基于 GPU 的大规模初始条件集合高精度轨道传播：通过皮卡-切比雪夫增广实现

Alessandro Masat * 和 Camilla Colombo†米兰理工大学，意大利米兰，*20156*

ArnaudBoutonnet ‡欧洲航天局（欧洲航天局欧洲航天运行中心），德国达姆施塔特，*64293*

arXiv:2301.03989v1 [cs.DC] 10 Jan 2023

在高精度要求下对大规模初始条件集合进行轨道传播，目前是航天任务开发中的一个瓶颈，例如在行星保护合规性分析中。所提出的方法可通过高效的皮卡-切比雪夫（PC）数值模拟，将任意力源纳入动力学模型。本文提出一种两级积分方案增广方法，使同一算法调用能够运行任意数量的模拟，从而充分利用高性能计算和 GPU（图形处理器）计算资源。文中展示了采用 C语言 和英伟达®CUDA® 编程语言实现后的性能，测试案例取自对类似太阳轨道器任务与金星首次共振相位的优化。

I. 引言

复杂的轨迹方案已成为行星际任务的标准选择，通常会采用多次引力辅助机动，以较低的燃料消耗抵达遥远的太空区域。近期的两个任务——欧洲航天局/美国国家航空航天局的太阳轨道器任务 [1] 和欧洲航天局的木星冰月探测器任务 [2], ——便是正在执行的典型案例。前者先进行地球和金星的准备性飞掠，随后利用与金星的多次共振交会提高航天器倾角，使其越过黄道，从而观测太阳的极区。后者需要多次飞掠才能抵达木星，之后又反复飞掠木星的不同卫星，以实现科学观测目标。若没有设计好的所有飞掠，这两个任务在实际中都难以实现，因为每节省一千克燃料，就意味着可以搭载更多科学设备。

设计高能量多次飞掠轨迹时，通常采用拼接圆锥曲线近似框架。该模型简单而有效，可用于一致地开展初步任务分析。确定可行的引力辅助机动后，便可优化初始轨迹猜测：通过细化初步轨道参数和／或添加修正机动，使其满足运行要求并符合完整的天体

*博士研究生，航空航天科学与技术系，Via G. La Masa 34，20156，意大利米兰，alessandro.masat@polimi.it
†副教授，航空航天科学与技术系，Via G. La Masa 34，20156，意大利米兰，camilla.colombo@polimi.it
‡高级任务分析师，HSO-GFA，欧洲航天局欧洲航天运行中心，Robert-Bosch-Str. 5，D-64293 达姆施塔特，德国，arnaud.boutonnet@esa.int

动力学。

然而，现实环境中存在更复杂的物理现象，其扰动效应可能对名义任务造成重大影响。反复飞掠也意味着处置物体与所飞掠行星发生碰撞的风险更高，可能影响任务是否符合行星保护要求。COSPAR [3] 制定了全球行星保护政策，例如，ESA要求在合规性评估中至少使用N体牛顿引力模型 [4]。迄今为止，这一精度要求使分析局限于基于蒙特卡罗的技术 [5–9]，并采用笛卡尔运动方程；此类模拟本身计算负担沉重，可能拖延任务开发进度。近期的研究进展 [10] 通过实现动力学运动的Kustaanheimo-Stiefel [11]公式，将所需计算时间几乎缩短了一半。

本研究受到 [12, 13], 中研究成果的启发。该研究旨在优化一段多次飞越轨迹弧段，同时利用N体相对论环境的自然动力学。尽管概念上较为复杂，B平面与皮卡-切比雪夫（PC）方法相结合，仍构成了总体上计算可行的算法；Matlab® 串行实现仅用几分钟便收敛到最优轨迹。其高效性的核心在于PC数值格式的不动点特性，这最大限度地减少了读取和插值星历数据的需求。测试案例将在第VI.A节中详细介绍。优化过程生成的轨迹集用于验证本研究提出的概念，具体而言，是验证采用修改版PC方法进行并行传播的效果。

过去几年中，修正PC方法在公式推导和实现方面不断发展，并提出了其在地球轨道中的潜在应用；该方法有助于提高数值分析的效率。Junkins 等人 [14]分析了该方法的性能，并将其效率与Runge-Kutta-Nystrom 12(10)积分器进行了比较，同时提出了一个二阶版本。随后，Koblick 和 Shankar[15] 使用 NASA 的 Java天体动力学工具包，通过测试不同的力模型，将分析扩展至高精度轨道传播。Woollands 等人 [16–18] 将该方法用作数值积分器，以求解 Lambert两点边值问题；他们还评估了采用动力学 Kuustanheimo-Stiefel表述的优势，并提出了多圈轨迹设计的解法。Swenson 等人 [19]采用微分修正方法，将修正PC方法应用于圆型限制性三体问题。Singh 等人 [20] 将该方法用作数值积分方案，对准冻结、近极地和低高度月球轨道进行了可行性研究，其中纳入了N体效应和球谐摄动。Koblick 等人 [21]利用该方法的不动点特性，将低推力轨迹设计表述为最优控制问题，对控制冲量进行离散化，并纳入了地球扁率 J_2 摄动。Macomber 等人[22] 引入了该方法的冷启动、温启动和热启动概念，探讨了通过改进初始条件以及利用算法不动点特性的可变精度力模型来提高效率的可能性。Woollands 等人 [23]将最优低推力设计扩展至高保真模型，以描述非球形

地球，并在摄动加速度中考虑任意数量的球谐函数。Woollands 和 Junkins[24] 开发了自适应PC方法，其中包括积分误差反馈，以加快皮卡迭代的收敛速度；还根据先前的稳定性分析，提出了用于确定该方法分段长度和多项式次数的经验规律。Atallah 等人 [25] 在不同的地球轨道算例中，将该方法与其他顺序积分技术进行了比较。

尽管该方法具有并行化的可能性，但在处理较长积分时段时的不稳定性，以及达到机器精度所需的节点数量有限（约为 200 个），都是将 PC 积分方案用于大规模并行/GPU（图形处理器）实现时的缺点。与该方法以往的发展不同，本文通过增广所积分的动力系统，实现对大批初始条件进行短期传播时的稳定大规模并行。将[12, 13], 中为找出最优轨迹而分析的所有模拟轨迹所得结果重新运行，作为所提出方法的直接应用，并深入探讨并行化方案。在单次运行中，所有轨迹弧段共有的时间节点使得 PC 的迭代改进过程可以重新组织。传播对象不再是单条轨迹的六维笛卡尔状态，而是一个二层增广状态，由同一时间节点处多条轨迹的状态经过适当排序后组成。无论增广系统中各个状态的排序方式如何，PC 过程的基本数学结构都保持不变，无需重新定义该方法的步骤、系数和矩阵。因此，该方法的最新进展，尤其是带误差反馈的二阶版本 [14] [24], ，也适用于增广版本。两个不同的增广层级使其能够保留细粒度的灵活性：在可行的情况下，可像单独模拟每条轨迹那样灵活，同时又不必过多牺牲计算效率。

基于另一种方法，目前已有用于传播大量初始条件的基于GPU的程序：Geda等人 [26] 为欧洲航天局开发了CUDAjectory工具，这是一种实现Runge-Kutta-Fehlberg 7/8格式的基于GPU的传播器。该工具通过并行处理大量不同样本的前向积分步骤，并将动力学函数的求值以及求值过程中涉及的简单矩阵乘法并行化，从而充分利用大规模并行。CUDAjectory的另一项有趣特性是采用了优化的星历读取和存储技术，该技术遵循Schrammel等人 [27], 的工作，可提高数据局部性并最大限度地减少所需的内存事务。尽管同样传播大量轨迹，但本文提出的方法在采用的数值格式上不同于CUDAjectory方法。PC方法本身就是一种可并行化的程序，而本文提出并实现的增强版本进一步提升了这一特性，从而使程序具有高度可扩展性和高性能。

最后，所提出方法的理念或可推广至更一般的不确定性传播问题，并应用于行星系统和地球系统，以开展鲁棒分析或碰撞概率计算。沉重的计算负担促使人们采用简化方法，力求以较低的计算成本实现可接受的

传播不确定性的精度 [28]。准确进行蒙特卡罗模拟的巨大计算负担被发现是一项重大挑战，例如，隼鸟2号任务的着陆设计 [29] 以及阿波菲斯小行星撞击地球概率的表征 [30]都面临这一问题。本文不拟详述为降低计算负担而开发的现有不确定性传播技术。所提出方法唯一相似之处在于，它借助成本低廉的 GPU 显著缩短了计算时间；这一思路可应用于任何其他需要对规模相对较大的初始条件集进行模拟的不确定性传播方法。

II. 皮卡-切比雪夫积分方法

A. 积分概念

皮卡迭代 [31] 是一种可用于求解初值/边值问题近似解的方法。设维数为 n 的状态为$\mathbf{x}$，自变量为 t，初值/边值条件为$\mathbf{x}_0$，动力学函数为$\mathbf{f}(\mathbf{x}, t)$，则该问题定义如下：

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t), \qquad \mathbf{x}_0 = \mathbf{x}(t_0) \tag{1}$$

从方程（1）所述初值/边值问题在区间 $[t_0, t]$ 内实际解$\mathbf{x}(t)$的初始近似$\mathbf{x}^{(0)}(t)$出发，第 i次皮卡迭代利用$\mathbf{x}^{(i)}(t)$改进对$\mathbf{x}(t)$的前一次近似$\mathbf{x}^{(i-1)}(t)$，如下式 [31]所示：

$$\mathbf{x}^{(i)}(t) = \mathbf{x}^{(0)}(t) + \int_{t_0}^{t} \mathbf{f}\big(\mathbf{x}^{(i-1)}(s), s\big) ds \tag{2}$$

只要初始近似$\mathbf{x}^{(0)}(t)$足够好且满足 $i \longrightarrow +\infty$ [31]，该方法就会收敛。

在解析皮卡迭代的背景下，通常很难进行多次迭代。$\mathbf{x}^{(i)}(t)$的表达式会变得越来越复杂，因此在前 2–3 步之后难以求得闭式解 [32]。同时，通过数值求积计算积分函数可能无法满足精度要求，因为通常只有前几次迭代能够改进函数近似。为开发可并行化的动力学运动积分例程，PC方法将皮卡迭代与切比雪夫多项式逼近相结合 [33]。一种遵循 Fukushima 工作的该方法推导过程 [34] 可概括为三个步骤：

1) 选择一个足够好的初始猜测 $\mathbf{x}^{(0)}(t)$。

2) 用切比雪夫多项式展开近似 $\mathbf{f}(\mathbf{x}, t)$ 和 $\mathbf{x}^{(0)}(t)$。

3) 执行皮卡迭代，更新切比雪夫插值多项式的系数。

当满足停止条件时，皮卡迭代将停止；停止条件基于两次

连续迭代之间的最大差值低于用户指定的某个容差。

如此定义的方法使得执行的皮卡迭代次数可以轻松超过解析情形。所涉及的表达式始终属于同一类型，即切比雪夫多项式。函数近似变成了通过节点进行的插值，这些节点应接近真实轨迹；而不再是一个全局函数，其迭代后的函数值仍取决于初始猜测的选择。此外，如果真实解 $\mathbf{x}(t)$ 与初始猜测 $\mathbf{x}^{(0)}(t)$ 之间的差异仅由小扰动 [31] 引起，那么只需少数几次迭代便足以使误差低于较小的容差。对于一般的弱受扰二体问题，以未受扰的开普勒解为起点，可确保该方法相对快速地收敛 [34]。在轨道模拟领域，Macomber [35] 将此类初始猜测称为预测校正迭代法的热启动，因为利用主导动力学部分的解析解可以减少所需的迭代次数。与之相对，冷启动则被定义为简单地将所有轨迹样本设为初始条件。通常，初始猜测越接近真实轨迹，所需的迭代次数就越少。半解析初始猜测或基于较简单模型传播得到的结果也是一种选择；对于类三体受扰轨迹，与开普勒近似相比，这会是更好的选择。Macomber 还提出了热启动的概念，适用于时间跨度涵盖地球多个行星轨道的情形[35], : 利用第一条轨道计算开普勒猜测与收敛轨迹之间的差异。随后利用球谐扰动的近周期性，将这一差异纳入起始轨迹，从而进一步减少了收敛所需的迭代次数。

B. 向量化并行计算的矩阵形式

该方法适用于并行或向量化实现（事实上，Fukushima 也提出了一个向量化版本 [36]），尤其适用于动力学函数的求值和矩阵乘法的执行。Bai 和 Junkins 在此技术基础上开展的后续研究，提出了修正PC方法 [37]以及面向 NVIDIA GPU 的 CUDA 实现 [32]。为简洁起见，并更清楚地展示并行化的可能性，本文按照 Koblick 等人 [38]提出的矩阵形式介绍该方法。

