doi: 10.7690/bgzdh.2020.08.006

并行计算在机动飞行轨迹生成中的应用

蒋　超，王维嘉，王　昊
(航空工业西安飞行自动控制研究所飞控部，西安 710065)

摘要：针对现有通用机动轨迹需要较长的预规划时间，无法在机载计算平台实时解算的问题，提出一种利用并行计算的方式对通用机动框架进行加速的方法。对现有的 MCTS 算法叶子节点并行、根节点并行和树并行方式进行分析，结合叶子节点并行和根节点并行方式各自的优点，对每棵搜索树采用叶子节点并行方法，分别利用 Pthread 和 CUDA 对并行通用机动框架进行加速，并以筋斗机动为例对加速效果进行测试。实验结果表明：并行通用机动框架不仅性能优于串行框架，而且可大幅缩短机动解算时间。

关键词：并行计算；蒙特卡罗树搜索算法；GPU；众核；通用机动框架
中图分类号：TP301　　**文献标志码：**A

Application of Parallel Computing in Production of Maneuvering Flight Trajectory

Jiang Chao, Wang Weijia, Wang Hao
(*Department of Flight Control*, *AVIC Xi'an Flight Automatic Control Research Institute*, *Xi'an* 710065, *China*)

Abstract: Aiming at longer pre-planning time for the existing general maneuvering trajectory, it is impossible to solve the problem of real-time solution in the airborne computing platform, and propose a method for accelerating the general maneuvering framework by using parallel computing. Analyzes the leaf node parallel, root node parallel and tree parallel of the existing MCTS algorithm, combined with the advantages of leaf node parallel and root nodes parallel, applied the leaf node parallel method for each search tree, and used the Pthread and Cuda to accelerate the parallel general maneuvering framework, and test the accelerated effect with the example of the maneuvering force. The experimental results show that the parallel maneuvering framework is not only better than the serial frame, but also can shorten the time of the maneuver calculation.

Keywords: parallel computing; Monte Carlo tree search algorithm; GPU; multi-core; general maneuvering frame

0　引言

采用强化学习的思想，基于 MCTS(蒙特卡罗搜索树)算法的通用机动控制框架在无人机动作空间反复试错，最终搜索出一条能实现期望的机动参考轨迹。该框架充分挖掘了无人机的机动潜能，有利于无人机更好地完成既定任务。由于 MCTS 算法是一种基于大量仿真抽样的搜索算法，需要较长的预规划时间，导致通用机动框架无法在机载计算平台实时解算；因此，笔者对该飞行机动轨迹框架的实时性进行改进。

1　现有的通用机动控制框架

现有的通用机动框架[1]以 MCTS 算法为核心，由飞机模型、控制律模型、抽样器、评价器、决策器等部分组成。其系统结构[2]如图 1 所示。

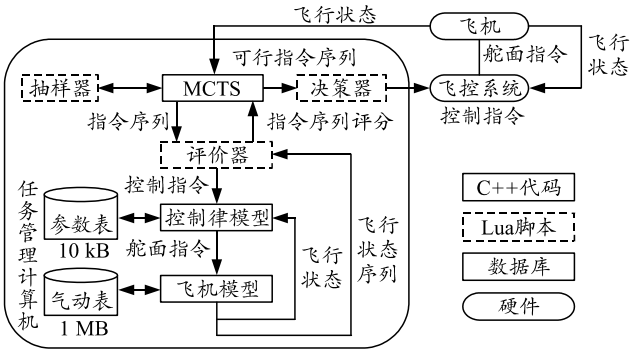

图 1　通用机动框架系统结构

通用机动框架各部分作用如下：

1) 飞机模型：该部分为一个六自由度飞机数字模型，需要加载飞机气动数据。利用飞机当前状态，插值得到飞机气动系数，从而计算得到气动力与气动力矩。根据飞机动力学方程，可计算出飞机运动状态。

收稿日期：2020-03-20；**修回日期：**2020-05-15
作者简介：蒋　超(1995—)，男，陕西人，硕士，从事智能空战与并行计算研究。E-mail: 1595863239@qq.com。

2) 控制律模型：以传统的 PID 控制律为基础，对飞机纵向与横航向进行控制，增强飞机稳定性和操纵性，同时，对飞机飞行品质进行改善。

3) 抽样器：在飞机动作空间进行抽样，获得指令序列，抽样方法选择随机抽样。

4) 评价器：对抽样得到的每一个指令序列进行评价，利用评分函数计算该指令下飞机状态序列对应的分值。

5) 决策器：在所有抽样指令序列集中选取评分最高的指令序列作为飞机实际的控制指令。

针对给定的机动动作，通用机动框架确定相应的动作空间。抽样器在该动作空间进行抽样，结合现有的最优树内搜索路径，得到机动指令序列。评价器结合飞机模型和控制律模型，将该机动指令序列作为飞机模型的输入，利用数字仿真的方式，得到该机动指令下飞机在飞行过程中所有状态向量的变化情况，根据评分函数，计算得到该指令序列的评分值。最后，评分值将通过反向传播的方式，对最优树内搜索路径上节点的评分值进行更新。

