

Energy- and Runtime-Efficient Trajectory Planning via SIMD Vectorization with Precision Reduction

omitted for review

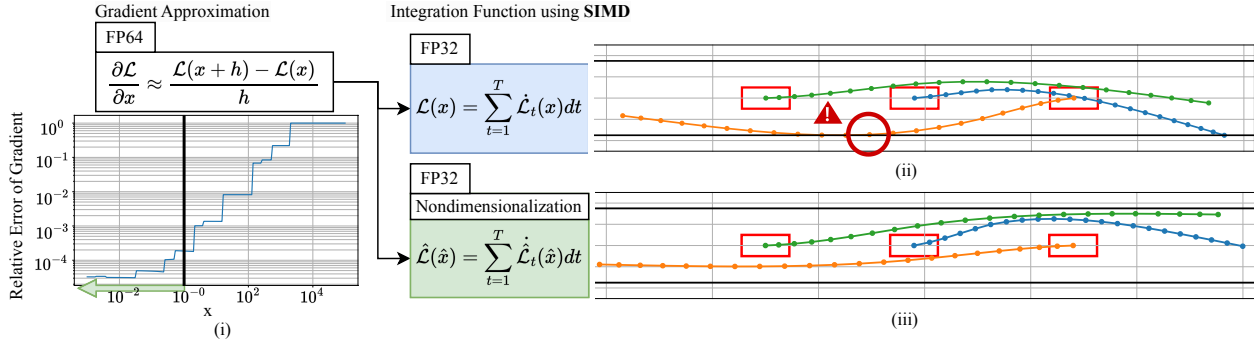


Fig. 1: Scheme illustrating the idea of using SIMD to optimize the runtime- and energy- efficiency of trajectory planning and nondimensionalization to optimize the relative error of gradient while using reduced floating-point precision, (i) the increasing relative error of gradient (shown by a polynomial example) with larger numerical values due to reduced floating-point precision, and the planned trajectories from the integration function without (ii) and with (iii) nondimensionalization.

Abstract—Trajectory planning is a critical module in autonomous driving, where real-time capability and accuracy directly impact safety. Owing to the inherent algorithmic complexity, the energy consumption of trajectory planning becomes a non-negligible concern. This paper proposes single instruction, multiple data (SIMD)-based CPU acceleration of trajectory planning, combined with nondimensionalization for precision-aware optimization. The approach analyzes the trade-off between reduced bit-width and planning reliability, introducing a new scaling scheme that optimizes numerical accuracy under reduced-precision computation. Experimental results on the NVIDIA Jetson Orin (Arm Cortex-A78AE with Arm NEON) demonstrate that vectorized trajectory planning with single-precision floating-point arithmetic (IEEE 754) achieves a $1.9\times$ speedup and 44% energy reduction compared to scalar double-precision arithmetic, while half-precision floating-point arithmetic fails to ensure reliability in critical scenarios. The implementation is available as open source software at [url-after-acceptance](https://github.com/ispcc/ispcc).

I. INTRODUCTION

Trajectory planning, as an essential module in autonomous driving, directly affects driving safety through its real-time capability and accuracy. To enhance the real-time performance, improvements can be pursued across different abstraction layers of the system stack, ranging from hardware to algorithms. At the hardware layer, increasing transistor density to integrate more circuits within the same chip area has traditionally been an effective means to enhance computational performance. However, with the end of Dennard scaling and the approaching limits of Moore’s law [1], this approach is becoming increasingly infeasible. At the software layer, performance can be improved through targeted compiler and runtime optimizations. For example, restructuring memory

layouts to improve cache locality, exploiting SIMD vectorization via compilers such as Intel SPMD Program Compiler (ispc)¹, and overlapping computation with communication through asynchronous execution can reduce latency. At the algorithmic layer, performance can be improved by reducing the search space, guiding sampling toward promising regions, or providing better initial trajectories for optimization to accelerate convergence. In addition, deep learning-based approaches, such as end-to-end learning, attempt to bypass intermediate modules by directly mapping sensor inputs to control actions. In addition to runtime performance, with the increasing functionality and complexity of autonomous driving modules, the issue of software energy consumption also deserves attention. Brown et al. [2] show that around 12.2% of the total energy of an autonomous electric vehicle is consumed by sensor processing and trajectory planning. Reducing computational energy consumption is therefore a research trend that deserves attention, especially considering the limited driving range of AEVs. Since algorithms such as point cloud processing, data fusion, and object recognition in autonomous driving systems are inherently well suited for parallel computation on the GPU, assigning other modules to the CPU in a balanced manner can effectively improve the system-level throughput.

