面向模型预测路径积分控制的领域专用硬件加速

Erwan Tanguy-Legac、Tommaso Belvedere、Gianluca Corsini、Marco Tognon、
Marcello Traiola 雷恩大学、Inria、法国国家科学研究中心、IRISA、UMR6074

arXiv:2601.12089v1 [cs.AR] 17 Jan 2026

摘要—**在实时环境下精确控制机器人系统是一项具有挑战性的问题。为此，机器人学界采用了多种算法，例如模型预测控制（MPC）和模型预测路径积分（MPPI）控制。前者难以应用于无人机等非线性系统，而后者则需要巨大的计算负载。图形处理器已被成功用于加速 MPPI 实现；然而，对于自主或无人目标，尤其是采用电池供电的目标而言，其功耗通常过高。另一方面，人们提出了定制设计，通常在现场可编程门阵列上实现，以加速机器人算法，同时相比基于图形处理器（或中央处理器）的实现显著降低能量消耗。然而，迄今尚未提出 MPPI 定制加速器。在本文中，我们提出了一种用于 MPPI 控制的硬件加速器，并对其执行过程进行了仿真。结果表明，与基于图形处理器的 MPPI 实现相比，MPPI 定制加速器能够生成精度更高的轨迹。**

I. 引言

现代机器人应用要求能够实时执行复杂任务，并对环境做出快速、准确的响应。得益于现代硬件计算能力的提升，模型预测控制（MPC）等技术已从以往主要用于大型工业设备，逐渐应用于各种机器人平台 [1], [2]。MPC 的工作方式是在滚动时域内为系统选择最佳轨迹。具体而言，它通过求解一个优化问题，在满足系统约束的条件下最小化代价函数。

另一种在机器人学领域逐渐流行的算法是模型预测路径积分（MPPI）控制 [3]。该算法可视为 MPC 的一种基于采样的版本：它通过在滚动时域内对多条轨迹进行采样并计算它们的加权平均，来求解优化问题。

尽管这些算法已成功应用于各种平台，但其计算量十分密集，需要高性能硬件。这对于非线性系统尤为如此，例如无人飞行器（UAV），其底层优化问题是非凸的，求解成本高昂，因此难以采用局部方法。当然，必须考虑这类计算所需的计算能力，尤其是对于便携式自主系统。因此，相关领域已开始将模型预测控制算法移植到高性能专用硬件上，例如图形处理器 [4] 和现场可编程门阵列 [5], ，二者都非常适合并行计算。

现场可编程门阵列已用于在嵌入式系统上实现 MPC，功耗低于 1 瓦 [6], ，也应用于许多其他机器人领域 [7],[8]。与中央处理器和图形处理器相比，它们在视觉内核计算 [9], 、图像处理 [10], 和网络安全 [11] 等多个领域能够显著降低功耗并提升整体性能。然而，据我们所知，目前尚无在专用硬件加速器（图形处理器除外）上实现 MPPI 的方案；本文将对此展开研究。

本文其余部分安排如下：首先，第二节概述模型预测路径积分（MPPI）控制算法。然后，在第三节中，我们介绍所设计的硬件加速器，该加速器利用专用集成电路的能力来加速 MPPI 算法。最后，我们给出实验结果，并讨论设计的局限性。

II. 模型预测路径积分控制

符号说明： 在本文中， H 表示时间范围的长度，N 表示采样轨迹的数量， x_t 表示系统在时间 t的状态， u_t 表示将在时间 t施加于系统的输入。此外，我们使用两个不同的代价函数：$\mathcal{L}$ 表示运行代价（针对轨迹的每个中间状态计算）， ϕ表示终端代价（针对轨迹的

最后一个状态计算）。最后，对于每个 $i \in [\![0, N-1]\!]$，上标 i 用于表示第 i^{th}条采样轨迹。

在机器人学中，实时轨迹优化与控制算法（例如模型预测控制和 MPPI）旨在计算控制输入，使机器人在遵守系统动力学和约束条件的同时朝目标移动，并优化性能代价（例如能量、平滑度、到目标的距离和避障）。