对于 N 切比雪夫节点和积分区间 $[t_0, t_{N-1}]$，自变量 t 预先采样为 $j = 0, 1, \ldots\ldots N-1$，即

$$t_j = \omega_2 \tau_j + \omega_1 \tag{3}$$

with

$$\tau_j = -\cos\left(\frac{j\pi}{N-1}\right), \quad \omega_1 = \frac{t_{N-1} + t_0}{2}, \quad \omega_2 = \frac{t_{N-1} - t_0}{2} \tag{4}$$

给定在皮卡迭代 $i-1$ 中计算得到的、由 n 维采样状态 $\mathbf{y}^{(i-1)}(t_j) = \mathbf{y}_j^{(i-1)}$，$j = 0, \ldots, N$ 构成的 $N \times n$ 维矩阵 $\mathbf{y}^{(i-1)}$，整个过程可概括为三个串行步骤，以获得这些状态

在第 i次迭代。T 第一步将动力学函数 **f** 的求值结果收集到 $N \times n$ 力矩阵 **F** [38] 中 :

$$\mathbf{F}_{j+1}^{(i)} = \omega_2 \ \mathbf{f}\left(\mathbf{y}_j^{(i-1)}, t_j\right), \quad j = 0, ..., N-1 \tag{5}$$

其次，将**A**、**C**、**S** 与方法的常矩阵相对应，其定义见 [38]，$N \times n$矩阵**B** ，按行可得 [38]

$$\mathbf{B}_1 = \mathbf{SAF} + 2\mathbf{y}_0, \quad \mathbf{B}_j = \mathbf{AF}, \quad j = 2, ..., N \tag{6}$$

第三，也是最后，$N \times n$ 状态猜测值的$\mathbf{y}^{(i)}$ 矩阵，即第 i次皮卡迭代的状态猜测值矩阵，为

$$\mathbf{y}^{(i)} = \mathbf{CB} \tag{7}$$

当两次连续皮卡迭代的状态$\mathbf{y}^{(i)}$ 与$\mathbf{y}^{(i-1)}$ 之间的最大差值低于用户选定的相对容差或绝对容差时，迭代过程即停止。

尽管理论上已证明其收敛，但较长的积分时段可能导致数值不稳定性，因为即使采用较大的 N，在经历多次轨道周转时，舍入误差仍会不断累积 [32, 34, 37]。Fukushima [34]建议采用分段方法作为解决方案；本研究已实现该方法，并使用修正PC方法按顺序逐个轨道进行积分* ，直至积分时段结束。

所提出算法的核心步骤遵循所述方案 [32, 37], ，并自动生成覆盖一个标称轨道周期的开普勒初始猜测值。

III. 双层系统增广

前文所述的不稳定性问题使 PC 算法难以实现大规模并行，因为节点数量增加到一定程度后，并不会带来精度提升。不过，正如下文所述，传播更多轨迹时可以提高并行效率。

A. 单层增广

与其仅演化由初始条件 $\mathbf{y}_0$, 确定的轨迹，不如将待积分的系统改写为： M 在相同的 N 时间节点上采样的不同轨迹，可以在同一个迭代过程中进行处理。在第 i次迭代中，矩阵 $\mathbf{Y}^{(i)}$ 收集了所有轨迹的全部样本，其第 j行与.....相关

*所提出的实现可自动处理正向或反向积分。

通过以下方式计算第 m 条轨迹的第 j 个时间采样点：

$$\mathbf{Y}_j^{(i)} = \left[\mathbf{y}_{j,1}^{(i)} \ \cdots \ \mathbf{y}_{j,m}^{(i)} \ \cdots \ \mathbf{y}_{j,M}^{(i)}\right], \quad j = 1, ..., N \tag{8}$$

动力学函数求值同样收集在矩阵 $\mathbf{F}^{(i)}$ 中：

$$\mathbf{F}_j^{(i)} = \left[\mathbf{F}_{j,1}^{(i)} \ \cdots \ \mathbf{F}_{j,m}^{(i)} \ \cdots \ \mathbf{F}_{j,M}^{(i)}\right], \quad j = 1, ..., N \tag{9}$$

其元素仍按样本逐个计算：

$$\mathbf{F}_{j,m}^{(i)} = \omega_2 \ \mathbf{f}\left(\mathbf{y}_{j,m}^{(i-1)}, t_{j-1}\right), \quad j = 1, ..., N \tag{10}$$

原则上，构建增广系统只需将不同的 M 轨迹矩阵沿列堆叠，以此定义$\mathbf{Y}^{(i)}$；$\mathbf{F}^{(i)}$ 也进行完全相同的修改。PC 迭代的结构保持不变，仍包含常规步骤。首先，使用$\mathbf{F}_j^{(i)}{}_{,m} = \omega_2 \mathbf{f}(\mathbf{y}_j^{(i-1)}{}_{m}, t_{j-1})$，对所有 M轨迹的全部 N 状态求取动力学函数。其次，对$\mathbf{B}_1 = \mathbf{SAF} + 2\mathbf{Y}_0$和$\mathbf{B}j = \mathbf{AF}$执行矩阵运算，其中$j = 2,..., N$。最后，使用 $\mathbf{Y}^{(i)} = \mathbf{CB}$更新所有 M 轨迹的猜测值。

B. 两级增广

沿列堆叠的规则可以再次应用，这次将每组包含不同M_p 轨迹的 P 组数据收集到一个矩阵中。增广矩阵 $\mathbf{Y}^{(i)}$ 现在构建如下：

$$\mathbf{Y}_j^{(i)} = \left[\mathbf{Y}_{j,1}^{(i)} \ \cdots \ \mathbf{Y}_{j,p}^{(i)} \ \cdots \ \mathbf{Y}_{j,P}^{(i)}\right], \quad j = 1, ..., N \tag{11}$$

with

$$\mathbf{Y}_{j,p}^{(i)} = \left[\mathbf{y}_{j,p,1}^{(i)} \ \cdots \ \mathbf{y}_{j,p,m}^{(i)} \ \cdots \ \mathbf{y}_{j,p,M_p}^{(i)}\right], \quad j = 1, ..., N \tag{12}$$

原则上，可以依据相同的逻辑构建无限多个增广层级，且都无需修改核心PC算法的结构。不过，重新定义迭代误差可能有助于实际应用，因为增广原理纯粹是出于计算方面的考虑。

可以考虑两种策略。第一种采用传统的误差定义，将轨迹样本视为同一系统的一部分，并将其最大值与迭代停止条件进行比较。第二种引入更灵活的逐块误差定义，将不同的轨迹块视为彼此独立，此时增广仅具有计算用途。这两种方法各有优缺点。前者实现起来更简单，而且由于

与维护分组拆分相比开销更低，因此计算效率必然高于后者。然而，对于所需迭代次数差异显著的不同轨迹而言，已收敛的块仍会占用计算资源；而逐块误差定义在这方面则灵活得多。

本研究利用两个增广层级构建一种混合方法，将外层块视为彼此独立，而在更严格的意义上将内层块视为一个整体。这样，相似轨迹组可以被视为独立但彼此分离的增广系统，从而最大限度地提升积分性能。两个层级的增广还为在同一高性能计算环境中处理单条轨迹提供了框架，可将其视为仅包含一个成员的组。举例来说，外层增广层级可用于“隔离”经历行星飞掠的轨迹子组，因为它们的动力学将与其余样本显著不同。

IV. 并行计算、GPU计算与英伟达® CUDA® 基础 s

可并行化程序的特点是其中某些部分可以同时执行。从概念上看，可能存在任务并行例程和数据并行例程：前者包含完全独立、无需按先后顺序执行的任务；后者则是指同一任务被重复执行，并应作用于不同数据。本研究聚焦于后者；当能够实现大规模并行化时，其特点有助于高效进行GPU计算。例如，计算两个矩阵的乘积是一项高度可并行化的任务，因为结果矩阵的每个元素都可以并行计算。同样，在对大量初始条件[†]进行轨道传播时，可以并行地对所有状态求值动力学函数。

A. 使用多个中央处理器的共享内存并行：OpenMP®

在讨论 GPU 计算的原理之前，最直接的并行化概念是使用多个 CPU（中央处理器）。共享内存并行是最简单的情形，此时机器上的所有计算核心都可以访问相同的公共内存位置。不过，更复杂的超级计算机采用分布式内存架构，由多个 CPU 组成的组各自访问独立的内存。这类系统还包括通信网络，用于在不同节点间分发和收集计算数据，并且需要采用自己的编程范式，其中包括数据和消息传递例程 [39]。

OpenMP 是一组预处理器指令，可在代码编译期间发挥作用，从而对 C、C++ 和 Fortran 程序进行简单的共享内存并行化 [40], 。只需稍作修改，即可加速程序中最耗时的部分；如果禁用了 OpenMP编译标志，OpenMP 指令就会被直接忽略，并视为注释。常见的并行化指令涉及 for循环，还可包含一些额外功能：例如，可以将完全嵌套的循环合并为一个更大的循环，从而提高

[†]假设它们彼此之间没有相互作用，且质量可以忽略不计。

程序效率，特定指令还可以帮助进行内存管理。尽管接口简单，大多数程序仍要求至少将某些变量声明为每个工作线程的私有变量，因为 OpenMP 默认将所有变量视为共享变量。通常需要设为私有的变量是循环计数器：每个线程都使用自己的变量至关重要，尤其是在循环计数器还用于标识共享数组中数据访问位置的情况下。更复杂的编程场景往往会出现，需要程序员显式控制共享变量的并发更新；但要解决所有这些问题超出了本文的范围。读者可参阅 OpenMP 编程指南，了解更全面、详细的信息 [40]。本段旨在简要介绍这一工具，它用于对所提出程序的 C 版本进行并行化。

并行动力学函数采用简单的 OpenMP for 循环并行化实现，将增广系统的所有状态合并到一个循环中。多个工作线程访问共享状态数组，计算加速度值并暂时存入线程私有变量，最后再将其复制回共享的全局加速度数组。OpenMP 利用 CPU 架构的灵活性，因此无需对动力学函数的最内层部分进行重大修改，就能实现高效的并行化程序。PC方法的矩阵乘法也“间接”使用了 OpenMP。程序的这一部分采用经过优化的 OpenBLAS [41–43] 库实现，这些库已内置 OpenMP 并行化功能。

B. 并行归约

并行归约机制也被用于计算 PC 迭代误差，以发挥并行化潜力。归约任务是一大类计算操作，旨在从数组元素中提取一个标量值。例如，对数组元素求和、查找数组中的最大或最小元素，以及执行“与”和“或”逻辑运算符。尽管这些任务看起来本质上是串行的，但归约任务同样存在并行化的可能：原则上，整个数组会被拆分成若干块，由多个并行工作线程分别处理各自数据块中的元素，共同完成归约任务。这个协作过程会持续进行，直到得到单个归约后的标量值。图 1 以图示展示了并行归约逻辑的示例：对于一个含有 16 个元素的数组，使用 16 个并行工作线程最终只需 5 个串行步骤。OpenMP 提供归约子句，程序员只需将最终归约所得的标量定义为共享变量 [40]。随后，编译器会确保扫描所有数组值，并避免同时覆盖归约标量。

C. GPU计算

GPU 计算的特点源自显卡的硬件架构，这种架构与传统计算单元的架构有着显著差异。图 2 以图示形式展示了这些差异：总的来说，GPU 将更多晶体管用于纯粹的数据处理，而非像

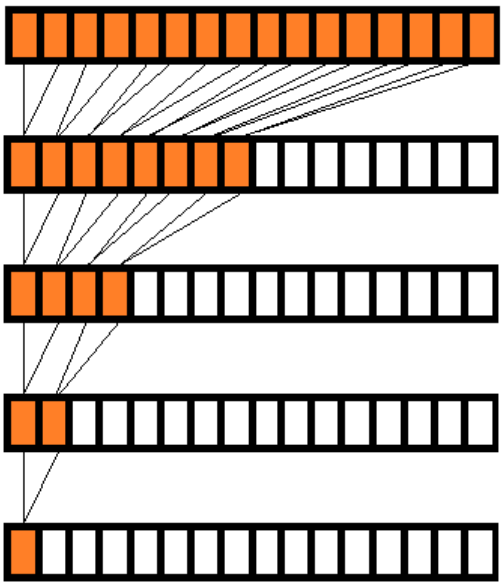


图1 并行归约示意图。

CPU 那样用于流程控制和内存管理 [44]。对应颜色表示同类型的芯片元件：绿色表示计算核心，黄色表示控制单元，紫色表示核心级缓存，蓝色表示共享缓存，橙色表示全局内存。