This paper introduces a method to enhance energy- and runtime efficiency by leveraging SIMD techniques on CPUs for trajectory planning, explores the balance between efficiency achieved through reduced-precision and the trade-offs in reli-

¹<https://github.com/ispcc/ispcc>

ability of the trajectory planning, proposes a novel approach to proposes a novel method to reduce precision loss in physical based equations. The experiments isolate and optimize the computation-intensive integration function from the original algorithm. The result shows: The single-precision optimized algorithm achieves up to a $1.9\times$ speedup and reduces energy consumption by 44% compared to the original algorithm; half-precision arithmetic fails to meet the requirements of safety-critical scenarios due to its limited numerical accuracy; nondimensionalization reduces computational error with negligible loss in computational efficiency, enabling the calculation of optimal paths with stringent constraints.

The paper makes the following contributions:

- Exploration of the trade-offs between runtime performance and planning reliability.
- Introduction of a novel method to improve numerical accuracy, based on the nondimensionalization of the motion differential equations.
- Improvement of up to $1.9\times$ in the runtime performance of dynamic-game-based trajectory planning.

II. BACKGROUND

A. SIMD

Single instruction, multiple data (SIMD) is a fundamental parallel computing paradigm widely supported in modern CPUs. Compared to the scalar execution model, where each instruction operates on a single data element, SIMD enables one instruction to be applied simultaneously across multiple data elements. Mainstream CPU architectures, including Arm's NEON and Intel's AVX extensions, provide SIMD extensions that allow efficient utilization of wide vector registers (e.g., 128-, 256-, or 512-bit). Additionally, both NEON and AVX extensions support logical and arithmetic operations across different bit widths. Taking the 128-bit NEON extensions used in the experiment as an example, they support 64-bit, 32-bit, and 16-bit floating-point and integer operations. The number of operands is inversely proportional to the bit-width. This means that reducing the bit-width by a certain multiple can yield a corresponding multiple of parallel processing. However, this involves trade-offs in computational accuracy. Especially for algorithms involving iterative operations, even minor errors can lead to significant loss of computational precision. This topic will be addressed in the experimental section.

Converting traditional programs operating on a single *scalar* value to a corresponding program operating on multiple values in a *vector* is called vectorization. Vectorization requires not only the use of special vector or SIMD data types but also changes to the control flow since all elements in a vector always perform the same operation. Compilers like GCC and LLVM for C-based languages support auto-vectorization, but they often abandon vectorization when complex conditional branches or loop-carried dependencies are present, as safe vectorization cannot be guaranteed. To address this limitation, several approaches emerged to ease the vectorization process: compiler-based standards like OpenMP allow the

programmer to steer vectorization via directives in the form of *pragmas*; language-based approaches like `ispc` [3] or `Sierra` [4] provide language constructs to inform the compiler how to best vectorize code; frameworks like Kokkos [5], Halide [6], and AnyDSL [7] offer programming models that allows programmers to express programs in a more convenient way for vectorization. These tools are typically optimized for specific hardware platforms and application domains, leading to differences in implementation and applicability.

B. Floating-Point Number Representation

The use of an exponent in the IEEE 754 floating-point format enables representation of a wide dynamic range, while the spacing between adjacent representable numbers grows with the magnitude of the value. Due to exponent biasing [8], floating-point numbers are more densely distributed near zero, meaning a large portion of representable numbers are within $[-1, 1]$, where they provide higher precision. Therefore, normalizing function parameters into this range enhances relative higher numerical accuracy. This phenomenon is clearly demonstrated in Fig. 1 (i).

C. Dynamic-Game-Based Trajectory Planner

Game theory is the traditional framework used to model multi-agent environments and define optimal policies within them. The traffic problem is often framed as a dynamic game [9], [10], [11], where the objective is to solve the Generalized Nash Equilibrium Problem (GNEP). In a Generalized Nash Equilibrium (GNE), each player adopts a strategy that is a best response to the strategies of the others, while also satisfying the shared constraints of the game, e.g., collision avoidance. The multi-agent trajectory planner used for this study models the traffic scenario as a GNEP [12], formalized as a set of optimization problems.