基于 MCTS 算法的通用机动框架利用蒙特卡罗模拟的方法[3]，对搜索树内新增节点评分值进行估计，其对问题解空间的搜索程度随树内节点数量的增加而增加。然而，现有的串行通用机动框架，算法每循环一次，只能对一个节点评分值进行估计，同时，每一次只能新增一个树内节点，使得算法对解空间的搜索速度较慢。以筋斗动作为例，在不加入宏动作的情况下，飞机机动时间 30 s，机动解算步长设置为 0.02 s，机动指令序列采样时间为 0.1 s，搜索深度为 300 层，对筋斗动作求解时，问题解空间大小为 4^{300}。加入宏动作后，虽可将问题搜索深度减小到 20 层，问题解空间大小减小为 4^{20}；但短时间内，仍然无法实现对解空间的充分搜索。

2 MCTS 并行化改造

MCTS 并行化方法如图 2 所示。

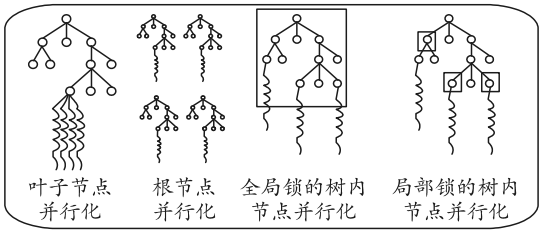

图 2 MCTS 并行化方法

2.1 叶子节点并行化

叶节点并行化最简单的一种并行化方法，主线程在树内游走，到达树内叶子节点，得到树内最优节点序列，然后为该叶子节点添加 n 个新节点，当该叶子节点的所有儿子节点数目小于 n 时，在其子节点下继续添加新节点(以上完成了 MCTS 算法选择和扩展步骤)。从所有新添加的节点开始，重复随机在动作空间选择动作，直到达到预设的搜索深度。将 n 个新节点分配给 n 个线程，得到所有新节点评分值。经由主线程将所有的分值依次为对应树内最优节点序列中的节点信息进行更新(反向传播)。

2.2 根节点并行化

根节点并行是在每个线程上构建一棵搜索树，线程之间独立工作。当到达预定的搜索时间(或搜索次数)，搜索停止，将所有线程的搜索树结果进行汇总。合并后的搜索树，相同位置的节点，被访问次数为所有树相同位置访问次数之和，节点平均分数则以加权平均的方式进行计算。根据合并后的搜索树，给出待求解问题的搜索结果。这种并行方法，线程间通信量最少，实现起来比较容易。多棵搜索树在问题的求解空间多个点同时进行搜索，更容易找到全局最优解。

2.3 树节点并行化

树节点并行方法只构建了 1 棵共享的搜索树，多个线程同时对搜索树进行维护(更新节点信息)，且分别重复进行选择、扩展、模拟、反向传播 4 种操作。

3 Pthread 和 CUDA 下通用机动框架改造

在文中 MCTS 算法并行化的方法中，分别利用 Pthread 和 CUDA 对现有的通用机动框架进行改造，缩短框架解算时间。

3.1 基于 Pthread 的并行化通用机动框架改造

Pthread(POSIX Threads)原本为 Linux 系统上实现多线程的线程接口，随着 Windows 系统下多线程 API 库的开发，使得基于 Windows 平台的多线程程序开发更为便捷。传统的进程只包含单个控制线程。该线程只在一个处理器核心上运行，造成多核处理器计算资源的极大浪费。多线程工作原理如图 3 所示，通过将一个进程的计算任务分配给多个线程，每个线程可在一个处理器核心上独立执行，可实现对多核处理器计算资源的充分利用，同时有效减少程序运行时间。

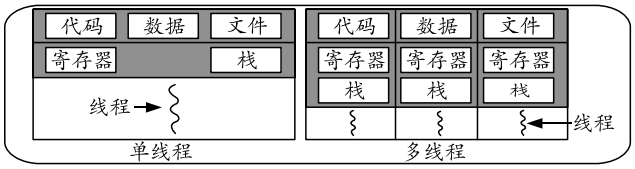

图 3　多线程工作原理

MCTS 并行化算法有叶子节点并行、根节点并行、树节点并行以及笔者提出的块并行方式。新框架系统结构如图 4 所示。鉴于当前实验室多核处理器核心数的限制(4 核 8 线程)，笔者采用叶子节点并行的方式实现 MCTS 并行化。