如前所述，MPPI 是一种基于采样的算法，可视为模型预测控制的一种变体。模型预测控制通过求解带约束的优化问题来计算最优轨迹，而 MPPI 则采用基于采样的方法。这种方法尤其适用于非线性系统，因为它无需梯度即可进行优化。如 [12], 中所述，它还支持使用不连续代价函数，并能自然地处理系统中的噪声。

为此，MPPI 可分为三个不同的步骤：首先，我们生成 N 个由 H 组成的输入序列，这些输入从以当前输入为中心的正态分布中随机采样。然后，对于每个序列，我们预测应用这些输入后系统的轨迹，并计算其代价。最后，我们对输入进行加权平均，以此近似得到最优轨迹，其中权重由相应轨迹的代价决定。随后，将该轨迹的第一个输入施加到系统的执行器上，并丢弃序列中的其余部分，以便开始新的计算。

这三个步骤将在本节其余部分进行简要介绍。其中，计算量最大的一步是展开，它会在整个时间范围内模拟系统的动态过程。我们的工作重点正是加速这一特定步骤（见第 III 节）。第一步和最后一步仍由中央处理器执行。

算法1展示了 MPPI 算法的伪代码。其三个核心步骤如下所述。

a) 输入采样（第 *1* 行）：算法首先从以当前输入 v 为中心、标准差为 σ 的多元正态分布中采样 N 个由 H 个输入组成的序列。

b) 展开（第 *2–10* 行）：将上一步采样的输入用于预测由此产生的轨迹及其代价，这一过程称为展开。这一步天然适合并行处理，因为每条轨迹彼此独立。为此，展开 依赖于系统模型，用它对系统动力学随时间进行离散积分。换句话说，在时间范围内的每个时间步，我们都会根据当前状态和待应用的输入来计算系统的下一个状态（第 6 行）。相应轨迹的代价也在每个时间步计算，并根据参考值（由一个状态和一个输入组成，见第 7 行）为其赋予代价值。为了突出轨迹的结果，我们会在展开结束时额外计算终端代价，以强调轨迹的最终状态（第 9 行）。随后将这些代价相加，得到每个输入序列 u^i 的终端代价 J^i。

c) 计算待施加的输入（第 *11–17* 行）：待施加的输入通过对展开过程中使用的输入序列进行加权平均来计算。我们根据每个输入序列 u^i 所产生轨迹的相关代价 J^i，为其分配权重 w^i。

$$u^* = \frac{\sum_{i=0}^{N-1} w^i u_0^i}{\sum_{i=0}^{N-1} w^i}$$

$$w^i = \exp\left(-h\frac{J^i - J_\top}{J_\perp - J_\top}\right)$$

其中，$J_\top$ =最大值$\{J^i\}$，$J_\perp$ =最小值$\{J^i\}$，而 $h > 0$ 是需要设定的参数。

最后，应用 $u^* = v'/d$，然后即可重新开始整个过程。

算法1 MPPI 算法

输入： 当前状态 x_0，参考值 R，当前输入 v

1: $u \leftarrow$ 从 $\mathcal{N}(v, \sigma)$ 中采样 $N \times H$ 输入
2: **对于** i，从 0 到 $N-1$ **执行**
3: $x_0^i \leftarrow x_0$
4: $c^i \leftarrow 0$
5: **对于** t，从0 到 $H-1$ **执行**
6: $x_{t+1}^i \leftarrow F(x_t^i,\ u_t^i)$
7: $c^i \leftarrow c^i + \mathcal{L}(x_t^i, u_t^i, R_t)$
8: **结束循环**
9: $J^i \leftarrow c^i + \phi(x_{H-1}^i, u_{H-1}^i, R_{H-1})$
10: **结束循环**
11: $J_\perp, J_\top \leftarrow$最小值$\{J^i\}$,最大值$\{J^i\}$
12: $v', d \leftarrow 0, 0$
13: **对于** i，从 0 到 $N-1$ **执行**
14: $v' \leftarrow v' + w^i u_0^i$
15: $d \leftarrow d + w^i$
16: **结束循环**
17: **返回** v'/d