Consider a time horizon discretized with N steps, the state and input size are denoted with n_X and n_U respectively. The state and the control input of vehicle i at timestep k are denoted with $\mathbf{x}_k^i = [x, y, \psi, v, l]^T \in \mathbb{R}^{n_X}$; that is, the cartesian coordinates, heading, speed and cost function, and $\mathbf{u}_k^i = [\delta, F]^T \in \mathbb{R}^{n_U}$, that is, the steering angle and longitudinal force.

Consider a GNEP with M players, each player i decides over its control input sequence, denoted with $U^i = [(\mathbf{u}_1^i)^T, \dots, (\mathbf{u}_{N-1}^i)^T]^T \in \mathbb{R}^{n_U(N-1)}$, that generates a trajectory denoted with $X^i = [(\mathbf{x}_1^i)^T, \dots, (\mathbf{x}_{N-1}^i)^T]^T \in \mathbb{R}^{n_X(N-1)}$, according to the dynamic model $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$.

The cost function of player i is denoted by $J^i(X^i, U^i) : \mathbb{R}^{n_U(N-1)} \rightarrow \mathbb{R}$, it depends on the trajectory and on the control input sequence. The goal of each agent is to minimize the cost function without violating the constraints, formally:

$$\begin{aligned} \min_{U^i} \quad & J^i(U^i) \\ \text{s.t.} \quad & C^i(U) < 0 \end{aligned} \quad (1)$$

where $C^i(U) < 0$ are the inequality constraints, that include collision avoidance and lane keep constraints. This set of M

Algorithm 1 Integration Function Algorithm

Input: control signal of every vehicle $U = [U^1, \dots, U^i, \dots, U^M]^T$, initial state of the system $\mathbf{x}_0 = [\mathbf{x}_0^1, \dots, \mathbf{x}_0^M]^T$, time constant of the first order longitudinal dynamics τ , the flexibility K with respect to the input F , the maximum speed v_{max} , the weight parameter of the cost function w and the time step dt .

Output: trajectories $X = [X^1, \dots, X^i, \dots, X^M]^T$.

```
1: for each vehicle  $i = 1 \dots M$  do
2:    $[x, y, \psi, v, l]^T = \mathbf{x}_0^i$ 
3:   for each time step  $k = 1 \dots N$  do
4:      $[\delta, F]^T = \mathbf{u}_k^i = U[i][k]$ 
5:      $\beta = \delta \cdot l_r / (l_f + l_r)$ 
6:      $\dot{x} = v \cdot \cos(\psi + \beta)$ 
7:      $\dot{y} = v \cdot \sin(\psi + \beta)$ 
8:      $\dot{v} = -v/\tau + K \cdot F$ 
9:      $\dot{\psi} = v \cdot \tan(\delta) \cdot \cos(\beta) / (l_f + l_r)$ 
10:     $\dot{l} = w \cdot (v - v_{max})^2$ 
11:     $[x, y, \psi, v, l]^T += [\dot{x}, \dot{y}, \dot{\psi}, \dot{v}, \dot{l}]^T \cdot dt$ 
12:    if  $v < 0$  then
13:       $v = 0$ 
14:    end if
15:     $X[i][k] = \mathbf{x}_k^i = [x, y, \psi, v, l]^T$ 
16:  end for
17: end for
18: return final result
```

problems constitute a GNEP since they are coupled through the collision avoidance constraints $C^i(U) < 0$ that depend on the control inputs of every vehicle $U = [U^1, \dots, U^i, \dots, U^M]^T \in \mathbb{R}^{nU(N-1)M}$. The problem follows a single-shooting approach, where the equality constraints linking the input sequence of vehicle i , U^i , to its trajectory X^i , through the dynamic model are enforced via the integration function $X = F(U, \mathbf{x}_0)$ (1), with $\mathbf{x}_0$ denoting the traffic state at time $t = 0$.

The solution to this set of optimization problems, $U^* = [U^{1*}, \dots, U^{i*}, \dots, U^{M*}]^T \in \mathbb{R}^{nU(N-1)M}$, is an open-loop generalized Nash equilibrium. The control signal U^i of each player i is the best response to the other agents' strategies U^{-i} , given the initial state of the system $\mathbf{x}_0 = [\mathbf{x}_0^1, \dots, \mathbf{x}_0^M]^T$. It is open-loop since the control input sequences U is a function of time, and not function of the state of the system at time step k . The GNEP is solved through a Trust Region solver based on an Augmented Lagrangian formulation [12].

