIA CUDA to exploit GPU parallelism in different ways: using parallel threads [22], exploiting *Shared Memory* and *syncthread* directive; overlapping computing and memory transfers using *Streams* and *Events* [23].

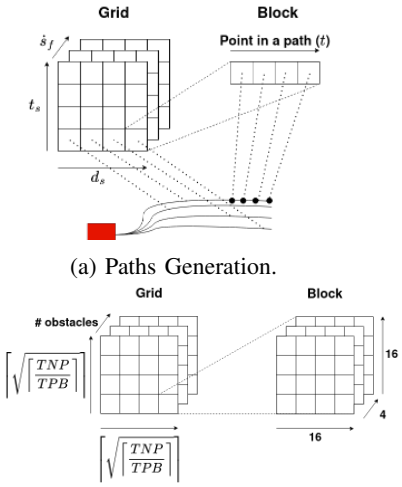
III. Frenet Path Planner

The path planning phase of an autonomous driving system consists of the generation of a path that the vehicles can safely follow. Typically such a path is discretized and it is represented as a list of path points. The *Frenet Path Planner* [16] uses the *Frenet Coordinates* [24], [25] to *generate* a set of paths. Those paths are converted into the *World coordinates* for *collision check* with the obstacles present in the environment. Finally, between the paths that do not show any potential collision, the best path is chosen based on a cost function. Considering the actual vehicle position in the *Frenet Coordinates* (s_0,d_0), a single path is a contiguous trajectory that links (s_0,d_0) to a final position (s_f,d_f), the corresponding velocities and accelerations are expressed as $\dot{s}, \ddot{s}, \dot{d}, \ddot{d}$. The *Frenet Path Planner* generates a set of paths that start from (s_0,d_0) and

terminate in a possible endpoint (s_f,d_f). This set is generated considering: 1) the initial state $S_0 =< s_0, d_0, \dot{s}_0, \dot{d}_0, \ddot{d}_0 >$; and 2) a candidate end state $S_f =< s_f, d_f, \dot{s}_f, \dot{d}_f, \ddot{d}_f >$. Each value of each end state leads to a range of possible values chosen as parameters. The path that links the start state to the end state is computed resolving two linear systems (see [16] for details). The path is discretized in f points, so to define a list of path points $P = [p_0, p_1, ...p_f]$ and it has an associated cost typically determined by the path jerk. The path P is expressed in *Frenet Coordinates*, therefore it must be reconverted back to *World coordinates*. At this point each path P is tested for collisions against obstacles that might be present in the environment. The distance between each path point and each obstacle is computed and the path is filtered out if it will collide with some obstacle. Finally, the path with the best cost and that does not collide with obstacles is returned as the path to follow.

IV. Our Implementation

We provide a novel GPU-based implementation of *Frenet Planner*. The goal is to reduce the computational time of the algorithm. Since, in the first part, each path is generated independently, we have exploited GPU parallelism for this work. Moreover, each point $p \in P$ can be computed independently. We set up a CUDA kernel with a three-dimensional launch grid that would allow us to compute each path concurrently (see Figure 1a): in each dimension, there is a different parameter and each block has a different value that this parameter can have. Moreover, we have mapped the computation of the points of a path to a different thread of the block associated with the path itself. By doing so, each thread computes the point of the path considering a different value of t. In this phase also the *Word Coordinate* of each point and the cost of the path is computed. Since all points of a path must be computed before performing the cost computation, we use the *synchthread* primitive to ensure that all threads associated with the path are finished. At this point, one selected thread performs the reduction to retrieve the path cost exploiting the *Shared Memory* of the GPU. For the next phase, in which each path is checked among all obstacles, we have implemented a CUDA kernel that performs each test in parallel. If the check fails, then the path of the point is marked as collided and the cost of the path is set to the maximum value to ensure that it will not be chosen as the best path. The kernel is launched spreading all path points on different threads; each block has a fixed dimension of $16 \times 16 \times 4$ threads (resp. on x, y and z axes), for a total of 1024 threads per block (TPB). The grid is composed of an amount of blocks that is adequate to cover the total number of points (TNP) on axes x and y, the axis z is used to map the obstacles (see Figure 1b). When all paths are checked, the one with the smaller cost is retrieved using a CUBlas API function call; specifically, by invoking the function *cublasI<T>amin*[2]. This function returns the index of the smaller item. Given that the paths that collide have the maximum cost and the best path choice procedure returns the path with the smallest cost value. If the retrieved best path is marked as collided, it means that there are no feasible paths around the generated trajectories.