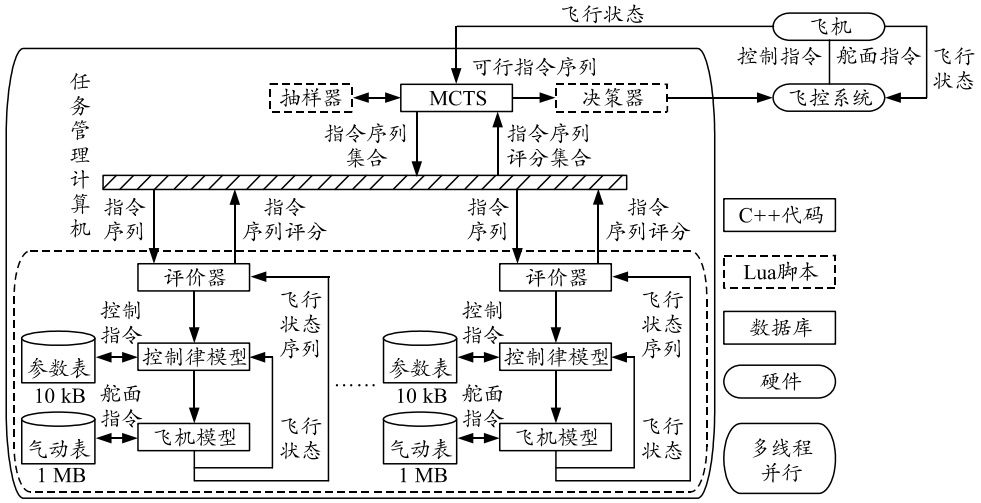

图 4　多核处理器并行的通用机动框架系统结构

根据 MCTS 算法叶子节点并行的思路，扩展阶段每次增加多个新节点，一个新节点可得到一条指令序列；因此，每一次树搜索过程，可得到多条指令序列，组成指令序列集合，再由主线程将指令序列集合分发给每个子线程。每个处理器核心上的线程对分到的指令序列进行模拟，最后将评分序列汇总到主线程。相比原有的串行通用机动框架，新框架一次可对解空间中多个点进行搜索，并对多条指令序列进行评价，提高了算法搜索速度。

3.2　基于 CUDA 的并行化通用机动框架改造

基于 CUDA 的并行 MCTS 通用机动框架系统结构如图 5 所示。由于 CPU 处理复杂逻辑能力较强，并行 MCTS 算法选择、扩展与反向传播等过程在 CPU 端完成，利用叶子节点并行、根节点并行或块并行的方式，在 CPU 端可生成多条机动指令序列，组成指令序列集合。将指令序列集合发送到 GPU 端，借助 GPU 强大的多线程并行计算能力，利用给定的指令序列，根据飞机气动方程对飞机机动过程进行数字模拟，根据相应的飞行状态序列对该指令序列进行评分，计算出对应的指令序列评分。GPU 内所有线程计算完成后，将评分结果汇总到 CPU 端，依次将每一条指令序列与相应评分作为反馈，对搜索树节点信息进行更新。利用 GPU 同时对多条指令序列进行模拟评分，有效减小了模拟过程平均执行时间，从而达到缩短框架解算时间的目的。