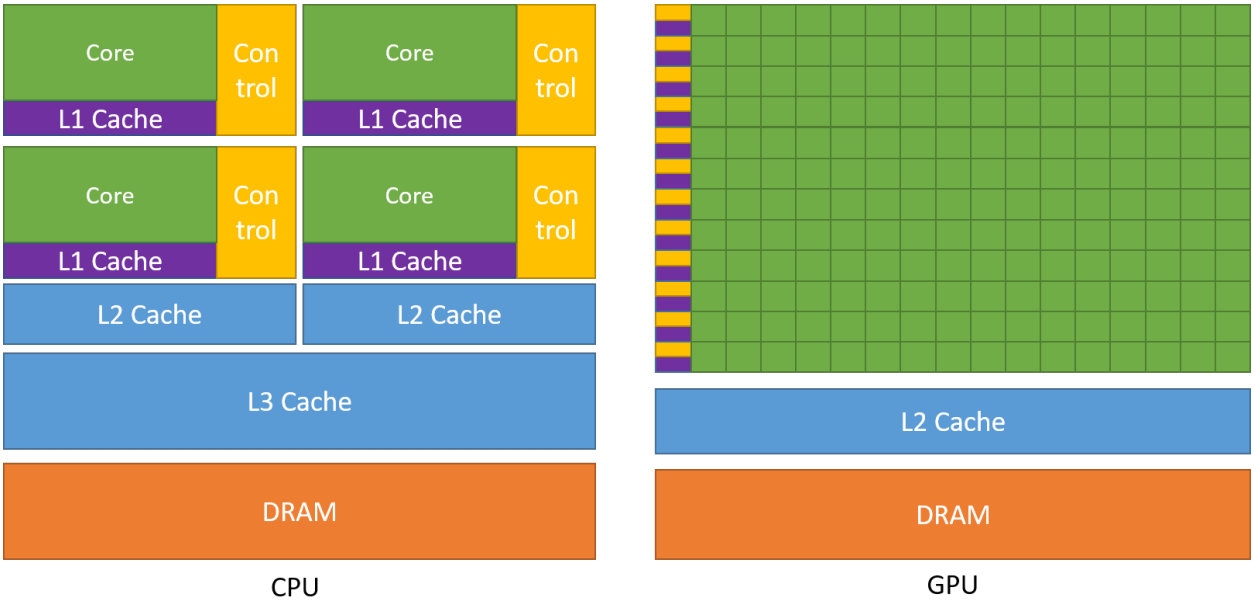


图2 CPU 与 GPU架构差异示意图。图片来源： [44]。

处理单元被分组为块（通常每块包含 32 个处理单元），每个块由一个控制器控制。同一块中的所有处理单元都执行相同的指令，该指令仅由控制器发出一次。这一特性，加上现代显卡通常配备数百至数千个处理单元，使图形处理器

适合实现大规模并行，但与 CPU 应用相比，灵活性较低，编程工作量也较大。下一小节将介绍一些关键概念，完整内容请参阅 CUDA C++ 编程指南 [44]。

D. 主要编程范式与 CUDA® 应用程序接口

本节旨在简要介绍 CUDA 语言和应用程序接口，并说明相关术语。文中使用斜体字体介绍 CUDA 专用名称和概念。完整说明请参阅 CUDA 用户手册[44]。

基本执行单元称为线程。线程可以组成块，同一块中的所有线程都可以使用一些共享内存（数十 KB）。一条指令始终由 32 个线程组成的线程组同时执行，该线程组称为线程束，与块中的线程数量无关。因此，线程数少于 32 的块无法充分利用硬件资源。通过组合多个 CPU 和 GPU，可以实现复杂配置以运行同一程序。程序流程始终由 CPU 控制，CPU 也负责控制 GPU 的执行。由 CPU 调用、但在 GPU 上执行的函数称为内核。

与标准编程相比，首要区别在于内存访问：GPU 无法读取常规计算内存，因此必须先将数据加载到图形处理器内存中，才能运行内核。同样，内核执行完毕后，必须先将数据取回 CPU，才能对同一数据执行其他 CPU 任务。因此，GPU 程序/子程序的执行流程始终是：先在主机上初始化，再将数据从 CPU 移至 GPU，随后执行内核，最后将数据从 GPU 取回 CPU，如图 3 所示。

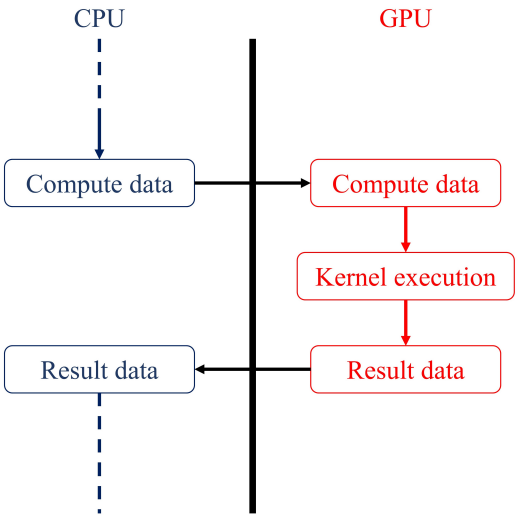


图 3 GPU 程序基本流程。

由于所有线程同时执行相同的指令，内核必须以面向线程束的方式编写。这也意味着需要显式管理各线程的数据访问，而复杂函数通常要求程序员手动优化内存访问和缓存利用率：对于 CPU，这些工作由编译器隐式控制；而显卡默认将数据存储在所谓的全局内存中。这种内存容量很大（数 GB），GPU变量初始化时默认使用它。这样即使数组很大，也能简化 CPU 与 GPU 之间的数据移动。不过，与共享内存相比，全局内存的访问延迟要高得多，因此在对 GPU变量进行重复读写操作时，效率不高。为克服这一局限，可将共享内存用于计算用途，把只读值存放在访问延迟较低的位置，而不只是存放供块内所有线程共享的值。此外，寄存器中也有少量缓存空间；寄存器是线程私有的位置，最多可存储 256 个单精度浮点值。由于寄存器容量较小，且相对论动力学函数的计算密集度较高，对于本文提出的应用而言，很难用寄存器替代共享内存。寄存器仅用于存放临时变量和句柄变量，以辅助最终的加速度计算。

数组元素的排列方式对于优化全局内存与共享 GPU 内存之间的数据移动也至关重要。CPU 与 GPU 之间的数据传输由数组大小决定，而当线程读写相邻位置上的值时，GPU 内部内存访问效率最高。满足这一条件时，所有线程的数据都能通过一次内存事务完成移动，从而最大限度地减少读写全局内存所耗费的周期。这种内存访问模式称为合并访问，是复杂 GPU 程序中的一项关键性能驱动因素：即使程序具备大规模并行化的可能性，非最优的内存访问也可能导致内存访问延迟无法通过并行化的计算任务得到任何弥补。

英伟达® 开发并维护 CUDA® 编程语言，大幅简化了在计算机程序中使用英伟达® 图形处理器的过程。CUDA 是基于 C语言++ 扩展构建的，包含一组关键字和 API函数，可让程序员构建自己的内核并控制 GPU执行流程。此外，还提供了一系列优化库，例如本工作中用于 PC矩阵乘法的基础线性代数库 cuBLAS® [44]。

E. 并发与高级功能

CPU-GPU二元性与协作带来了更多可能性，而不仅仅是加速程序中的密集部分。通常，内核调用是异步的，不会阻塞 CPU 的工作，因此 CPU 可以在 GPU 内核仍在运行时执行其他任务。此外，现代图形处理器能够同时管理两次内存传输（一个方向各一次，即从 CPU 到 GPU 以及从 GPU 到 CPU），同时让其计算单元满负荷运行，以执行一个或多个并发内核 [44]。

CUDA® 允许借助流将一系列存在顺序依赖关系的 GPU 函数调用排入队列：例如，应用程序可能需要先将一些数据传输到 GPU，然后才能执行自定义内核，

而且必须先完成该内核的执行，才能调用 cuBLAS® 函数；调用结束后，应将处理后的数据传回 CPU。只需将存在顺序依赖关系的 GPU 函数调用分配到同一个流即可。可以同时创建和使用多个流，其产生的行为类似于向超级计算设施提交批处理作业。CUDA® 可确保同一流中的执行顺序和同步正确无误，而不同流之间则必须手动进行同步。编译器通常会安排执行和内存事务，以最大限度地利用 GPU，并将不同流中的 GPU 函数调用和数据传输重叠执行 [44]。

尽管延迟较低，优化后的程序还旨在控制线程访问共享内存位置的方式。如果要求两个相邻线程访问两个不连续的数组元素，则内存访问需要两个周期，而非一个周期，从而稍微降低程序执行速度。这个问题称为存储体冲突，通过将随线程变化的元素存储在共享内存数组的主维度中即可避免。如果所有线程都访问同一个内存位置，则不会发生存储体冲突。

V. 实现

本节概述的各种实现中，将所有传播轨迹分别进行独立 PC 运行的情况作为基准测试。这使我们能够针对同一个测试案例，直接评估增广 PC 算法相对于原始积分器的性能。所有算法均使用 C 语言实现，唯一的例外是使用 CUDA 编写的 GPU 程序。

A. 独立运行 PC 工作流

独立 PC 运行的基本工作流如图 4 的框图所示。当轨迹数量较大时，唯一可行的并行化只能在最高层进行，这不可避免地会给内部串行执行以及仍可并行化的内部函数带来相当大的开销。事实上，PC 过程中的所有逐轨迹步骤本身仍然都是可并行化的算法。

B. 顺序增广 PC 工作流

增广 PC 积分的实现仅采用单层增广，以突出该实现带来的开销方面的流水线优势。图 5 给出了增广 PC 积分工作流的框图表示。从 PC 迭代的角度来看，概念上的变化仅在于：初始采样在一个数组中完成，该数组包含增广系统所有轨迹的全部状态向量。