(a) Paths Generation.

(b) Paths Generation.

Fig. 1: CUDA launch configurations.

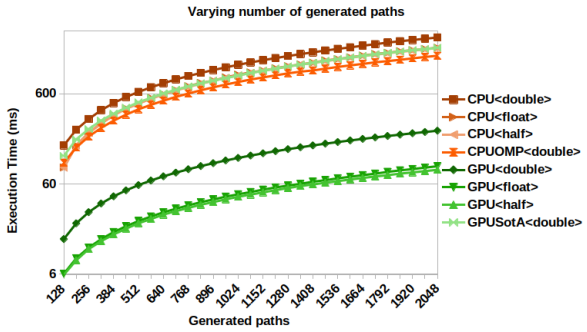
V. Results

We compared our implementation (GPU) with a publicly available CPU implementation of the *Frenet Planner*[3] (CPU), with the same implementation using OpenMP[4] to exploit CPU parallelism (CPUOMP), and with the state-of-the-art GPU implementation [21] (GPUSotA), the authors do not share the code but we have replicated it. We performed the comparison by varying the precision type (*half*, *float*, *double*). These precision types have different data sizes as defined by IEEE standard [26] and this means a different amount of data to exchange between CPU and GPU but also different accuracy in the results. We performed our tests on an NVIDIA Xavier AGX. This embedded board is equipped with a CUDA-capable GPU (512 NVIDIA cores) and an ARM CPU (8 cores). It features 32GB of DDR SDRAM. We performed two types of measures. One varying the number of generated paths while maintaining a fixed path length, and the other varying lengths while maintaining a fixed number of generated paths. For each test, we have performed 100 iterations, and we report the average time. The average overall execution times for these two variants are reported in Figure 2, with Figure 2a displaying the results for the varying number of generated paths and Figure 2b displaying the results for the varying path length. Our GPU implementation is significantly faster than the other implementations (including SotA). The precision type impacts the execution time, indeed, the execution time is higher using *double* precision than using *float* or *half*. The execution times of the latter two types are similar but *half* type produces a lower execution time. Indeed, on GPU, using the *float* precision type the times, on average, are reduced by about 74 ms with respect to *double* in the first experiment, and about 36 ms in the second; using the *half* precision type, the reduction is

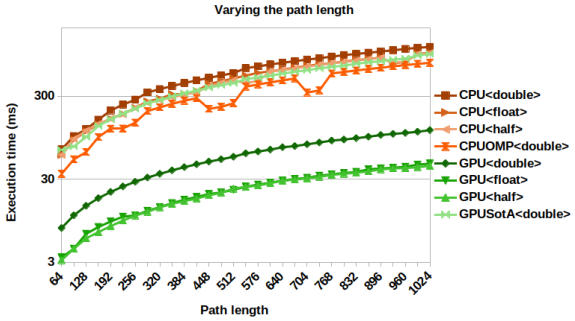
[2]https://docs.nvidia.com/cuda/cublas/index.html#cublasi-lt-t-gt-amin

[3]https://github.com/arvindjha114/frenet_planner_agv

[4]https://www.openmp.org/



(a) Ex. Time - Varying numbers of generated paths



(b) Ex. Time - Varying the path length

Fig. 2: Overall average execution time (ms)