The single-shooting approach makes the real time feasibility highly dependent on the integration function $F(U^i, \mathbf{x}_0^i)$, that is called numerous times, particularly during gradient computation. For this reason, the integration function must be rigorously optimized, making it a natural candidate for techniques targeting runtime performance and energy efficiency.

III. RELATED WORK

At the algorithmic level, optimization techniques, such as heuristic-guided sampling [13], can accelerate trajectory planning by reducing theoretical complexity. However, these

improvements are often limited, sometimes exhibiting only sub-linear gains with respect to problem size, and in certain cases may compromise optimality guarantees [14]. Sampling-based trajectory planning is naturally well suited for execution on GPUs due to their high degree of data parallelism [15], [16]. GPU performance heavily relies on regular control flow and coalesced memory accesses, whereas irregular applications with dynamic branching and irregular memory access patterns suffer from thread divergence and bandwidth inefficiency [17]. In heterogeneous CPU-GPU deployments, compute-intensive yet control-sensitive components (e.g., decision making and control) are often retained on the CPU, while massively data-parallel stages (e.g., perception modules such as data fusion and object detection) are deployed on the GPU [18]. However, the data preprocessing, the data-transfers between host and device and the kernel-launch synchronization can introduce non-negligible overheads that increase end-to-end latency [19]. Acceleration on CPUs typically relies on multi-core parallelism. Multi-core CPUs are more favorable for optimization-based algorithms, as such algorithms often involve iterative functions with complex control flow and loop dependencies. However, the efficiency gains achieved through multi-core parallelism generally exhibit sub-linear speedup with respect to the number of cores, because of the resource sharing, cache coherency, synchronization, load imbalance, and parallelization overhead [20]. Moreover, this resource-driven performance improvement is typically accompanied by a significant increase in energy consumption. Energy consumption often grows faster than performance, leading to a decline in performance-per-watt efficiency [20].

In this work, SIMD vectorization, natively supported by most modern CPUs, is exploited to accelerate the dynamic-game-based trajectory planner [12]. The compiler (`ispc`) (see Section IV-B) is employed, which enables explicit control of vectorization, allows programmers to specify whether a variable is treated as scalar or vectorized across SIMD lanes and generates the vectorized assembly code automatically. Compared to scalar execution, vectorized computations incur a slight increase in power consumption, see the Fig.12 and Fig.13 in [21], but achieve multi-fold runtime speedups.

IV. METHODOLOGY

Modern CPUs widely support vectorized logic and arithmetic operations. In particular, the Arm Cortex-A78AE cores implement 128-bit Advanced SIMD (NEON), which supports arithmetic and logical operations on double-precision, single-precision, and half-precision floating-point data. Reducing the operand bit-width from 64 to 32 or 16 increases the number of parallel elements per vector register. For example, on the Cortex-A78AE, SIMD arithmetic can operate on 2 double-precision, 4 single-precision, or 8 half-precision values per instruction, corresponding to theoretical speedups of $2\times$, $4\times$ and $8\times$, respectively, relative to scalar execution. This approach requires neither algorithmic modifications nor specialized hardware design, and it benefits from high development efficiency through compiler-assisted vectorization via `ispc`.

Richard,
do
I
need
delete
this
ref-
erence?
Oth-
er-
wise
the
re-
view
could
prob-
ably
pre-
dict
who
is
au-
thor
of
this
pa-
per
right?

Because the dynamic-game-based trajectory planning involves iterative state updates across multiple time steps, small precision losses can accumulate and amplify over iterations. To mitigate this effect, nondimensionalization is introduced (see Section IV-A), which standardizes parameter values into the interval $[-1, 1]$ to improve numerical accuracy.

A. Nondimensionalization for Mitigating Precision Loss

Due to the characteristics of floating-point representation (see Section II-B), mapping variables from their natural domain to the standardized interval $[-1, 1]$ helps mitigate precision loss. Formally, it is necessary to define an operator $\mathcal{N}$ such that:

$$\mathcal{N} : \mathcal{D} \rightarrow \mathcal{C} \quad (2)$$

where $\mathcal{D} = [\xi_{\min}, \xi_{\max}] \subset \mathbb{R}$ is the natural domain of the generic variable ξ , used in the integrate function, while $\mathcal{C} := [-1, 1] \subset \mathbb{R}$ is the target codomain of the operator.