C语言. OpenMP® 并行化增广 PC 工作流

增广系统积分的并行化变得更加细粒度，其块方案表示如图 6 所示。它直接作用于用于动力学函数求值的单个状态向量，以及

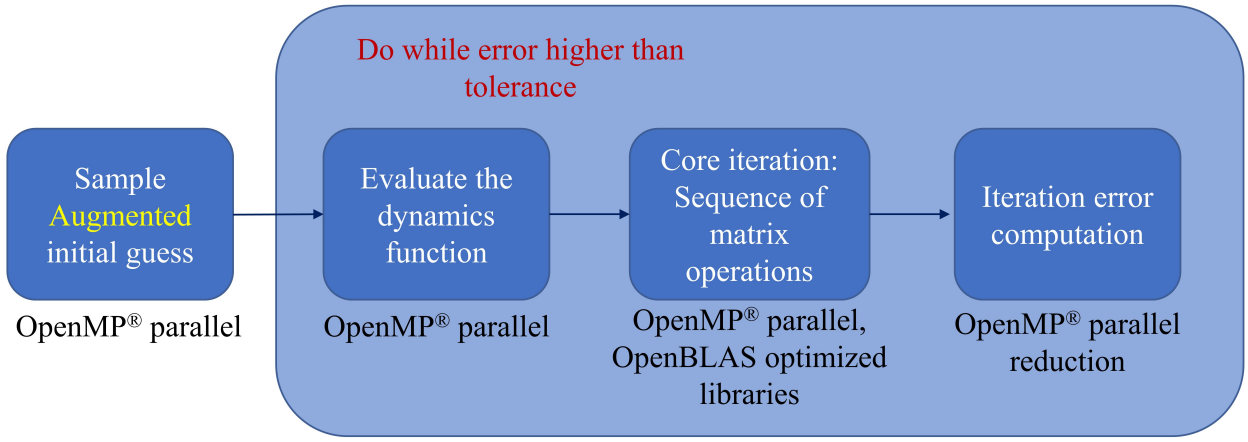


图 4 标准 PC 工作流。

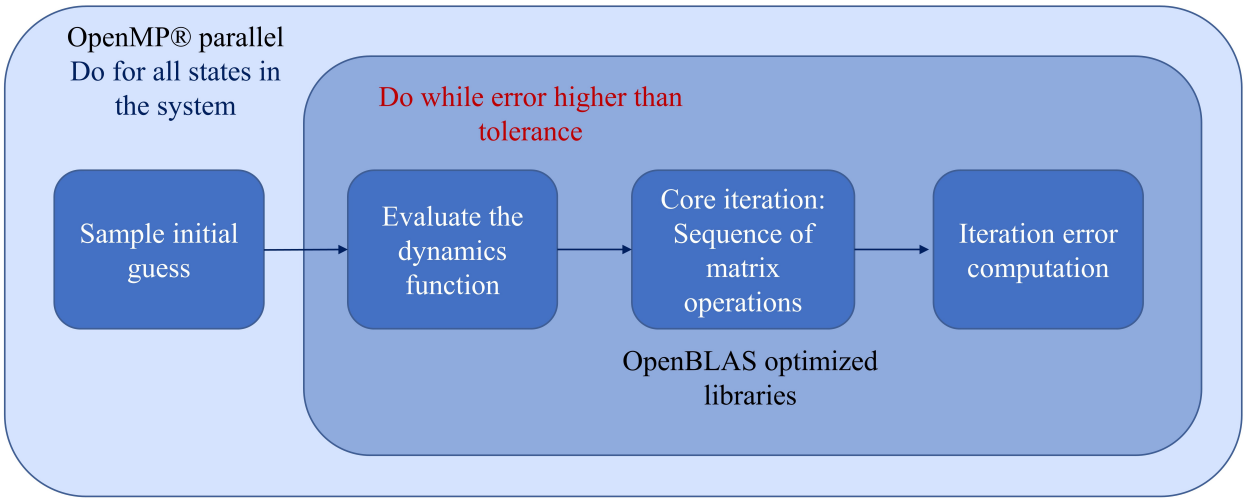


图 5 增广 PC 工作流。

矩阵运算的基本乘积；此外，这些运算已由 OpenBLAS 库 [41] 实现。另外，可以通过 OpenMP® 执行归约运算，以协同并行地搜索最大误差。

D. CUDA® 增广 PC 工作流

CUDA® 算法的框图表示如图 7 所示。该算法利用两级增强概念，为每个 CUDA® 流分配一个高级增广系统，并在低级增广系统上利用基于线程的并行性。

其主要优势在于 CPU 与 GPU 协同执行整体任务，并尽可能让更多操作并发执行。每个低级增强系统最初由主机进行采样，然后传输至 GPU。CUDA® 流管理应用程序接口可使 CPU 对下一个更高级增广

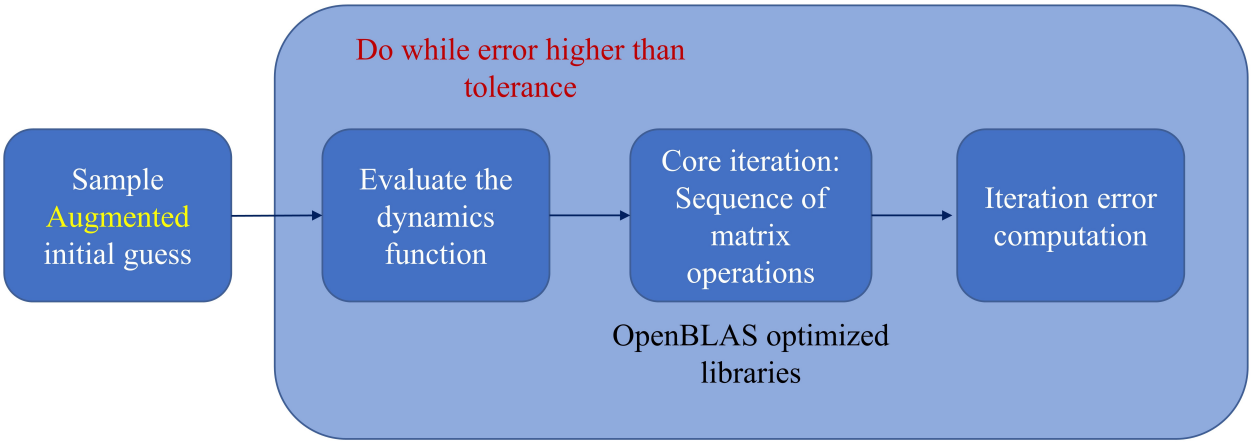


图 6 增强并通过 OpenMP® 并行化的 PC 工作流。

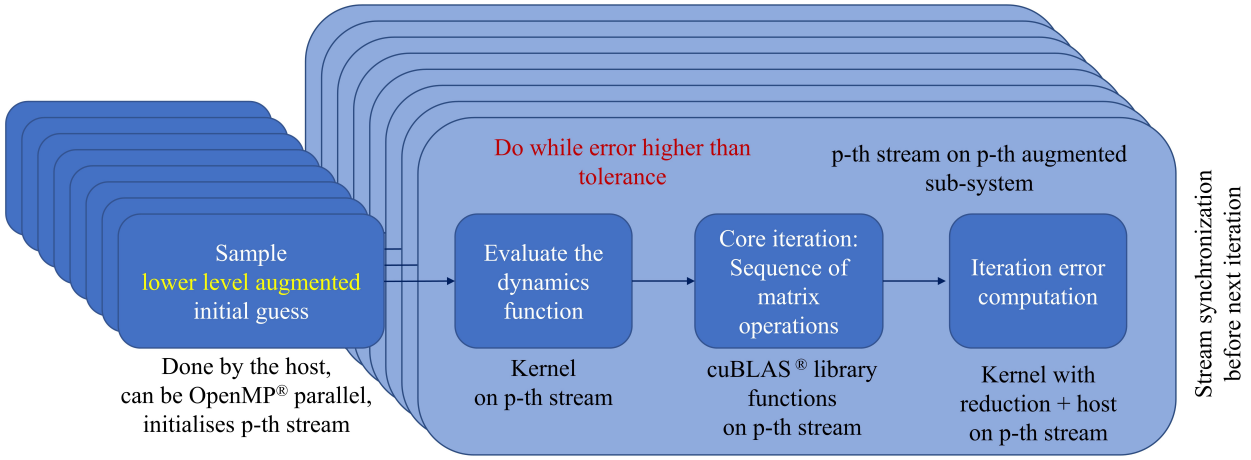


图 7 增强的 CUDA® PC 工作流。

系统进行采样的同时，对已启动的系统进行内存传输和内核执行‡。同样，PC 迭代的最后一步需要将每条流计算出的迭代误差从 GPU 取回 CPU，以便进行循环控制，这也能受益于流并发。每次 while 循环迭代结束时都必须进行流同步，才能实现所有流的重叠执行。若为每个更高级增广系统分别运行独立循环，则会导致执行完全串行，无法重叠。如果只生成一个 CUDA®流，则会重现标准的单级增广系统情形，不过采用的是 GPU 计算加速度，而非前文所述的 OpenMP® 实现。

CUDA® 内核的以 warp 为中心的编程模型要求对低级增强系统的定义稍作修改。连续的数组元素应具有相同的分量类型（即连续的 x 坐标，接着是连续的 y 坐标，以此类推），而不是按状态向量逐个存储。这一点看起来可能

‡只要 CPU 内存通过特定的 CUDA® 函数 [44] 分配为分页，这一点就成立。

像是实现细节，但它对于确保合并全局内存访问至关重要。否则会产生过高的延迟，即使进行密集的并行化 GPU 计算也无法将其隐藏。刚才讨论的修改不会影响整体算法结构，只需按照这种数组元素逻辑实现动力学和误差内核即可。下一节将进一步讨论这一点，尤其是它对动力学函数求值的影响。

1. 动力学模型、数组排序与 *CUDA* 核函数

PC 积分方案的核心在于每次皮卡迭代中对动力学函数进行求值。对于待积分系统的所有状态，这项任务可以并行执行；然而，尽管概念上很简单，在 GPU 计算中的实现却未必直接明了。动力学模型越复杂，其实现不可避免地越相互交织，可能需要访问分布在多个数组中的数据，而且这些数组的大小可能存在显著差异。所提出应用的精度要求，需要采用爱因斯坦-英费尔德-霍夫曼方程所描述的受限相对论 N 体问题 [45], ，其动力学函数形式如下：

$$\ddot{\mathbf{r}} = \mathbf{f}\left(\mathbf{r}, \dot{\mathbf{r}}, \mathbf{r}_i, \dot{\mathbf{r}}_i, \ddot{\mathbf{r}}_i\right) \tag{13}$$

其中， i 概括了对星历模型中所有主要天体状态的依赖关系，例如太阳系行星；$\mathbf{r}$、 $\dot{\mathbf{r}}$、 $\ddot{\mathbf{r}}$ 分别表示位置、速度和加速度。遗憾的是，这些关系是非线性的：因此，每个 CUDA® 线程都无法仅对单个数组元素执行简单操作，因为计算任一加速度分量都需要每个状态的位置和速度。此外，$\mathbf{r}_i$、 $\dot{\mathbf{r}}_i$、 $\ddot{\mathbf{r}}_i$ 的星历数据也会进入动力学函数。这些因素表明，应当实现动力学内核，让每个 CUDA® 线程处理一个完整的状态向量，而非单个元素。同时，合并全局内存访问对于获得高性能内核仍至关重要。然而，PC 方法对数组形状施加了部分约束：构成该方法的矩阵乘法要求将采样轨迹状态存储为总矩阵的各行，并以不同时间标识每一行。对于按列主序排列的增广状态矩阵，连续的状态元素会被末尾的时间节点隔开，可能导致多个线程束发生非合并内存访问。相反，按行主序排列的状态数组会使所有状态元素都发生非合并访问。

为解决这些问题，将低层级增广状态重新表述为：沿列堆叠增广系统中所有状态向量的相同分量：

$$\begin{gathered}\mathbf{Y}_j^{(i)} = \left[x_{j,1}^{(i)} \;\cdots\; x_{j,M}^{(i)} \;\cdots\; p_{j,1}^{(i)} \;\cdots\; p_{j,M}^{(i)} \;\cdots\; \dot{z}_{j,1}^{(i)} \;\cdots\; \dot{z}_{j,M}^{(i)} \right], \\ j = 1, \ldots, N, \quad p = y, z, \dot{x}, \dot{y}\end{gathered} \tag{14}$$

其中 (x, y, z) 是 $\mathbf{r}$的笛卡尔分量。在列主序排序情况下，这一优势显而易见，因为

所有公共分量都位于相邻的内存地址中。由于在所述应用中，增广系统中的状态数量远大于采样的轨迹节点数量，因此在行优先排序数组情况下，也会出现许多连续的状态分量。此外，较高层级的增广系统定义可以保持不变，因为不同的 CUDA® 流将分别处理每个较低层级的子系统。增广力矩阵$\mathbf{F}_j^{(i)}$ 可以相应地进行调整，而无需对表征 PC 迭代的矩阵乘法进行任何修改。最后，采用这种数组排序方法可以自动避免存储体冲突（见第 IV.E 节的解释）。

内核设计与数组排序策略密切相关。尤其是，固定时间节点发挥着关键作用。不同状态向量共享时间节点这一特点决定了增广系统的逻辑；设计线程块时也应利用这一特点，充分发挥可用共享内存的作用。具体而言，如果一个块中的所有线程都处理对应于同一时间节点的状态向量，存储行星星历所需的内存量就会降至最低。因此，所提出的程序采用行优先排序策略对状态矩阵和力矩阵进行排序。增广状态矩阵以二维块数组的形式访问：一个维度（行）对应不同的时间节点，另一个维度则用于将同一时间节点的状态拆分成较小的块，每块包含 32 个状态[§]。这样，二维块数组中同一块内的所有状态都需要完全相同的星历数据，因为它们都对应同一时间节点。对于固定时间节点的线程块，即使从共享内存读取星历数据，也会自动避免存储体冲突，因为所有线程都被要求访问相同的星历项或向量分量 [44]。

动力学函数的计算属于计算密集型任务：大部分工作在于对有限的数据量进行计算，而非在两个内存位置之间传输大量信息。此外，状态元素的值需要反复访问。出于这些原因，并且考虑到线程私有寄存器的容量有限，我们也使用共享内存暂存状态值，因为与 GPU 全局内存相比，共享内存的延迟更低 [44]。最终的动力学内核如图 8 所示。

此外，仔细研究爱因斯坦-英费尔德-霍夫曼方程可以发现，每个传播天体的加速度不仅取决于其状态，以及星历模型中主要天体的引力参数和状态，还明确取决于这些天体的引力势和加速度 [45]。在所提出应用的限制性问题中，这些贡献可以在动力学函数本身求值之前计算出来，从而免除每次皮卡迭代都重复执行这一任务所带来的计算负担。星历模型中主要天体的引力势和加速度值由 CPU 在开始皮卡迭代之前计算，然后移入 GPU 全局内存，最终加载到与每个时间节点对应的线程块的共享内存中