鉴于算法中计算需求最高的步骤是展开，因此这是我们最感兴趣的加速对象

。正如引言中所述，目前通常使用图形处理器来加速这一过程。然而，图形处理器与传统计算架构（例如中央处理器）非常相似，只是在并行任务方面效率更高。因此，它们存在局限性，因为无法利用操作的内部结构。图形处理器非常擅长并行化（即 可以并行运行多个滚动实例），但无法利用 MPPI 在整个时间范围内固有的流水线特性。不过，定制加速器可以解决这一问题，它能在图形处理器已有并行能力的基础上实现流水线处理。

III. 面向 MPPI 的专用硬件加速器

MPPI 非常适合并行化，因为每次展开彼此独立。因此，图形处理器通常用于执行 MPPI 所需的大部分计算，因为它们在处理并行任务时性能出色 [13]。通过实例化多个计算单元，每个单元计算一次展开，即可在专用硬件加速器上实现同样的并行处理。

专用硬件加速器提供了比图形处理器更多的加速机会。接下来的两个小节将深入探讨该算法，介绍我们在设计中实现的加速机会。图1展示了加速器的最终架构。

A. 对整个展开进行流水线处理

展开计算主要在一个循环中进行，每次迭代只需要上一次迭代的结果（以及执行期间保持不变的一些外部数据）。得益于这一特性，我们可以展开循环，并通过流水线实现。流水线循环无需等到整个循环执行完毕后才开始下一次循环。第一次迭代完成后，数据即可移至流水线的下一个阶段，同时第一个阶段可以接收新的输入。

举例来说，我们考虑一个包含三个步骤（$H = 3$）的短时域，以及一个短输入序列的展开 （α_0，β_0，γ_0）。在传统的（即非流水线处理）展开计算方式中，每个输入都需要经过 3 次迭代，才能得到最终的展开结果。要计算由 3 个输入组成的输入序列的完整展开序列，需要 9 次迭代。为展示流水线处理的优势，图 2 给出了一个简单的三级流水线示例，其中每一级都执行操作 $x_i \mapsto x_{i+1}$。第一次迭代从加载开始 α_0 到流水线的第一级。在第二次迭代时，第一级已完成计算，结果α_1 由第二级处理，而第一级可以接收新的输入 β_0。按照这种方式，经过 5 次迭代后，整个输入序列便处理完毕，展开也随之完成。对于包含 k 个输入的序列，非流水线版本需要迭代 $3k$ 次，而流水线版本需要 $k+2$次（第一个结果在 3 个周期后出现，此后每个周期产生一个结果）。一般而言，对于时域 H，流水线需要迭代$k + H - 1$ 次，而非流水线版本需要迭代 $H \cdot k$次。对于较长的输入序列 k，流水线版本的加速比为 $\lim_{k\to\infty} \frac{H\cdot k}{k+H-1} \approx H$（即图中示例的 $\frac{3\cdot k}{k+3-1} \approx 3$）。

我们通过对滚动展开循环进行流水线处理，将这一概念应用于设计中。最终设计采用 N 条流水线，每条有 $H+1$个阶段，其中终端代价计算在一个独立的流水线阶段中进行。

总而言之，流水线的主要优势在于：当系统前一状态仍在计算时，我们就可以计算其下一状态。这样可以提高吞吐量，因为在处理前一批输入的同时，系统可以接收新的输入。不过，循环迭代的总次数保持不变，因此与图形处理器相比，我们的加速器所需的延迟变化不大。

B. 加速单个流水线阶段

我们的架构现在由一系列流水线阶段组成，每个阶段计算一次滚动展开迭代。此外，误差计算与下一状态的计算彼此独立，可以并行化，如图1所示。

从系统角度来看，参考值 R 和当前输入 v 由主机 CPU 传输至加速器，而系统的当前状态 x_0 则从传感器读取。每条流水线都会计算轨迹$[x_0, F(x_0, u_0^i)\ \ F(F(x_0,\ u_0^i)\ \ u_1^i)\ \ \ldots]$的代价，并将其发送回中央处理器。因此，中央处理器负责计算最终轨迹。