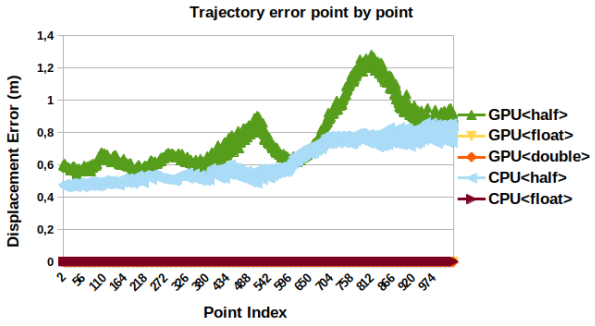


Fig. 3: Trajectory error for each point

by about 3 ms with respect to *float* in the first experiment, and about 1 ms in the second. The trend of all implementations is linear but the CPU implementation shows a steeper trend due to its serialized processing. In the first experiment (varying the number of generated paths) the speedup achieved by GPU implementation with respect to the CPU implementation using the same precision is constant for *double* (around 11x), increases up to 20x for *float* precision, and up to 22x for *half* precision. In the second experiment (varying the path length) the speedup is less constant but has the same average values as the first experiment. The GPU implementation of *Path generation* phase is constructed exploiting *shared memory* and minimizing the critical operations of a GPU accelerated application, i.e. accesses to global memory and branch divergence. Moreover, the merging of *World coordinates* conversion in the same kernel of *Path generation* reduces the overhead of kernel launches. Moreover, we elected to implement data transfers between the host (CPU) and the device (GPU) using explicit copies to have more control over these operations and be able to execute them in parallel to the kernel execution using *Streams*. In this manner, we can copy the obstacles data (required by the collision check task) during the path generation effectively nullifying the memory copy overhead. Finally, performing the best path choice on GPU reduces the number of memory copies between the CPU and GPU. Indeed, only the best path is copied instead all. These aspects contribute to the huge execution time reduction of this phase with respect to the CPU implementation (up to 22x) but also with respect to the GPU SotA (up to 8x) in which the authors do not consider the memory copies overhead and it results in a slower execution compared to CPU OMP considering the overall pipeline.

A. *Precision error*

Using fewer bits to represent input and output data leads to a significant reduction in execution times. We investigate how reducing data type precision and the different hardware architectures affect the quality of the computed trajectories. We compared the trajectories generated using the different data precision formats in both CPU and GPU implementations with respect to trajectories generated by the CPU *double* implementation (henceforth our baseline). We compare paths generated by different precision types but with the same parameters and the *Frenet* algorithm was set to generate 1024 different paths, each of them with 1024 path points. On top of this, we have simulated a vehicle that follows the selected path but when it reaches a new location it generates again the path to follow. In this way, the generated paths will always feature a different initial state and this is the typical behavior of a vehicle that uses the *Frenet path planner*. We repeat the generation 300 times and we compare each point of each selected path. We have used the Average Trajectory Error (ATE) which is the average displacement error of each point of the path. The displacement error is computed as the Euclidean Distance. As expected the ATE is 0m for GPU *double* implementation. On the other hand *half* precision shows a larger error (0.7747m in GPU and 0,6183m in CPU) due to the fact that the reduced precision impacts the results. The error in the *float* versions is negligible (0.0005m in GPU and 0,0027m in CPU). The small difference between CPU and GPU versions using the same data type is the result of the different hardware architectures. We also report in Figure 3 the errors for each point of the path, from 0 to 1024. For each point, we report the mean error measured in each of the 300 paths. We can see that the error within a trajectory increases as the vehicle move further from the path starting point, hence, the precision error tends to maximize towards the last points. Typically, the vehicle follows the selected path for the first points and then recomputes it based on the new state, i.e. its new position; therefore, it is unlikely that the vehicle will reach the ending points that were initially generated. This aspect is crucial since we previously highlighted that precision errors are more significant towards those final points, as the trajectory

is constantly re-generated, and the path final points are almost never reached. For this reason, it does make sense to measure the average error of the path traveled by the vehicle using the sequence of selected paths computed during the trip. We measured the ATE of the trajectory of the simulated vehicle that moves as described above. By doing so we obtain the following errors: 0.5993m for *half* GPU (0,4801m on CPU), 0.0001m for *float* GPU (0,0025 on CPU), and always 0m for *double* GPU. Qualitatively, the errors reported for *float* and *half* precision do not impact the quality of the generated trajectory due to the fact that the average error is small and allow the vehicle to avoid obstacles. Indeed, in all cases, the simulated vehicle is able to avoid an obstacle, so the error does not mine the safety of generated trajectory. To conclude, by decreasing data precision we obtain noticeable performance improvements in terms of execution times (especially using GPU). This comes at the cost of larger ATE values. The choice of precision within data types depends on the context; our experiments, however, clearly show how the *float* version on GPU easily represents the best choice, as it leads to an extremely low ATE compared to the huge execution time reduction (60% of time reduction respect to *double* on GPU).

VI. Conclusion

In this work, we proposed a novel and optimized GPU implementation of the *Frenet Path Planner* algorithm. To the best of our knowledge, this is the first implementation that ports the entire algorithm pipeline on GPU. Moreover, we release the source code of our implementation. We investigated the execution time of our implementation compared to the other CPU and GPU SotA implementations and we obtain a speedup of up to 22x. Moreover, we investigated the execution time using different data types: *double*, *float*, *half*. Regarding this aspect, we also investigated the impact on trajectory precision when varying these data types. The results confirm that using our implementation with *float* is the best trade-off between computational time and trajectory precision. In future work, we plan to investigate the use of CUDA Unified Virtual Memory (UVM) instead of explicit memory copies and to integrate in our implementation a trajectory prediction for the surrounding vehicles in order to manage dynamic obstacles. Lastly we want to perform an energy consumption analysis.