[§]32 是大多数英伟达® 显卡 [44]的线程束大小。对于状态数量不是块大小整数倍的增广系统，动力学内核的实现方式是让最后一个线程块处理状态数量除以块大小所得余数对应的部分。

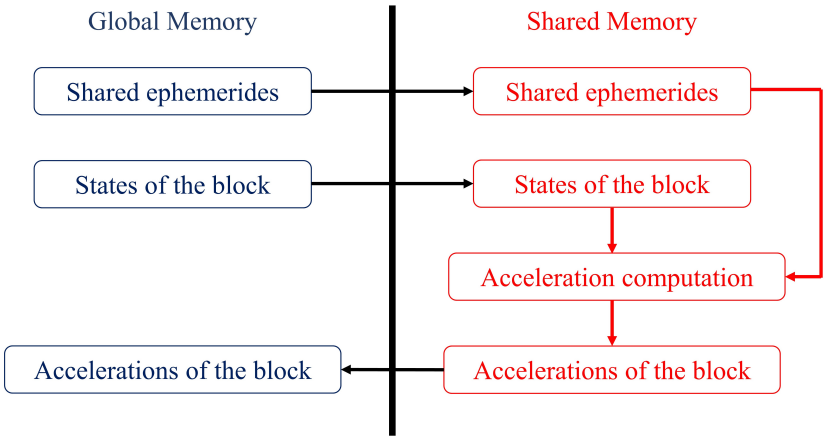


图 8 动力学内核的内存管理。

以及星历状态和引力参数。

2. CPU-GPU 协同迭代误差计算

所有涉及的 CUDA® 内核都会在每次皮卡迭代中运行。它们必须由 CPU 调用；当连续两步之间的误差低于所需容差时，CPU 还会终止皮卡迭代 while 循环。由于更新后的状态已经位于 GPU 上，因此可以利用 GPU 的大规模并行能力加速迭代误差的计算，只向 CPU 传输少量数据供循环控制使用。尽管处理的是增广状态系统，误差仍按每个状态分别定义，并使用所有状态误差中的最大值来控制循环。

误差计算过程涉及两个独立的内核和一个CPU函数。第一个内核计算位置误差和速度误差，并针对增广系统中的所有状态，存储两者中的较大值。第二个内核计算每4096个状态组成的组中的最大误差：创建1024个线程大小的块，在将连续四段状态误差读入共享内存时，找出各段的初步最大值。随后，1024个线程协作，通过归约驱动并行¶，从剩余的1024个状态误差中找出实际最大误差。最后，将最大误差复制回全局内存，存入一个仅包含归约后误差的新数组。然后将整个数组复制回CPU，由CPU采用传统的基于for循环的串行方法找出实际最大值。即便增广状态包含数百万个状态向量，这种方法也使CPU只需对至多数百到数千个候选值进行串行搜索，与其他步骤相比，计算开销可以忽略不计。

¶与动力学内核类似，误差核函数的实现方式是：让最后一个线程块处理状态数除以块大小所得余数（归约内核的块大小为 4096）。相应地控制归约步骤：仅启用线程数小于或等于该块中状态数的那些步骤。

VI. 程序性能

A. 测试案例

测试案例沿用了 [12, 13]中已有的计算结果。复现了类似太阳轨道器任务与金星发生的首次共振相位，并设计出一条在飞掠注入和飞掠离开期间也连续的轨迹。该方法基于飞掠的B平面 [46] 描述，并采用PC方法进行高效的不动点模拟，以在相对论性N体环境‖ 中探索轨迹，同时尽量减少控制飞掠入口所需的深空机动。按照图9所示的轨迹方案，设计算法采用向后递推、类似动态规划的递归逻辑，即先设计飞掠 $j+1$ ，再设计飞掠 j，以保持精度和连续性，同时满足任务要求。读者可在 [12, 13]中进一步了解设计算法及相关技术。

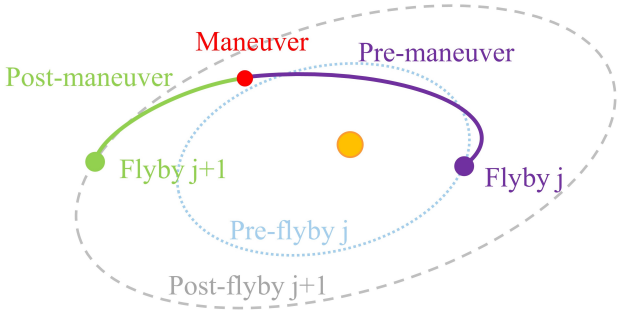


图 9 飞掠 j 至飞掠 $j+1$阶段的轨道示意图。

在配备 Intel® Core™ i7-7700 CPU（3.60 GHz）并运行 Windows® 10 Pro 的本地工作站上，使用单个核心重新运行了 [12, 13] 中提出的 Matlab®** 实现。该算法在 506.3 秒内完成了 13509 次 PC积分，并收敛至所需机动对应的残差 $\Delta\mathbf{r}^*$ 和脉冲作用 $\Delta\mathbf{v}^*$，相关机动见表 1。为更详细地说明其中的 PC积分，每次积分均使用 200 个切比雪夫节点覆盖该弧段

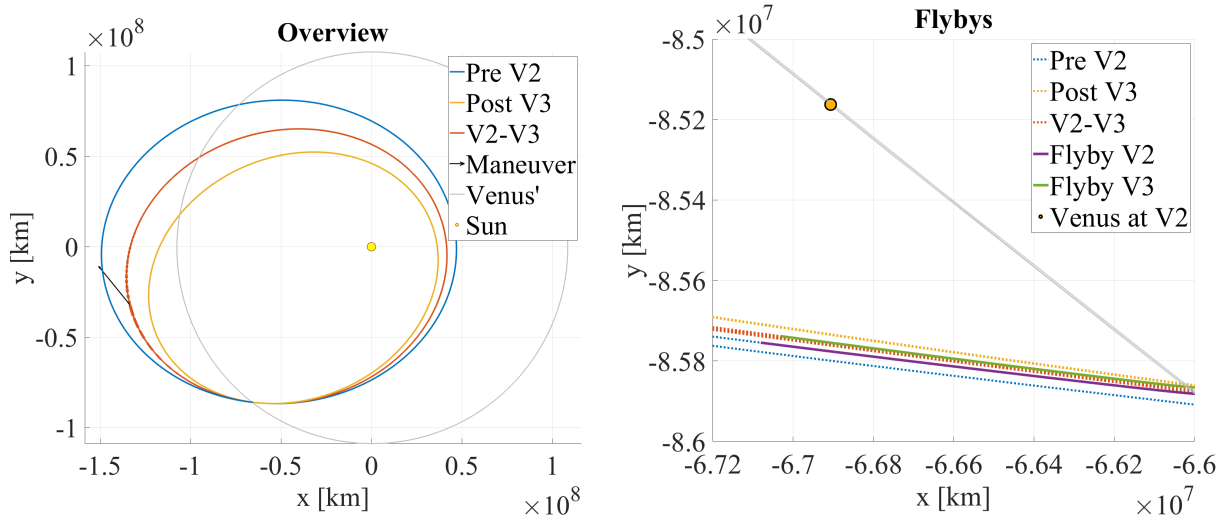
表 1 优化结果：机动时刻（对应于飞掠后轨道远心点处）的位置差残差 $\Delta\mathbf{r}^*$ 和校正代价 $\Delta\mathbf{v}^*$ 。

$\Delta\mathbf{r}^*$ [m]			$\Delta\mathbf{v}^*$ [米/秒]		
x	y	z	x	y	z
−0.52	−0.52	−1.19	−1.28	1.57	0.22

连接飞掠出口与机动点，长度约为0.87个轨道周期。图10a和图10b展示了在相对论性N体环境中优化得到的最终轨迹。

‖即使在飞掠期间也必须确保轨迹连续性，这要求极高的设计精度，因为近距离接近时快速变化的动力学会导致轨迹发散，从而变得毫无意义，而且这种情况在首次飞掠后就会出现。

**更新至 Matlab® R2021b 后，获得相同最终结果所需的运行时间更长，但这不影响本文讨论，因为性能分析仅针对 C语言和 CUDA 算法版本进行。



(a) 深空修正机动及概述。 **(b) 金星 V2 和 V3 飞掠的局部放大图。**

Fig. 10 太阳轨道器与金星的首次连续共振相位。图片来自 [12, 13] .

B. 计算设置

为纯算法性能评估建立统一框架，我们使用 PC 积分的 C 和 CUDA 实现，重新运行优化过程中生成的 13509 个初始条件，以最终获得表1中的结果。将使用该算法的 C 实现执行 13509 次独立运行作为基准案例，分别采用完全串行方式和 OpenMP [40]并行化方式。PC 迭代中的矩阵运算使用 OpenBLAS 库 [41–43] 完成。所有展示的 C 算法运行均在一台运行 Ubuntu Linux 20.04 的机器上执行，该机器配备 40 个物理核心／80 个逻辑核心，采用运行频率为 2.1 GHz 的 Intel® Xeon™ CPU E5-4620 V4 处理器，并设置不同数量的 OpenMP 线程，同时启用“o3” gcc编译器优化。由于可用 GPU 所在的物理机器的限制，CUDA™ 代码在与 Matlab® 最优解计算相同的工作站上运行。在这种情况下，Intel® Core™ i7-7700 CPU（3.60 GHz）上的四核心 OpenMP®并行化与 NVIDIA®GTX 1050（1.3GHz）显卡††协同执行。最后需要指出，在所有展示的案例中，针对相应初始条件，不同实现之间的相对误差始终低于指定的 PC相对容差（10^{-12}），因此可以仅比较运行时间。这些微小差异是不同机器、库和实现所执行的基本运算顺序不可避免地略有不同所致。

C. 精度比较

图11展示了 C 语言增强程序和 CUDA® 程序的传播误差随时间的变化。该误差以相对于独立运行情况的位置和速度相对误差衡量。两种情况下，误差均保持

††这是一款 2016 年的低端游戏显卡，其设计用途与本研究所需的双精度计算相去甚远。现代游戏／专业显卡在此应用中的运行速度可提高至该型号的 60 倍，而数据中心显卡则可提高至 400–500 倍。

low高于用于终止皮卡-切比雪夫迭代的指定相对容差，该容差设为 10^{-1} [2].

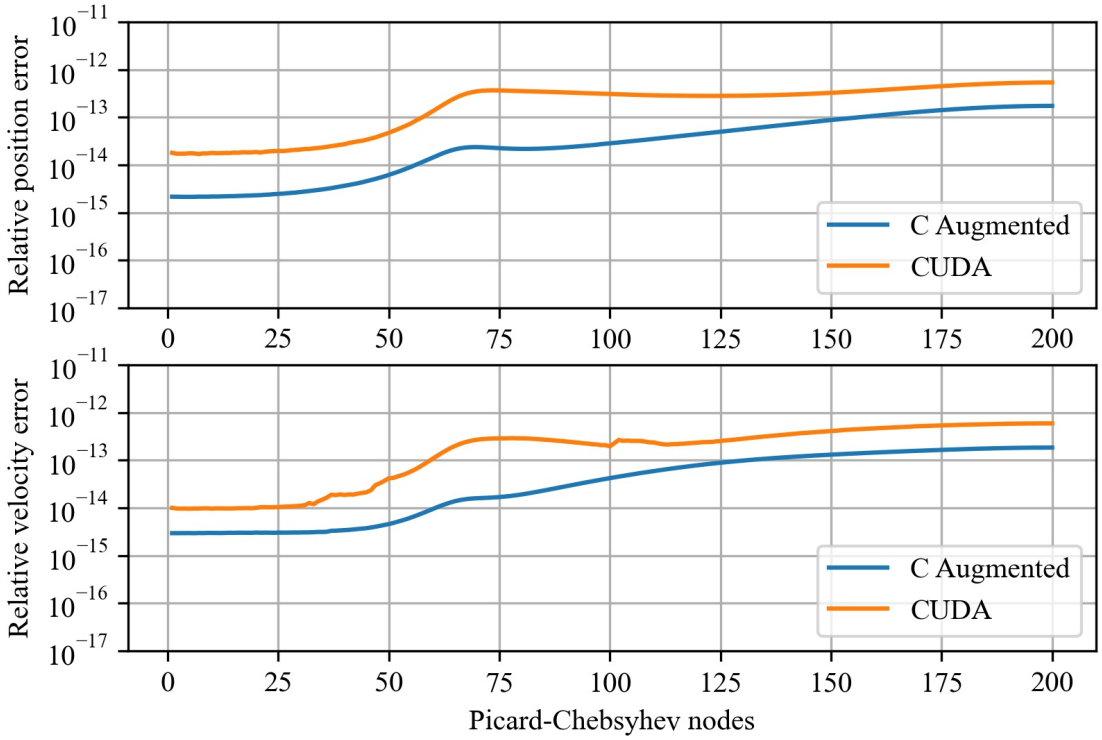


图 11 C语言增广程序和 CUDA® 程序的误差，即增广系统中所有状态误差的平均值。

图 12 则展示了仅针对动力学函数的误差变化，并区分了 CUDA® 程序的两种不同编译方式：启用（红色虚线）和未启用（橙色） -use-fast-math 标志。尽管图11中出现了误差，C 语言增强程序在动力学函数计算中并未显示误差。这意味着，误差仅在 Picard-Chebyshev 迭代过程中累积，这是由矩阵乘法结构不同所致。OpenBLAS 库 [41–43]中的矩阵乘法实现针对矩阵大小进行了优化。这是高性能计算问题中一个已知现象：数字的浮点表示会受到矩阵元素基本数学运算顺序变化的影响，从而相对于非并行化的参考解产生微小偏差。并行化与未并行化的 C 语言增强程序之间未观察到差异。除此之外，CUDA® 程序还会受到不同编译器引入的误差影响。除了运行环境不同（C 语言增强程序使用带有 GNU® 编译器的 Ubuntu Linux, CUDA ® 程序使用 Windows 10 Pro、Microsoft® Visual Studio® 和英伟达®CUDA® 编译器）之外，在编译时设置不同的优化标志也会产生细微差异。启用和禁用快速数学优化选项的对比实验是在