IV. 实验评估

本节介绍使用我们提出的加速器架构获得的仿真结果，以及综合结果。

A. 仿真

如前所述，我们加速器的流水线架构使我们能够提高系统的吞吐量。为了展示所提出的 MPPI

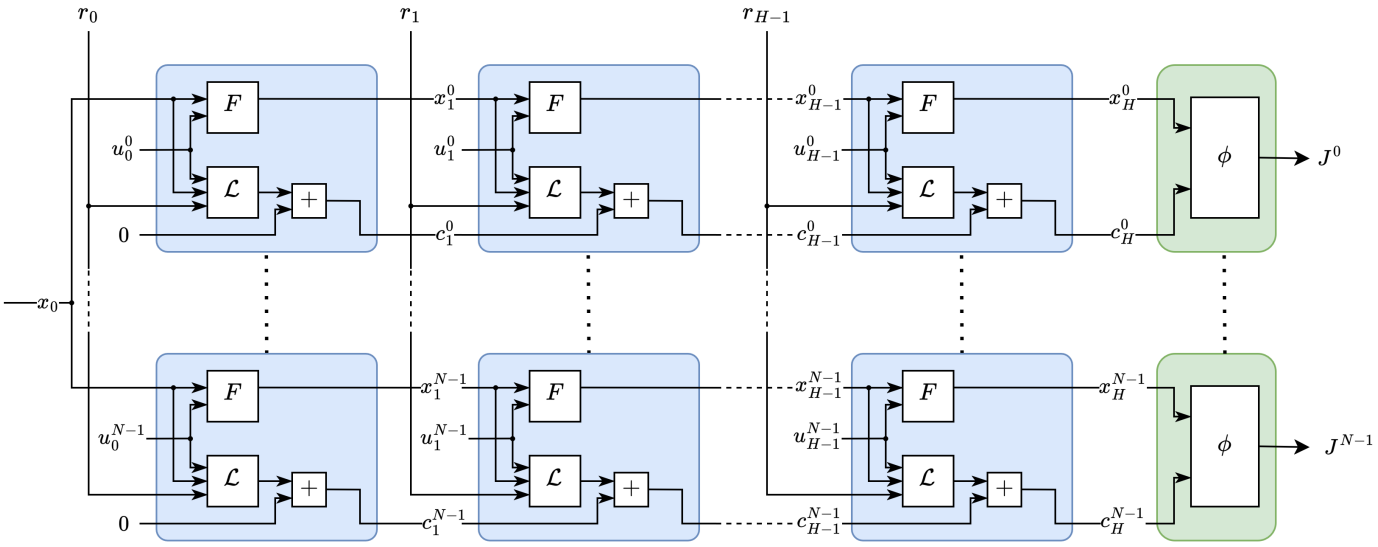

图1：所提出的加速器架构。

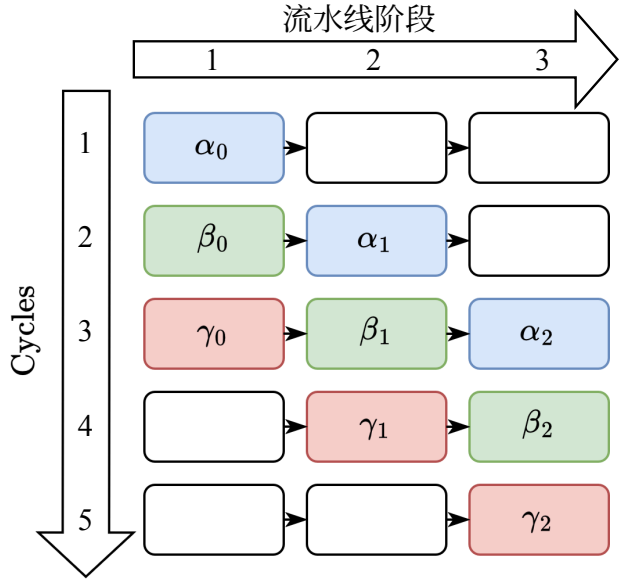

图2：将三级流水线应用于序列（α_0，β_0，γ_0）的示例

针对无人飞行器的定制流水线加速度，我们使用所提出的 MPPI 加速度实现对四旋翼飞行器的动力学进行仿真。第一个仿真任务是到达目标位置（x，y，z）$=(0.5, 0.5, 0.5)$。为了运行这些仿真，我们使用四旋翼飞行器的模型 F；给定一个状态和一组输入，该模型会生成一个新状态。一个状态由位置、旋转、加速度和角速度组成，输入是要施加在各个轴上的力矩，以及一个全局推力值。随后，每次仿真运行 10 秒，采用由 25 个间隔组成的滚动时域，每个间隔为 0.02 秒（即，滚动预测可提前 0.5 秒预测轨迹）。图3展示了两种情况下得到的轨迹。