The main challenge lies in applying this operator to variables in physically meaningful nonlinear differential equations. To address this, the equations are first reformulated in nondimensional form. Nondimensionalization is a standard technique that simplifies and parameterizes problems involving physical units, particularly in systems governed by differential equations.

Each variable is measured relative to some appropriate units, that refer to quantities intrinsic to the system. These units are a specific length $L = 64 \text{ m}$, speed $V = 64 \text{ m/s}$, mass $M = 1/16 \text{ kg}$, and cost $C = 4096$. Table I shows the standardized variables, variables sharing the same unit of measure are standardized consistently and therefore are not repeated. The nondimensional motion equations become:

$$\begin{aligned} \hat{\beta} &= \hat{\delta} \frac{l_r}{L}, & \hat{x} &= \hat{v} \cos(2\pi\hat{\psi} + 2\pi\hat{\beta}), \\ \hat{v} &= -\frac{\hat{v}}{\hat{\tau}} + \hat{k}\hat{F}, & \hat{y} &= \hat{v} \sin(2\pi\hat{\psi} + 2\pi\hat{\beta}), \\ \hat{l} &= \hat{w}(\hat{v} - \hat{v}_{\max})^2, & \hat{\psi} &= \frac{\hat{v} \tan(2\pi\hat{\delta}) \cos(2\pi\hat{\beta})}{2\pi(\hat{l}_f + \hat{l}_r)}. \end{aligned} \quad (3)$$

These scaled nondimensional equations allow the variables in the integration function to be included inside the interval $[-1, 1]$.

B. Intel Implicit SPMD Program Compiler

ispc provides a C-like syntax; high-quality SIMD code generation; portability across over a decade of CPU generations such as x86 SSE2, SSE4, AVX, AVX2, and AVX512, as well as ARM NEON and recent Intel GPUs. Performance-critical functions can be extracted and compiled as external libraries, which are invoked directly by the original application. In these extracted functions, vectorization behavior can be explicitly controlled using the `varying` and `uniform` keywords. Consider the following simple C++ statement:

```
s_t0[x] += param.dt * ds_t0[x];
```

TABLE I: Nondimensional variables

Variable	Measure	Nondimensionalization
x	$[m]$	$\hat{x} = x/L$
v	$[m/s]$	$\hat{v} = v/V$
ψ	$[rad]$	$\hat{\psi} = \psi/(2\pi)$
dt	$[s]$	$\hat{dt} = dt \cdot V/L$
τ	$[s]$	$\hat{\tau} = \tau \cdot V/L$
F	$[Kg \cdot m/s^2]$	$\hat{F} = F \cdot L/(M \cdot V^2)$
K	$[1/Kg]$	$\hat{K} = K \cdot M$
l	$[cost]$	$\hat{l} = l/C$
w	$[cost \cdot s/m^2]$	$\hat{w} = w \cdot VL/C$
$d\psi/dt$	$[rad/s]$	$\hat{d\psi}/\hat{dt} = (L/2\pi V) \cdot d\psi/dt$
dv/dt	$[m/s^2]$	$\hat{dv}/\hat{dt} = (L/V^2) \cdot dv/dt$
dl/dt	$[cost/s]$	$\hat{dl}/\hat{dt} = (L/VC) \cdot dl/dt$

For ARM, gcc generates the following assembly not utilizing vectorization:

```
fmadd s10, s7, s13, s10
```

ispc generates vectorized assembly code utilizing a vector register of 4 single-precision floats:

```
fmla v0.4s, v1.4s, v2.4s
```

V. CHALLENGES

As explained in subsection II-C, the optimization focuses on the integration function. Thanks to automatic code generation by ispc, much of the manual effort required for writing SIMD intrinsics can be avoided. However, several aspects must be considered, which are discussed in the following subsections.

A. Data Layout

In C/C++, the array of structures (AoS) layout is commonly used, where fields with different attributes are stored contiguously, corresponding to the way the program sequentially processes objects. However, SIMD execution benefits from grouping identical attributes together, since vectorized operations act on homogeneous data elements. Therefore, in our experiments, the AoS layout is transformed into a structure of arrays (SoA) layout to improve data layout and maintain cache locality.