以排除动力学内核实现问题的可能性。比较两次编译可以发现，达到 10^{-12} 的误差尖峰似乎以不可预测的方式出现；由于它们在两种情况下分别发生于不同的皮卡-切比雪夫节点，因此完全可以归因于编译器。除此之外，误差水平始终低两个数量级以上。无论如何，这种误差来源的累积效应并未使总误差超过 10^{-12} 迭代容差，正如图11所示。

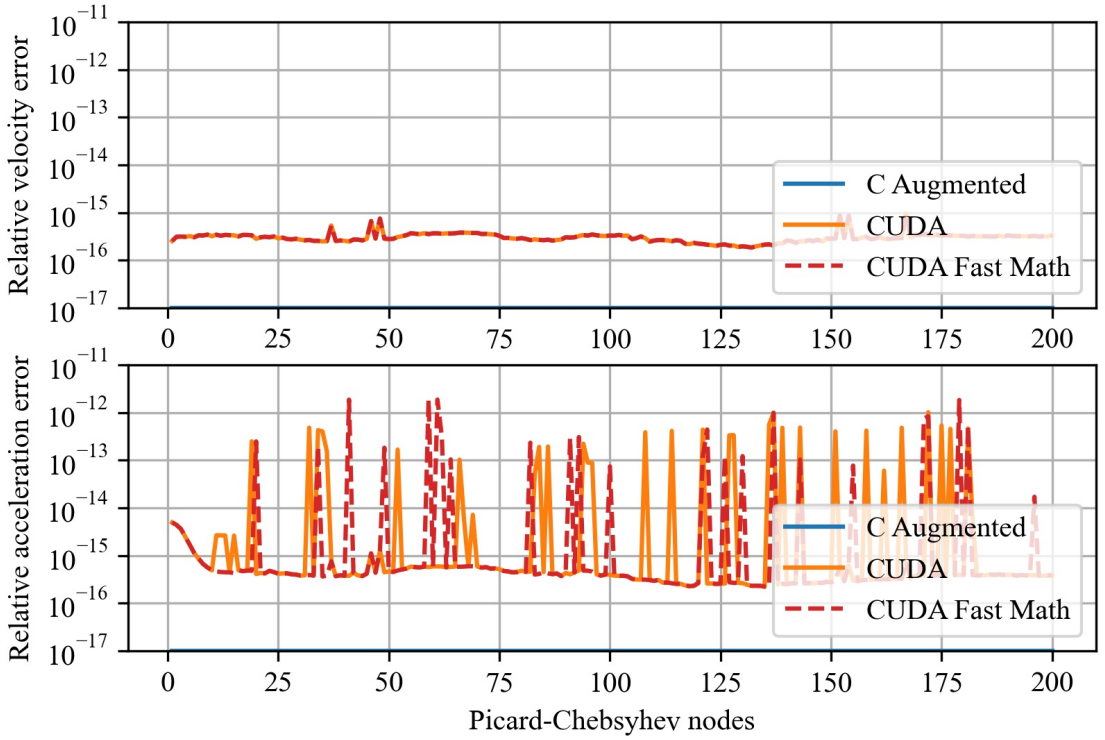


图12 C语言增广程序和 CUDA® 程序的误差，即增广系统中所有状态误差的平均值。

D. 性能比较

表 2 展示了顺序C程序之间的运行时间差异。即使在这种串行情况下，也能立即看出增广积分效率更高：增广系统方法的运行速度快了 23.87%，这是因为共享外层while循环减少了开销。在所考虑的应用中，各条轨迹均需要 41 或 42 次 PC 迭代，因此，即使某些轨迹的 PC 过程已经收敛，继续运行它们所需的开销也微不足道。

在 C 实现中研究了独立轨迹积分和增广系统的可扩展性，即评估随着

Tab表 2 独立运行与增广系统执行的串行运行时间 **s.**

Case	运行时间 [s]
独立运行	245.02
增广系统	186.54

OpenMP® 线程数量的增加，这两个程序的执行速度能提升多少。图 13 显示，增广系统具有出色的可扩展性，随着 OpenMP® 线程数量增加，其运行时间持续缩短。相比之下，独立轨迹积分在超过一定线程数量后，运行时间反而更长。这是因为并行化本身会给整个程序的执行带来一些开销，而新增的并行线程无法弥补这部分开销。

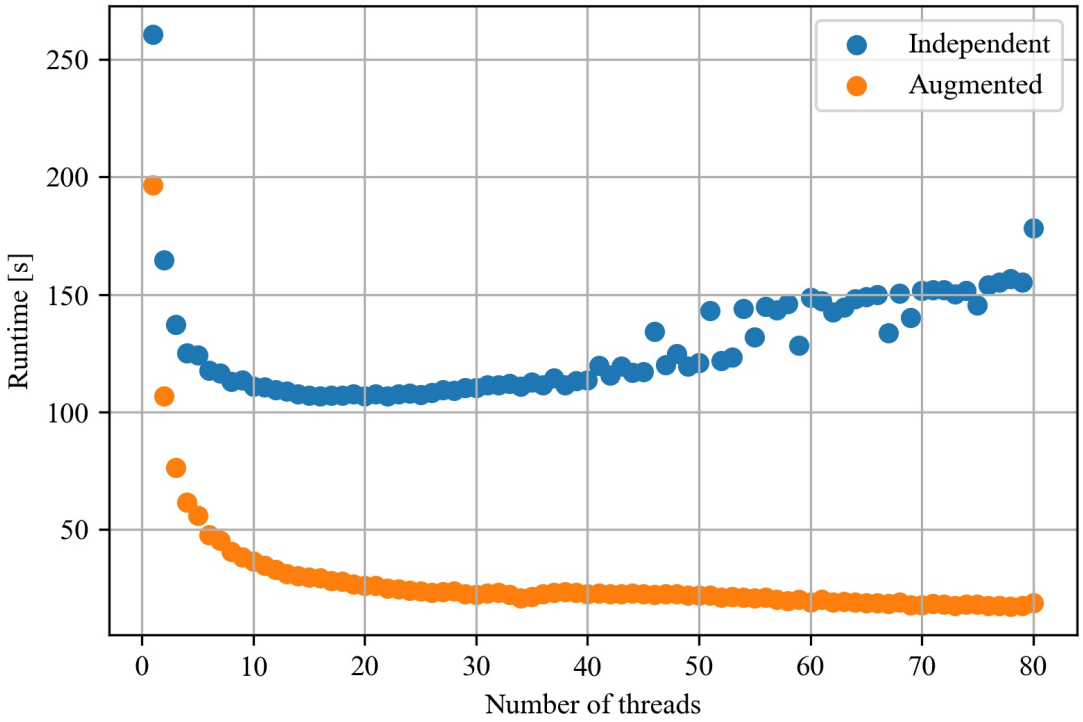


图 13 使用 OpenMP® 并行化且线程数可变时，增广系统与独立积分的 C 代码运行时间比较。

图 14 展示了所实现的加速比，即串行运行时间与并行运行时间之比，并显示了 OpenMP® 线程数量变化时的情况。该指标衡量算法的可并行化程度，数值越高，表明加速效果越显著。增广系统积分再次展现出出色的可扩展性，即使在线程数量较多时也是如此，这表明即使尚未实现 CUDA® 实现的性能，GPU 计算迁移也具有很高的效率。

在评估了 p 之后　　CUDA® 实现的性能。如果线程数设为 1　　，

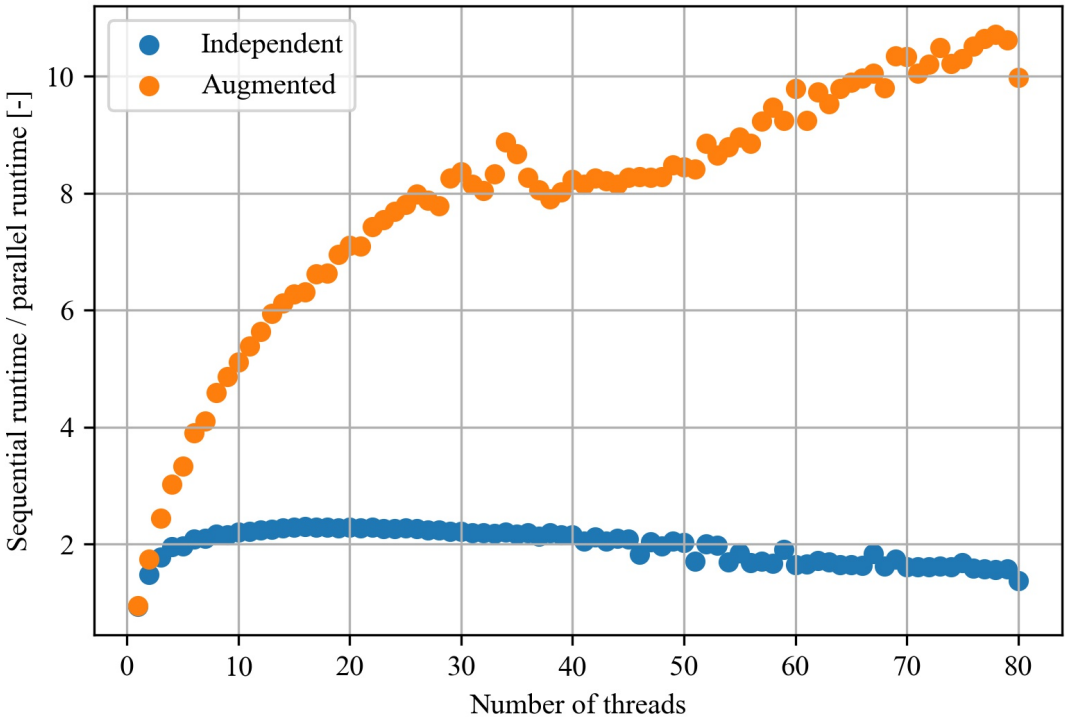


图14 增广系统与独立积分的 C 代码加速比比较，采用 OpenMP® 并行化并改变线程数。

从概念上讲，该算法可简化为串行情况。然而，在编译时启用编译器优化和 OpenMP®标志后，并行程序会在可能存在的多个工作线程之间引入数据分发和检索任务。使用 OpenMP®时，线程数通过环境变量在程序运行前指定。因此，仅激活一个线程只会带来并行化产生的所有开销，却没有任何计算收益。因此，独立系统和增广系统在单线程下的运行时间都高于基准串行运行时间；后者是在启用优化但未启用 OpenMP® 标志的情况下编译的。

最后，CUDA® 实现‡‡ 的整体执行速度最快，用时 15.84 秒。整个轨迹集被分为 10 个流§§。实际应用中的流定义准则应与传播器的运行环境相适应，这是实现所保留的唯一灵活性。不过，从 GPU 的角度来看，内核越大，GPU 的利用率通常越高。因此，创建过多的外层增广层级（即启用

‡‡仅展示未使用 -use-fast-math选项进行编译所获得的结果。运行时间没有显著差异（差异小于 0.1 秒）。§§该值由 13509 个状态除以 9 的整数除法结果得出，其中第十个流包含的状态数等于该除法的余数。