基于图形处理器的实现运行 $N = 2000$ 个并行滚动预测，而硬件加速实现则限于 $N = 200$ 个。这样做是为了强调专用加速器的硬件空间往往有限。不过，由于我们的加速器采用流水线，因此吞吐量要高得多。

图4展示了本节开头所述简单仿真的结果。值得注意的是，尽管流水线版本的滚动计算步骤并行执行次数减少了 10 倍（因此，采样的轨迹也减少了 10 倍），但它生成的轨迹更加平滑，并且能更好地稳定在目标位置。不过，这是通过对系统执行器进行更精细的控制实现的，而这并非总是可行的。这一点在输入值图中尤为明显：使用流水线版本时，输入值的变化幅度大得多。

我们运行的第二个仿真场景包含一个简单的静态障碍物，它位于第一次仿真所定义的轨迹上。图 5 展示了此次仿真的结果：基于图形处理器的 MPPI 实现未能找到一条合适轨迹，使无人机到达目标；而我们提出的 MPPI 定制加速度实现则成功找到了这样的轨迹。

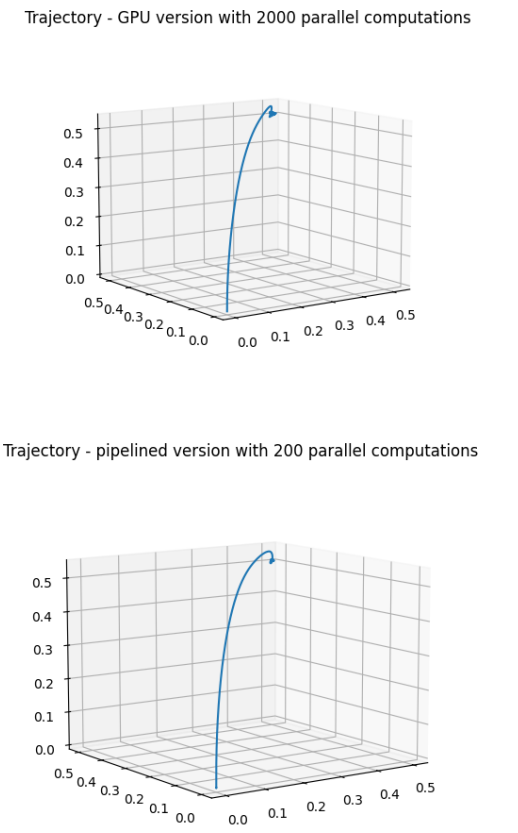

图3：无障碍物仿真。位置以三维空间绘制。上方的仿真系统包含一个图形处理器，执行 2000 次并行滚动预测；下方则仿真了一个配备 200 条并行流水线的现场可编程门阵列的系统。

仿真结果表明，假设加速器上单次滚动展开迭代的计算时间最多比图形处理器上慢十倍（计入数据传输），且能够合成 200 个并行流水线实例，那么采用流水线架构可以显著改善我们系统的轨迹表现。然而，对于四旋翼飞行器这类复杂动态系统，仿真无法完全取代真实世界测试。因此，在开展此类测试之前，必须针对实际目标硬件平台合成设计。