B. Numerical Differentiation

A critical consideration in precision reduction is its effect on numerical differentiation. The trajectory planner is based on a Trust region solver to compute the GNE. The algorithm estimates the gradient of the augmented Lagrangian $\mathcal{L}^i(U)$ for each traffic participant, using the finite difference approximation:

$$\frac{\partial \mathcal{L}^i}{\partial u_j^i}(U) \approx \frac{\mathcal{L}^i(U + h e_j) - \mathcal{L}^i(U)}{h} \quad (4)$$

where e_j is the unit vector and h should approach zero ideally. The choice of the step size h is critical: A large h results in truncation errors, as the perturbation may smooth out local variations of the augmented Lagrangian $\mathcal{L}^i(U)$, leading to inaccurate gradient estimates and potentially skipping the

the loop is missing in the code, without a loop there is nothing to vectorize; this would also allow to show uniform/varying in ispc?

this sentence may be deleted

true descent direction; if h is chosen excessively small, finite precision arithmetic cannot represent the difference accurately. In this regime, round-off error dominates, and the numerical accuracy of floating-point formats becomes insufficient. In the designed scenarios presented in section VI, a step size of $h = 3.3 \times 10^{-4}$ was empirically found to yield consistent gradient estimates, such that both the single-precision and double-precision implementations produced trajectories that satisfies all constraints.

VI. EXPERIMENT

The experiments considered three configurations for trajectory planning: scalar double-precision arithmetic (denoted as FP64), vectorized single-precision arithmetic (FP32_v4), and vectorized half-precision arithmetic (FP16_v8). Among them, the FP64 version serves as the baseline for comparison. To evaluate the reliability of the computed trajectories, three safety-critical scenarios were designed:

- Intersection;
- Dynamic-Avoidance;
- Static-Obstacle.

In the intersection scenario, trajectories are planned for eight vehicles. This number is chosen deliberately, since a 128-bit SIMD register can operate on four single-precision operands, and eight vehicles align well with vectorized execution. The setup is intentionally designed to present a high risk of collision, as each vehicle's path intersects with that of at least one other vehicle. The *Dynamic-Avoidance* scenario evaluates the reliability when encountering oncoming vehicles, while the *Static-Avoidance* scenario examines collision-avoidance of multiple static obstacles.

TABLE II: Costs and Constraint Violations in the Scenarios

	Intersection		Dynamic-Avoidance		Static-Obstacle	
	l	c_v	l	c_v	l	c_v
FP64	119	No	16	No	36	No
FP32_v4	119	No	17	No	28	No
FP16_v8	453	Yes	1024	Yes	5126	Yes

A. Evaluation of Precision Loss

To evaluate the impact of using reduced-precision arithmetic on trajectory planning, three different levels of arithmetic precision are applied to the considered scenarios. The resulting trajectories are assessed using the cost l (introduced in subsection II-C) and the constraint violation, denoted as c_v , which includes collision avoidance, input limits, and lane boundary conditions, to determine whether the precision is sufficient. The results are reported in Table II.

The experiments show that half-precision arithmetic yields significantly higher costs in the all scenarios and violate all constraints. This indicates that half-precision precision is insufficient for accuracy-sensitive trajectory planning. Because around the value of 1.0, numerical rounding errors in half-precision representation become approximately 10^{-3} , leading to unstable optimization results. In contrast, the costs of FP64 and FP32_v4 are comparable, with FP32_v4 even achieving

lower costs in the *Static-Obstacle* scenario. Therefore, single-precision arithmetic meets the safety requirements, and the subsequent experiments focus on FP32_v4.