References

[1] A. Raji, A. Liniger, A. Giove, A. Toschi, N. Musiu, D. Morra, M. Verucchi, D. Caporale, and M. Bertogna, "Motion planning and control for multi vehicle autonomous racing at high speeds," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2775–2782, IEEE, 2022.

[2] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp, "Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3149–3154, IEEE, 2019.

[3] M. Verucchi, L. Bartoli, F. Bagni, F. Gatti, P. Burgio, and M. Bertogna, "Real-time clustering and lidar-camera fusion on embedded platforms for self-driving cars," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pp. 398–405, IEEE, 2020.

[4] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, pp. 1048–1066, Nov. 1993.

[5] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001.

[6] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (St. Louis, MO, USA), pp. 1879–1884, IEEE, Oct. 2009.

[7] J.-w. Choi, R. E. Curry, and G. H. Elkaim, "Continuous Curvature Path Generation Based on Bézier Curves for Autonomous Vehicles.," *IAENG International Journal of Applied Mathematics*, vol. 40, no. 2, 2010. Publisher: Citeseer.

[8] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for Bertha - A local, continuous method," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, (MI, USA), pp. 450–457, IEEE, June 2014.

[9] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm:"follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[10] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, pp. 308–333, Mar. 2009.

[11] C. Gotte, M. Keller, C. Hass, K.-H. Glander, A. Seewald, and T. Bertram, "A model predictive combined planning and control approach for guidance of automated vehicles," in *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, (Yokohama, Japan), pp. 69–74, IEEE, Nov. 2015.

[12] S. J. Anderson, S. B. Karumanchi, and K. Iagnemma, "Constraint-based planning and control for safe, semi-autonomous operation of vehicles," in *2012 IEEE Intelligent Vehicles Symposium*, (Alcal de Henares , Madrid, Spain), pp. 383–388, IEEE, June 2012.

[13] P. Fiorini and Z. Shiller, "Time optimal trajectory planning in dynamic environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, (Minneapolis, MN, USA), pp. 1553–1558, IEEE, 1996.

[14] T. Heil, A. Lange, and S. Cramer, "Adaptive and efficient lane change path planning for automated vehicles," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, (Rio de Janeiro, Brazil), pp. 479–484, IEEE, Nov. 2016.

[15] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, pp. 640–673, Sept. 2008.

[16] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," in *2010 IEEE International Conference on Robotics and Automation*, (Anchorage, AK), pp. 987–993, IEEE, May 2010.

[17] S. M. LaValle and others, "Rapidly-exploring random trees: A new tool for path planning," 1998. Publisher: Ames, IA, USA.

[18] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, and J. How, "Motion planning for urban driving using RRT," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Nice), pp. 1681–1686, IEEE, Sept. 2008.

[19] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, (Gold Coast City, Australia), pp. 391–396, IEEE, June 2013.

[20] M. Moghadam and G. H. Elkaim, "An Autonomous Driving Framework for Long-Term Decision-Making and Short-Term Trajectory Planning on Frenet Space," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, (Lyon, France), pp. 1745–1750, IEEE, Aug. 2021.

[21] J. Fickenscher, S. Schmidt, F. Hannig, M. Bouzouraa, and J. Teich, "Path Planning for Highly Automated Driving on Embedded GPUs," *Journal of Low Power Electronics and Applications*, vol. 8, p. 35, Oct. 2018.

[22] N. Capodieci, R. Cavicchioli, and A. Marongiu, "A taxonomy of modern gpgpu programming methods: on the benefits of a unified specification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1649–1662, 2021.

[23] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna, "Dissecting the cuda scheduling hierarchy: a performance and predictability perspective," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 213–225, IEEE, 2020.

[24] F. Frenet, "Sur les courbes a double courbure.," *Journal de mathématiques pures et appliquées*, pp. 437–447, 1852.

[25] J.-A. Serret, "Sur quelques formules relatives à la théorie des courbes à double courbure.," *Journal de mathématiques pures et appliquées*, pp. 193–207, 1851.

[26] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.