大量独立的 CUDA® 流，而每个流包含的轨迹过少）会导致性能下降，最终得到的结果与此前针对单条轨迹的独立积分案例中观察到的情况相同。

表3总结了前文讨论的不同情形下，在 C 实现中选取不同核心数时的运行时间结果。

表3 独立运行和增广系统执行的运行时间。每项均为10次不同运行的平均值。

Case	线程	GPU	运行时间 [秒]
C 独立性	1	-	245.02
C语言 增广系统	1	-	186.54
C语言 独立	8	-	113.09
C语言 增强	8	-	40.60
C语言 独立	40	-	113.54
C语言 增强	40	-	22.67
C语言 独立	80	-	178.18
C语言 增强	80	-	18.70
CUDA® 增广	4	GTX 1050	15.84

图15显示，与所有样本的串行模拟和独立模拟相比，增广系统实现的加速效果不断提升。CUDA® 实现的运行速度比基准情形快 15 倍以上，这表明增广 PC 算法适用于高性能计算和 GPU 计算设施。

E. CUDA® 内核性能分析与优化

最终的内核实现是经过长期、细致的剖析与优化过程得出的，该过程使用了 Nsight® Compute 工具和英伟达® GeForce® GTX 1050 显卡。动力学内核采用 32 个单元的块大小，主要是因为需要将共享内存用于临时存储双精度状态和加速度向量元素；这些数据太大，无法存放在线程私有寄存器中。误差计算内核不存在这一瓶颈，因此每个块可以启用全部 1024 个线程，从而尽量减少内存事务并最大化归约效果。在读取星历数据中的误差时，进一步进行的四次归约（详见 V.D.2节）是多次试验的结果：增加更多归约层会导致内核减速，而 CPU函数最终调用所获得的加速不足以弥补这一减速。类似地，移除其中一些归约层会导致 CPU函数减速，而内核加速不足以弥补这一减速。不过，读取过程中所选用的归约次数可能会受到所选测试案例的问题规模影响。需要传播的初始条件集合较大（或较小）时，最大误差计算可能会有不同的最优实现。相比之下，32 已经是动力学内核的最小块大小，与问题规模无关。其他非相对论动力学函数可能会以不同方式使用

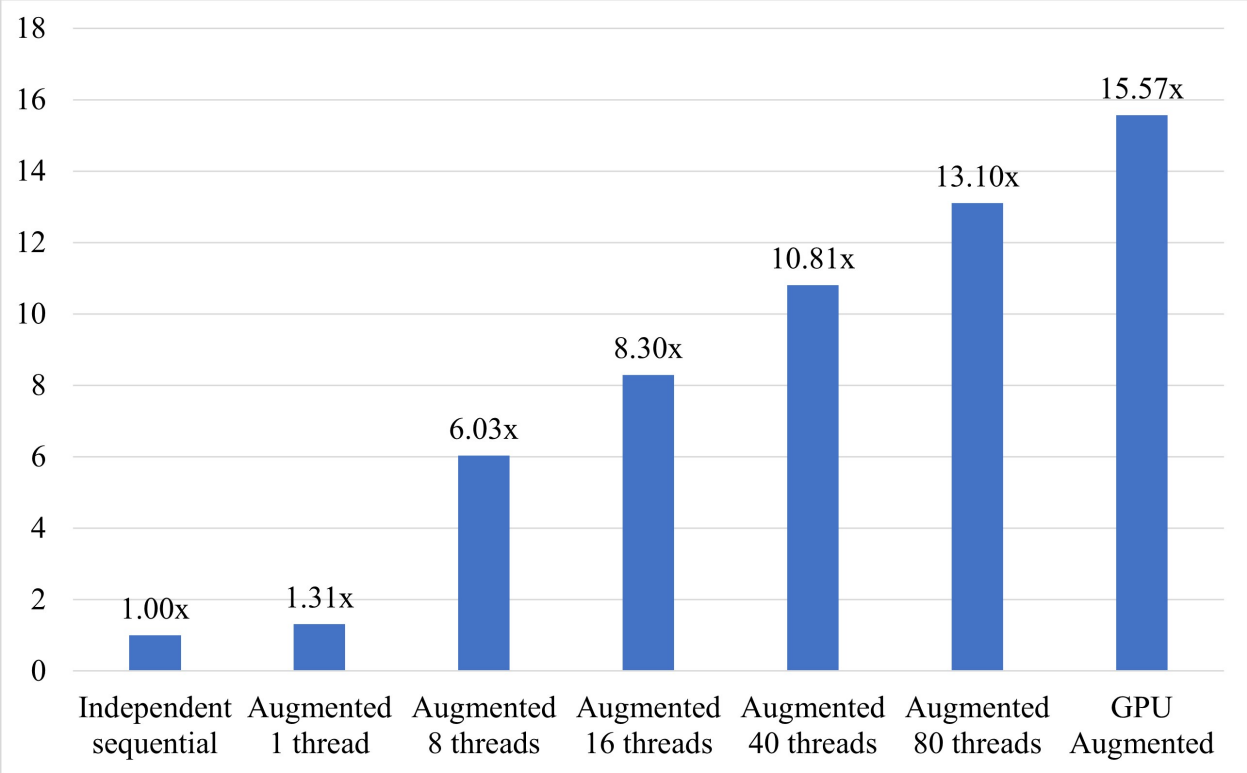


图 15 C语言与 CUDA® 实现的加速比比较。

共享内存，从而可能允许使用更大的线程块大小。

内核性能通过性能分析过程得出的三个数值进行概括，即计算操作的平均多处理器占用率（表 4 和表 5 中的**计算**）、共享内存利用率（表 4 和表 5 中的**共享内存**）以及内存吞吐量¶¶（表 4 和表 5 中的**吞吐量**）。尽管性能分析工具能提供更详细的信息，但这些性能分析指标已足以充分描述内核性能。此外，还展示了 cuBLAS® 中用于 PC方法所包含的矩阵乘法的内核性能分析指标，包括矩阵-矩阵乘积（dgemm）、矩阵-向量乘积（dgemv）和逐元素求和（双精度 AXPY 运算），以便与广为人知且经过高度优化的库函数进行性能比较。此外，还将性能分析活动中测得的内核运行时间纳入比较，以突出两种受测增广系统规模之间的差异。

表4展示了由所有 13509 个待传播状态构成的增广系统的内核性能分析结果。计算受限内核很容易辨认，它们最充分地利用了 GPU 的计算能力（动力学（dynamics）、dgemm、误差计算（errCompute）和最大误差归约（maxReduce））。矩阵-矩阵乘法比动力学函数更受内存限制，因为必须同时将更大的数组加载到

¶¶对于计算密集度不高但仍具有高度并行性的任务，内存传输可能成为最终瓶颈；此指标用于衡量 CPU-GPU 通信带宽的利用率。

***为简洁起见，内核以 cuBLAS® API 函数的名称标识，但针对每种特定的 GPU 架构和问题规模，都会调用经过优化的内核。

共享内存中才能执行计算。相反，动力学内核的计算瓶颈更为突出，因为与 dgemm 中简单的乘法和求和相比，其算法复杂得多。其余内核以及强度较低但仍高度可并行化的误差计算，都具有较高的内存吞吐量。所用 GTX 1050 显卡的最大内存带宽为 112.1 吉比特每秒：吞吐量越接近这一数值，内存事务的管理效果就越好，这对内存受限问题至关重要。如果在内核中添加一些计算，就不可避免地会损失部分吞吐量，因为这会在与共享内存或全局内存之间进行事务传输时引入延迟来源。

表4 13509 大小增广状态内核性能分析结果。

内核	计算 [%]	共享内存 [%]	吞吐量 [吉比特每秒]	运行时间 [μs]
动力学	97.11	24.96	1.91	137470
dgemm	94.15	26.67	4.06	164210
dgemv	47.14	84.54	93.24	1400
双精度 AXPY 运算	18.18	74.78	60.26	32.48
误差计算	99.41	53.26	59.36	4740
最大值归约	36.80	71.36	79.25	279.36

表5展示了由1501个待传播状态构成的增广系统的内核性能分析结果†††。与表4中包含13509个状态的单一增广状态情况相比，计算受限程度较高的内核性能并未显著下降，而其他内核则出现了性能下降。这凸显了GPU计算适用于强度极高且可并行化的任务；在这类任务中，大规模任务并行化能够大幅抵消内核调用开销。不过，成功处理规模较小的问题的能力对于提高程序的灵活性仍至关重要，尤其是在采用所提出的两级增广方案时，该方案包含彼此差异显著的较小子系统。

表5 规模为1501的增广状态内核性能分析结果。

内核	计算 [%]	共享内存 [%]	吞吐量 [吉比特每秒]	运行时间 [μs]
动力学	96.86	24.92	1.91	15300
dgemm	93.67	26.65	4.08	18390
dgemv	42.59	78.74	87.53	170.85
双精度 AXPY 运算	9.19	33.95	36.31	6.62
误差计算	98.25	53.44	59.78	529.34
最大值归约	26.29	63.35	68.20	42.88

表6比较了CUDA® 程序在启用1条和10条流时的运行时间。从理论上讲，

†††此数值是将包含 13509 个状态的完整增广系统拆分为 10 个 CUDA® 流后得出的；除最后一个流外，每个流所含状态数均等于 13509 除以 9 的整数商。第十个也是最后一个流所含的状态数等于前述整数除法的余数。

前者在更充分利用GPU资源方面具有优势，而后者则更积极地利用CPU-GPU并发。尽管GPU效率较低，两者实现的运行时间几乎相同。因此，两级增广方案能够高效处理低层增广子系统大小不一的情况，展现了所提出计算方案的灵活性，尽管要成功实现该方案，需要进行细粒度代码优化。

表6 1条流和10条流的CUDA® 程序执行。每种情况取10次不同运行的平均值。

Case	运行时间 [s]
1条流	15.78
10条流	15.84

七、结论

本文探讨了高性能CPU集群和GPU计算架构为大量初始条件的短程轨道传播带来的优势。所测试的案例研究了设计类似太阳轨道器与金星共振相位所需的运行，这种相位经过优化，可在相对论性N体环境中“冲浪”；本文提出了一种两级增广策略，并使用C语言和CUDA® 编程语言实现。

无论在串行还是并行化情况下，传播增广系统的性能始终优于独立传播各条轨迹。增广带来的优势体现在两个方面：一是相较于反复独立运行，开销更低；二是能够采用更细粒度的并行化，并利用经过优化的线性代数库。

该方法具有良好的可扩展性，因此能够在GPU架构上实现；即使是低端和老旧的显卡，也已能达到与普通规模集群节点相当的性能。数据中心显卡型号可使算法运行速度提升约400至500倍，而最新的游戏GPU应已能让程序加速60倍。此外，实现带误差反馈的修正PC方法二阶版本，预计还能进一步提速两到三倍。

尽管本文通过一个源自轨迹优化应用的测试案例展示了该方案，但它实际上是一种完全通用的轨道传播器。任何需要传播大量初始条件的应用，都能受益于该算法所带来的高计算效率；无需依赖超级计算设施，使用成本低得多的显卡即可。

未来的工作可继续开发该软件工具，依次处理更长的积分时段，并作为通用传播器实现掠飞探测程序。同时，可修改用于实现所提出的轨迹设计应用的优化方案，以适应大规模并行搜索方法。在实现方面，相较于外层的基于流的管理，使用 CUDA® 图的优势

也可以研究增广系统。

致谢

这项研究获得了欧洲研究理事会（ERC）通过欧盟“地平线2020年”研究与创新计划提供的资助，作为COMPASS 项目的一部分（资助协议编号 679086），网址：www.compass.polimi.it；此外，还获得了欧洲航天局（ESA）通过开放空间创新平台（OSIP）共同资助的研究项目“考虑一般演化不确定性的鲁棒轨迹设计”的资助，合同编号：4000135476/21/NL/GLC/my。

参考文献

[1] 欧洲航天局（ESA），“太阳轨道器定义研究报告（红皮书）”，技术网址：https://sci.esa.int/cosmic-vision/ 2011年7月 . Rep.