B. 设计综合

硬件综合有两种选择：可重新编程的电路（例如现场可编程门阵列），可用于快速原型设计和测试新设计；以及专用集成电路，它们的

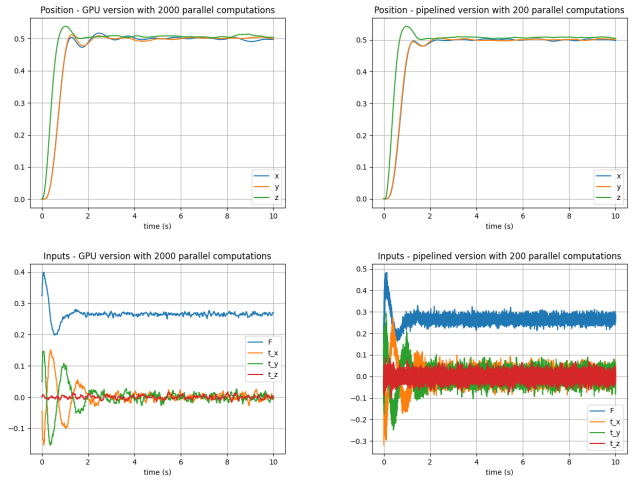

图4：无障碍物仿真。左侧的仿真系统包含一个图形处理器，执行 2000 次并行滚动预测；右侧则仿真了一个配备 200 条并行流水线的现场可编程门阵列的系统。第一行绘制了位置随时间的变化，第二行展示了输入（即发送给执行器的指令），同样以时间为横轴绘制。

效率更高，但需要更长、成本更高的设计周期。

图6展示了所提出系统的高层视图。该系统由一台主机（中央处理器或微控制器）组成，主机将 MPPI 算法中展开部分的计算卸载到硬件加速器，并通过传感器和执行器与环境交互。

我们的实现采用 *VHDL* 和*Verilog*编写。我们使用 *Vivado 2023.1* 在一块 *Alveo U55C* 数据中心 *FPGA*上进行了综合。值得注意的是，我们的设计对数值采用 32 位浮点表示，尽管 *U55C* 并未配备浮点运算单元。这可以解释表 I 中所示的 LUT 使用量为何较高。

由于单个流水线实例的资源使用量很高（12.58 %），我们决定引入一种新的权衡方案：不再让每个流水线级只计算一次滚动展开迭代，而是让每个流水线级计算多次滚动展开迭代。在表 I 中，我们将每个流水线级执行的迭代次数称为该级的大小。这样一来，我们便能减少流水线级的总数，同时保持轨迹生成的时间范围不变。这一调整只会给这些级带来少量开销。不过，在

流水线数量	每条流水线的阶段数	阶段大小	利用率（LUT 占比）
1	25	1	12.58%
5	25	1	83。28%
2	5	5	66.30%
10	1	25	69.22%
14	1	25	96。56%

表 I：使用*Alveo U55C* 数据中心 *FPGA* 和*Vivado 2023.1*默认综合设置的利用率结果。

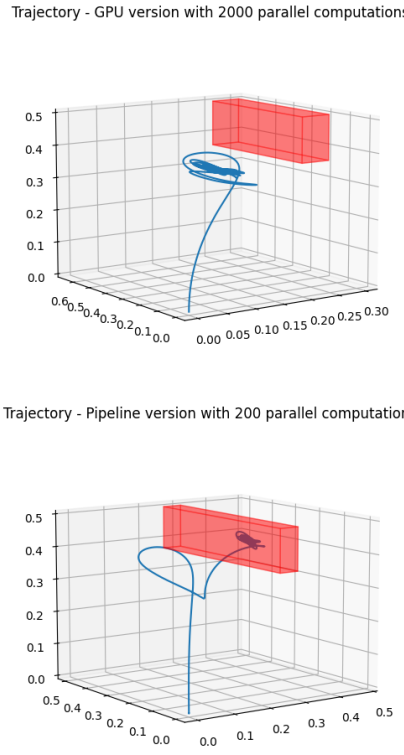

图5：存在障碍物时的仿真。位置绘制于三维空间中；红色方框为障碍物。上方的仿真系统包含一台执行 2000 次并行滚动预测的图形处理器；下方则将系统模拟为配备 200 条并行流水线的现场可编程门阵列。

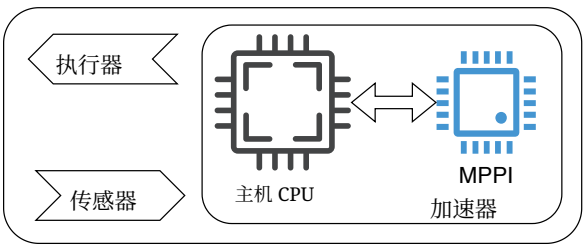

图6：系统的高层视图。

针对专用集成电路时，这种权衡可能并非必要，因为专用集成电路的查找表数量并不固定。专用集成电路能够实现更大的设计，但代价是失去可重新编程特性，因此适合在设计开发流程的最后阶段部署。此外，专用集成电路的专用特性还能使设计更加高效，进一步降低功耗 [14]。