B. Improvement of Precision via Nondimensionalization

To intuitively evaluate the effect of nondimensionalization on improving numerical precision in trajectory planning, the experiments intentionally reduce the inter-vehicle distance and increase the vehicle speed to induce constraint violations. In the *Intersection* scenario (Fig. 2), nondimensionalization induces only minor path changes overall, but it alters the strategies of the *orange* and *brown* vehicles substantially—especially the *orange* one. With nondimensionalization, the *orange* vehicle slows, yields to the right-moving *blue* vehicle, and then detours on the left to avoid collision, which is safer than accelerating to pass in front of the blue trajectory. The *brown* trajectory with nondimensionalization is also more robust with respect to the distant right-moving *blue* vehicle. Without nondimensionalization, the *purple* and *pink* vehicles violate the collision avoidance constraints at time step 4; with nondimensionalization, a mild leftward steering along the direction of travel removes this contact. In the *Dynamic-Avoidance* scenario (see Fig. 1 (ii) and (iii)), the *orange* vehicle touches the lane boundary (a constraint violation), whereas the version with nondimensionalization keeps the vehicle well in the lane. In the *Static-Obstacle* scenario, nondimensionalization eliminates the impending collision between the *orange* vehicle and the *blue* obstacle (Fig. 3).

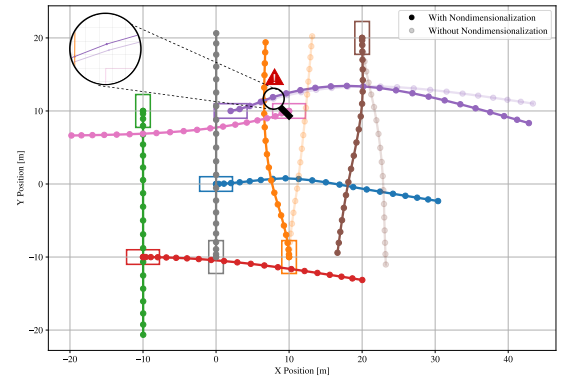


Fig. 2: Resulting trajectories in the intersection scenario.

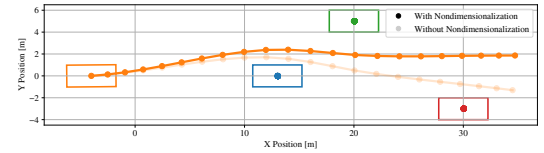


Fig. 3: Resulting trajectories in the static obstacle avoidance scenario.

C. Runtime Improvement and Energy Consumption

The *Intersection* scenario is used to analyze the relationship between the runtime performance and the utilization of the SIMD lanes by gradually reducing the number of vehicles, as shown in Fig. 4. All experiments are conducted using single-core execution on the NVIDIA Jetson Orin. The results show that only when the number of vehicles exceeds one, the runtime of the FP32_v4 version becomes lower than that of the scalar FP64 version, indicating that except for the single-vehicle case, vectorization introduces no additional computational cost. When the number of vehicles reaches four, the runtime improvement becomes substantial. When the number of vehicles reaches eight, the vectorized version achieves a speedup of up to $1.94\times$. For intersection scenarios with many vehicles in urban, the proposed vectorized method provides substantial performance benefits compared to the scalar baseline. It is worth noting that the runtime performance can be further improved, for example, by exploiting multi-core parallelism or by reducing the maximum number of iterations allowed for the Trust Region solver. In the experiments, this parameter is set to 100. Based on empirical observations, such a high value is rarely required in practice. The purpose of this study is not to optimize this parameter, but rather to set it conservatively high to ensure that the solver can reliably converge to the optimal trajectory.

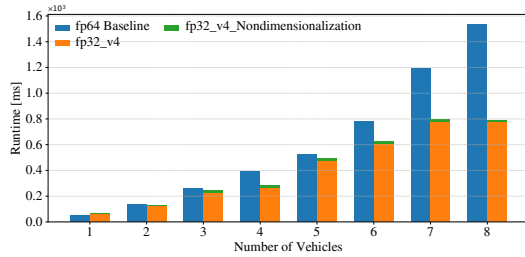


Fig. 4: Predicted trajectories for the *Static_Obstacle* scenario with single-precision arithmetic with (bottom) and without (top) nondimensionalization.

Using the INA3221 power monitor integrated on the Jetson Orin board, power consumption was measured every 10 ms. Before executing the application, a 10 s idle phase was recorded to determine the system’s baseline power. In the experiment, the average energy consumption was measured by running the *Intersection* scenario with eight vehicles for 100 calls. The results show that the baseline FP64 version consumed 0.97 J, whereas the FP32_v4 version consumed only 0.54 J, corresponding to 56% of the baseline.