[2] 欧洲航天局（ESA），“木星冰卫星探测器：探索气态巨行星周围宜居世界的出现。定义研究报告。”技术报告1.0，2014年。网址：https://sci.esa.int/web/juice/-/54994-juice-definition- study-report。

[3] COSPAR——空间研究委员会，“COSPAR行星保护政策”，《今日空间研究》，第208卷，2020年，第10–22页。https://doi.org/https://doi.org/10.1016/j.srt.2020.07.009，网址：https://www.sciencedirect.com/science/article/pii/ S1752929820300372。

[4] Kminek, G.，“ESA行星保护要求，技术报告 ESSB-ST-U-001”，技术报告，欧洲航天局，2012年。

[5] Jehn, R.，“估算阿丽亚娜上面级的撞击概率”，技术报告，MAS工作论文601，欧洲航天局，2014年12月。

[6] Wallace, M .，《行星保护轨迹分析与设计的海量并行贝叶斯方法》，*2015年AAS/AIAA天体动力学专家会议论文集*，第AAS 15-535 卷，美国科罗拉多州韦尔，2015年。网址：https://trs.jpl.nasa. gov/handle/2014/45859。

[7] Colombo, C.、Letizia, F. 和VanDerEynde,J.，《SNAPPshot ESA 行星保护合规性验证软件最终报告 V1.0，技术报告 ESA-IPL-POM-MB-LE-2015-315》，技术报告，南安普顿大学，2016年。

[8] Romano, M.，Rosacco, M.，Colombo, C.，“使用蒙特卡罗直线抽样和子集模拟计算近地天体的撞击概率”，天体力学与动力天文学，第 132 卷，第 8 期，2020年，第 42 页。https://doi.org/10.1007/s10569-020-09981-5，网址 https://doi.org/10.1007/s10569-020-09981-5。

[9] Romano, M .，“用于行星保护合规性验证的轨道传播与不确定性建模”，博士 .D. 论文，

米兰理工大学，导师：Colombo、Camilla 和 Sánchez Pérez、José Manuel，2020年2月。
https://doi.org/10.13140/ RG.2.2.19692.80001。

[10] Masat, A., Romano, M., 和 Colombo, C.，《Kustaanheimo–Stiefel变量在行星保护合规性分析中的应用》，《制导、控制与动力学杂志》，第45卷，第7期，2022年，第1286–1298页。https://doi.org/10.2514/1.G006255，网址：https://doi.org/10.2514/1.G006255。

[11] Stiefel, E. L., 和 Scheifele, G.，《线性与正则天体力学》，数学科学基础丛书，施普林格，柏林，1971年。

[12] Masat, A., Romano, M., 和 Colombo, C.，“结合B平面与皮卡-切比雪夫方法，对受摄行星际共振轨迹进行连续设计”，第*31*届 *AAS/AIAA*航天飞行动力学会议，第AAS-21-289卷，美国北卡罗来纳州夏洛特，2021年。

[13] Masat, A., 和 Colombo, C.，“B平面与Picard–Chebyshev积分方法：在行星际多次飞掠轨迹中应对复杂轨道摄动”，《宇航学报》，第194卷，2022年，第216–228页。https://doi.org/10.1016/j.actaastro. 2022.01.045，网址 https://www.sciencedirect.com/science/article/pii/S0094576522000546。

[14] Junkins, J. L., Bani Younes, A., Woollands, R. M., 和 Bai，X.，“皮卡迭代、切比雪夫多项式与切比雪夫-皮卡方法：在天体动力学中的应用”，《航天科学杂志》，第60卷，第3期，2013年，第623–653页。https://doi.org/10.1007/s40295-015-0061-1，网址 https://doi.org/10.1007/s40295-015-0061-1。

[15] Koblick, D., , and Shankar, P., “用于高精度轨道传播的修正 Picard-Chebyshev 方法评估”，《航空航天工程杂志》，第 28 卷，第 5 期，2015年，第 04014125 页。https://doi.org/10.1061/(ASCE)AS.1943-5525.0000463，网址：https://doi.org/10.1061/(ASCE)AS.1943-5525.0000463。

[16] Woollands, R. M., Bani Younes, A., and Junkins, J. L., “利用正则化和皮卡迭代求解受扰兰伯特问题的新方法”，《制导、控制与动力学杂志》，第 38 卷，第 9 期，2015年，第 1548–1562 页。https://doi.org/10. 2514/1.G001028，网址：https://doi.org/10.2514/1.G001028。

[17] Woollands, R. M., Read, J. L., Probe, A. B., and Junkins, J. L.,《利用特解法和皮卡迭代求解受扰兰伯特问题的多圈解》，《航天科学杂志》，第 64 卷，第 4 期，2017 年，第 361–378 页。https://doi.org/10.1007/s40295-017-0116-6，网址：https://doi.org/10.1007/s40295-017-0116-6。

[18] Woollands, R. M., Read, J., Hernandez, K., Probe, A., and Junkins, J. L.,《面向空间态势感知大规模并行应用的统一兰伯特工具》，《航天科学杂志》，第 65 卷，第 1 期，2018 年，第 29–45 页。https://doi.org/10.1007/s40295-017-0118-4，网址：https://doi.org/10.1007/s40295-017-0118-4。

[19] Swenson, T.、Woollands, R.、Junkins, J. 和 Lo, M.，《将修正切比雪夫皮卡迭代应用于微分修正，以提高鲁棒性并缩短计算时间》，《宇航科学杂志》，第64卷，第3期，2017年，第267–284页。https://doi.org/10.1007/s40295-016-0110-4。

[20] Singh, S. K.、Woollands, R.、Taheri, E. 和 Junkins, J.，《准冻结近极地极低高度月球轨道的可行性》，《宇航学报》，第166卷，2020年，第450–468页。https://doi.org/https://doi.org/10.1016/j.actaastro.2019.10.037，网址 https://www.sciencedirect.com/science/article/pii/S0094576519313657。

[21] Koblick, D., Xu, S., Fogel, J. 和 Shankar, P.，《使用修正 Picard-Chebyshev 方法通过冲量离散化进行低推力最短时间轨道转移非线性优化》，《工程与科学计算机建模》，第 111 卷，第 1 期，2016 年。https://doi.org/10.3970/cmes.2016.111.001，网址 https://doi.org/10.3970/cmes.2016.111.001。

[22] Macomber, B., Probe, A. B., Woollands, R., Read, J. 和 Junkins, J.L.，《提升摄动轨道传播中修正 Chebyshev-Picard 迭代效率》，《工程与科学计算机建模》，第 111 卷，第 1 期，2016 年，第 29–64 页。https://doi.org/10.3970/cmes.2016.111.029，网址 http://www.techscience.com/CMES/v111n1/27313。

[23] Woollands, R., Taheri, E., and Junkins, J. L., “高效计算低推力引力摄动轨道转移的最优解”，《航天科学杂志》，第67卷，第2期，2020年，第458–484页。https://doi.org/10.1007/s40295-019-00152-9，网址：https://doi.org/10.1007/s40295-019-00152-9。

[24] Woollands, R., and Junkins, J. L., “通过自适应 Picard-Chebyshev 迭代求解非线性微分方程：在天体动力学中的应用”，《制导、控制与动力学杂志》，第42卷，第5期，2019年，第1007–1022页。https://doi.org/10.2514/1.G003318，网址：https://doi.org/10.2514/1.G003318。

[25] Atallah, A. M.,Woollands, R.M., Elgohary, T. A., and Junkins, J. L.,“用于传播摄动轨道的六种数值积分器的精度与效率比较”，《航天科学杂志》，第 67 卷，第 2 期，2020年，第 511–538 页。https://doi.org/10.1007/s40295-019-00167-2，网址：https://doi.org/10.1007/s40295-019-00167-2。

[26] Geda,M.,Noomen, R.,andRenk,F.,“使用图形处理器大规模并行化轨迹传播”，硕士论文，代尔夫特理工大学，2019年。网址：http://resolver.tudelft.nl/uuid:1db3f2d1-c2bb-4188-bd1e-dac67bfd9dab。

[27] Schrammel, F., Renk, F., Mazaheri, A., and Wolf, F.,“用于图形处理器上航天器轨迹仿真的高效星历模型”，*Euro-Par 2020*：并行处理，编者：M. Malawski 和 K. Rzadca，施普林格国际出版公司，Cham，2020年，第561–577页。

[28] Li, J.-S., Yang, Z., and Luo, Y.-Z.,“空间物体碰撞概率计算方法综述”，天体动力学，第6卷，第2期，2022年，第95–120页。https://doi.org/10.1007/s42064-021-0125-x，网址：https://doi.org/10.1007/s42064-021-0125-x。

[29] Yoshikawa, K., Sawada, H., Kikuchi, S., Ogawa,N.,Mimasu, Y., Ono,G., Takei,Y.,Terui, F., Saiki,T.,Yasuda,S., Matsushima, K., Masuda, T., and Tsuda, Y.,“隼鸟2号着陆建模与分析”，天体动力学，第4卷，第2期，2020年，第119–135页。https://doi.org/10.1007/s42064-020-0073-x，网址：https://doi.org/10.1007/s42064-020-0073-x。

[30] Armellin, R., Di Lizia, P., Bernelli-Zazzera, F. 和 Berz, M.，《利用微分代数表征小行星近距离交会：阿波菲斯案例》，《天体力学与动力天文学》，第 107 卷，第 4 期，2010年，第 451–470 页。https://doi.org/10.1007/s10569-010-9283-5，网址：https://doi.org/10.1007/s10569-010-9283-5。

[31] Hairer，E.、Wanner，G. 和 Nørsett，S. P.，《常微分方程求解 I》，第 2 版，Springer Berlin，1993年。https://doi.org/10.1007/978-3-540-78862-1。[32] Bai，X. 和 Junkins, J. L.，《利用 NVIDIA GPU 通过 Picard-Chebyshev 方法求解初值问题》，《航天科学进展》，2010年。[33] Rivlin，T. J.，《切比雪夫多项式》，《计算数学》，第 30 卷，1976年。https://doi.org/10.2307/2005983。[34] Fukushima，T.，《皮卡迭代方法、切比雪夫多项式逼近与动力学运动的全局数值积分》，《天文杂志》，第 113 卷，1997年，第 1909–1914 页。https://doi.org/10.1086/118404。

[35] Macomber，B. D.，“针对一般摄动轨道和受约束动力学系统的 Chebyshev-Picard 迭代效率增强”，博士学位论文，得克萨斯农工大学，导师：Junkins, John L.，2015年8月。网址：https: //hdl.handle.net/1969.1/155745。

[36] Fukushima，T.，“通过 Picard-Chebyshev 方法进行动力学运动的向量积分”，《天文杂志》，第113卷，1997年，第2325页。https://doi.org/10.1086/118443。

[37] Bai，X. 和 Junkins, J. L.，“用于轨道传播的改进的切比雪夫-Picard 迭代方法”，《航天科学杂志》，第58卷，第4期，2011年，第583–613页。https://doi.org/10.1007/BF03321533，网址 https://doi.org/10.1007/BF03321533。

[38] Koblick, D., Poole，M.和Shankar, P., “使用修正 Picard-Chebyshev 方法进行并行高精度轨道传播”，*ASME* 国际机械工程大会与展览会，论文集（*IMECE*），2012年。https: //doi.org/10.1115/IMECE2012-87878。

[39] Forum, M. P., “MPI：消息传递接口标准”，技术报告，美国，1994年。

[40] Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D.和McDonald, J., 使用 *OpenMP* 进行并行编程，Morgan Kaufmann，2001年。

[41] “Xianyi/OpenBLAS”，最后访问时间：2021年11月。URL https://github.com/xianyi/OpenBLAS# supported-cpus-and- operating-systems。

[42] Wang, Q., Zhang, X., Zhang, Y.和Yi, Q., “AUGEM：在 x86 CPU 上自动生成高性能稠密线性代数内核”，*SC '13*：国际高性能计算、网络、存储与分析会议论文集，2013年，第1–12页。https://doi.org/10.1145/2503210.2503219。

[43] Xianyi, Z., Qian, W. 和 Yunquan, Z.，《面向龙芯3A处理器的三级基本线性代数子程序性能模型驱动优化》，*2012*年*IEEE*第*18*届并行与分布式系统国际会议，2012年，第684–691页。https://doi.org/10.1109/ ICPADS.2012.97。

[44] 英伟达公司，《CUDA C语言++ 编程指南》，2021年11月。网址：https://docs.nvidia.com/cuda/cuda-c-programming- guide/index.html。

[45] Seidelmann, P. K.,《天文年历说明补编》，University Science Books，美国纽约州梅尔维尔市沃尔特·惠特曼路1305号，110室，邮编11747，1992年。

[46] Valsecchi, G. B., Milani, A., Gronchi, G. F., 和 Chesley, S. R.，“近距离接近的共振返回：解析理论”，《天文学与天体物理学》，第 408 卷，第 3 期，2003 年，第 1179–1196 页。https://doi.org/10.1051/0004-6361:20031039，网址 https://doi.org/10.1051/0004- 6361:20031039。