V. 结论

总之，本文展示了使用专用硬件加速器加速 MPPI 算法的可行性。结果表明，这种加速器能够在系统中发现更加平滑的轨迹。专用硬件加速器的声誉也表明，使用它们应当能够降低算法的功耗。

未来的工作可以提出更优的硬件实现方案，进一步降低资源使用量和功耗。这项工作也为修改 MPPI 算法提供了可能，使其能够更好地适应硬件所提供的加速机会。最后，将加速器扩展到整个算法，而不仅仅是 展开步骤，也将是有趣的研究方向。

参考文献

[1] H. Nguyen、M. Kamel、K. Alexis 和 R. Siegwart，《微型飞行器的模型预测控制：综述》，载于*2021*年欧洲控制会议（*ECC*）。IEEE，2021年，第1556–1563页。[2] S. Katayama、M. Murooka 和 Y. T. and，《足式机器人和人形机器人的模型预测控制：模型与算法》，*Advanced Robotics*，第37卷，第5期，第298–315页，2023年。[3] G. Williams、A. Aldrich 和 E. A. Theodorou，《模型预测路径积分控制：从理论到并行计算》，第40卷，第2期，第344–357页。[4] K. M. Abughalieh 和 S. G. Alawneh，《模型预测控制并行实现综述》，第7卷，第34 348–34 360页。[5] A. Joos 和 W. Fichter，《基于现场可编程门阵列的机载飞行计算机上约束非线性模型预测控制器的并行实现》，载于*Advances in Aerospace Guidance, Navigationand Control*，第273–286页。

[6] S. Lucia、D. Navarro、O. Lucia、P. Zometa 和 R. Findeisen，《使用高级综合工具为嵌入式系统优化模型预测控制的现场可编程门阵列实现》，*IEEE*工业信息学汇刊，第14卷，第1期，第137–145页，2018年。[7] Z. Wan、B. Yu、T. Y. Li、J. Tang、Y. Zhu、Y. Wang、A. Raychowdhury 和 S. Liu，《基于现场可编程门阵列的机器人计算综述》，*IEEE*电路与系统杂志，第21卷，74，2021 2，第48页。[8] Y. Wu、L. Luo、S. Yin、M. Yu、F. Qiao、H. Huang、X. Shi、Q. Wei 和 X. Liu，《适用于动态环境中移动机器人的节能型DS-SLAM现场可编程门阵列加速器》，《应用科学》，第11卷，第4期，2021年。[9] M. Qasaimeh、K. Denolf、J. Lo、K. Vissers、J. Zambreno 和 P. H. Jones，《比较视觉内核的中央处理器、图形处理器和现场可编程门阵列实现的能效》，载于*2019*年*IEEE*嵌入式软件与系统国际会议（*ICESS*），2019年，第1–8页。[10] F. Siddiqui、S. Amiri、U. I. Minhas、T. Deng、R. Woods、K. Rafferty 和 D. Crookes，《用于图像处理应用的基于现场可编程门阵列的处理器加速》，《成像学杂志》，第5卷，第1期，2019年。[11] T. Hayajneh、S. Ullah、B. J. Mohd 和 K. S. Balagani，《采用现场可编程门阵列实现的多媒体应用增强型无线局域网安全系统》，*IEEE*系统期刊，第11卷，第4期，第2536–2545页，2017年。[12] G. Williams、P. Drews、B. Goldfain、J. M. Rehg 和 E. A. Theodorou，《信息论模型预测控制：理论及其在自动驾驶中的应用》，第34卷，第6期，第1603–1622页。[13] G. Turrisi、V. Modugno、L. Amatucci、D. Kanoulas 和 C. Semini，《基于GPU采样的随机预测控制器对足式运动的益处》，载于*2024*年*IEEE/RSJ*智能机器人与系统国际会议（*IROS*），2024年，第13 757–13 764页。[14] A. Amara、F. Amiel 和 T. Ea，《低功耗应用中的现场可编程门阵列与专用集成电路之比较》，《微电子学杂志》，第37卷，第8期，第669–677页，2006年。