VII. CONCLUSION

This paper investigated the impact of the vectorization and the arithmetic precision on the runtime and energy efficiency of the trajectory planning for autonomous vehicles. By comparing the scalar FP64 and the vectorized FP32_v4 implementations, the results demonstrate that the proposed nondimensionalization and vectorization strategies significantly re-

duce computation time while maintaining numerical precision and safety. When the number of vehicles reaches four, the runtime improvement becomes substantial, whereas in the single-vehicle case, the vectorized version performs slightly slower due to underutilized SIMD lanes. In the most complex scenario, the FP32_v4 version achieved up to $1.9\times$ speedup and consumed only 56% of the energy of the baseline. These findings highlight the potential of mixed-precision and SIMD-based optimization for real-time trajectory planning on embedded platforms. Future work can extend this approach by exploring arithmetic precisions between single and half precision on hardware-supported platforms, such as FPGAs, to further enhance computational and energy efficiency.

REFERENCES

- [1] H. Sutter, “The free lunch is over: A fundamental turn toward concurrency in software,” *Dr. Dobbs’s Journal*, vol. 30, no. 3, pp. 202–210, 2005.
- [2] N. E. Brown et al., “Evaluation of autonomous vehicle sensing and compute load on a chassis dynamometer,” in *ITSC*, 2023, pp. 1989–1995.
- [3] M. Pharr et al., “Ispc: A spmd compiler for high-performance cpu programming,” in *2012 Innovative Parallel Computing (InPar)*, 2012, pp. 1–13.
- [4] R. Leißa et al., “Sierra: A simd extension for c++,” in *Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing*, ser. WPMVP ’14, ACM, 2014, pp. 17–24.
- [5] D. Sahasrabudhe et al., “A portable simd primitive using kokkos for heterogeneous architectures,” in *Accelerator Programming Using Directives*, Springer International Publishing, 2020, pp. 140–163.
- [6] J. Ragan-Kelley et al., “Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines,” in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI ’13, ACM, 2013, pp. 519–530.
- [7] R. Leißa et al., “Anydsl: A partial evaluation framework for programming high-performance libraries,” *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018.
- [8] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, Mar. 1991.
- [9] S. Le Cleac’h et al., “Algames: A fast solver for constrained dynamic games,” in *Robotics: Science and Systems XVI*, ser. RSS2020, Robotics: Science and Systems Foundation, Jul. 2020.
- [10] W. Schwarting et al., “Social behavior for autonomous vehicles,” *Proceedings of the National Academy of Sciences*, vol. 116, no. 50, pp. 24 972–24 978, 2019.
- [11] G. Lucente et al., “A bayesian approach with prior mixed strategy nash equilibrium for vehicle intention prediction,” *Automotive Innovation*, vol. 6, no. 3, pp. 425–437, Aug. 2023.

- [12] G. Lucente et al., “Deepgame-tp: Integrating dynamic game theory and deep learning for trajectory planning,” *IEEE OJ-ITS*, vol. 5, pp. 873–888, 2024.
- [13] J. D. Gammell et al., “Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” in *IEEE/RSJ IROS*, 2014, pp. 2997–3004.
- [14] J. Luo et al., “An empirical study of optimal motion planning,” in *IEEE/RSJ IROS*, 2014, pp. 1761–1768.
- [15] S. Heinrich et al., “Real-time trajectory optimization under motion uncertainty using a gpu,” in *IEEE/RSJ IROS*, 2015, pp. 3572–3577.
- [16] E. S. Morales et al., “Parallel multi-hypothesis algorithm for criticality estimation in traffic and collision avoidance,” in *IEEE IV*, 2019, pp. 2164–2171.
- [17] M. Burtscher et al., “A quantitative study of irregular programs on gpus,” in *IEEE HSWC*, 2012, pp. 141–151.
- [18] S. Liu et al., “Computer architectures for autonomous driving,” *Computer*, vol. 50, no. 8, pp. 18–25, 2017.
- [19] D. Lustig et al., “Reducing gpu offload latency via fine-grained cpu-gpu synchronization,” in *IEEE HPCA*, 2013, pp. 354–365.
- [20] I. Calero et al., “Power, energy, and performance analysis of single- and multi-threaded applications in the arm thunderx2,” *Journal of Parallel and Distributed Computing*, vol. 204, p. 105 118, 2025.
- [21] T. Jakobs et al., “Performance and energy consumption of the simd gram–schmidt process for vector orthogonalization,” *The Journal of Supercomputing*, vol. 76, no. 3, pp. 1999–2021, 2020.