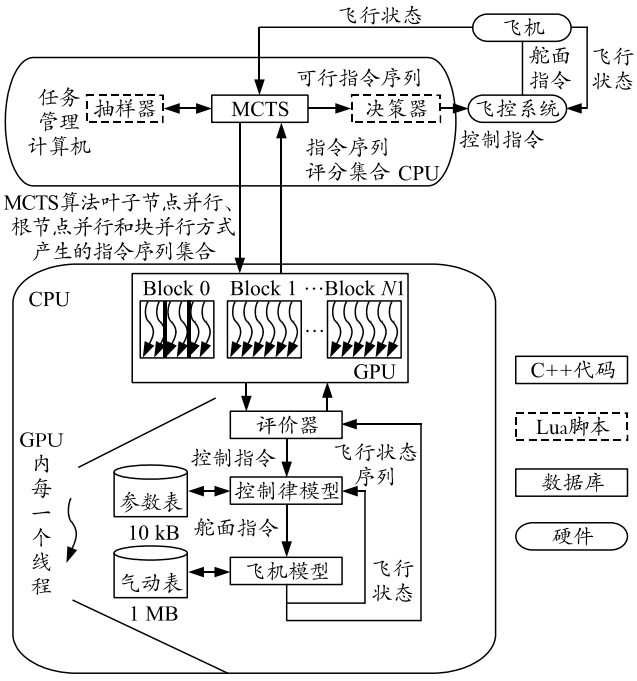

图 5　基于 GPU 并行的通用机动框架

4　实验验证

针对提出的基于 Pthread 和 CUDA 的并行通用机动框架，笔者分别对 2 种方案进行机动仿真实验，以测试并行框架搜索性能和加速效果。

4.1　Pthread 加速实验

4.1.1　飞机模型

实验以开源的 F16 飞机为模型。

4.1.2 机动指令设置

实验中，笔者设定飞机初始状态为在 3 km 高度处，以 230 m/s 速度做水平直线飞行，接收到机动指令后，利用基于 Pthread[4]的并行通用机动框架求解出满足机动要求的参考轨迹。

实验以筋斗指令为例进行测试。为了对每一次模拟进行评价，笔者根据机动终止状态 S^*制定机动评价函数如下：

$$f_{\text{eval}}(S^*)=\begin{cases}|360-\theta|+100 & (\theta<360°)\\ k_1*\Delta x+k_2*\Delta h+k_3*\Delta t & (\theta>360°)\end{cases}。$$

其中：θ 为飞机俯仰角；Δx 为飞机筋斗过程前飞距离；Δh 为飞机筋斗过程高度变化量；Δt 为筋斗过程持续时间。上述变量定义如图 6 所示。在实际筋斗机动中，应使 Δx、Δh、Δt 越小越好；因此，筋斗机动求解为一个多目标优化问题，通过加权求和的方式，赋予每个变量不同的权重 k_1、k_2、k_3。

将问题转化单目标优化问题，使得最终评价函数值 $f_{\text{eval}}(S^*)$越小越好。其中 k_1、k_2、k_3 取值分别为 0.000 5, 0.1, 0.1。

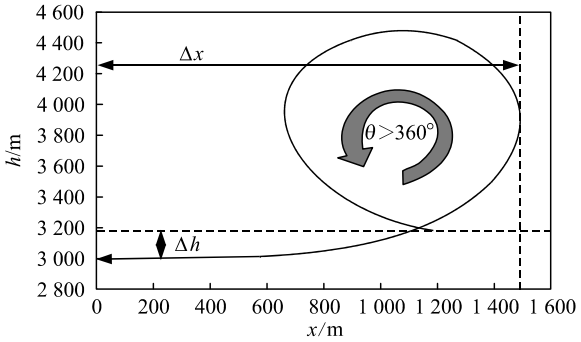

图 6 筋斗动作评价变量定义

由于筋斗是纵平面机动，增加油门开度和对升降舵面进行控制有益于筋斗动作的实现；因此，将飞机操纵动作空间设置为{a_0 a_1 a_2 a_3}。其中：

a_0：增加油门开度，并且升降舵增加；

a_1：增加油门开度，升降舵面不变；

a_2：增加油门开度，升降舵面减少；

a_3：回正动作，油门开度和升降舵面回到了配平值。

4.1.3 实验硬件环境

实验硬件环境如表 1 所示。

表 1 实验硬件环境

参数名称	参数性能
处理器型号	Intel i7-3770 CPU 主频 3.4 GHz 四核八线程
内存	4 GB
操作系统	Windows 7

4.1.4 实验结果分析

笔者采用叶子节点并行的方案，对基于 Pthread 的并行通用机动框架进行加速测试。由于 CPU 为 4 核 8 线程，选取线程数分别为 1、2、4、8，每种情况重复实验 20 次，数据结果取平均值。笔者分别统计实验中评价函数值、程序运行时间和程序加速比(程序运行时间与相同搜索次数时，线程数为 1 的程序运行时间的比值)，实验结果如表 2—6 所示。

表 2 评价函数均值变化情况 min

线程数	搜索次数						
	80	160	320	640	960	1 280	1 600
1	160	120	110	75	42	37	19
2	210	110	76	43	25	22	24
4	175	140	103	49	57	27	22
8	170	147	109	51	49	46	27

表 3 程序运行时间变化情况 s

线程数	搜索次数						
	80	160	320	640	960	1 280	1 600
1	12	16	24	44	70	94	115
2	9	13	18	25	45	57	57
4	5	7	11	18	24	32	42
8	2	5	7	14	19	22	29

表 4 程序加速比变化情况

线程数	搜索次数						
	80	160	320	640	960	1 280	1 600
2	1.6	1.6	1.6	1.6	1.7	1.7	1.7
4	2.4	2.5	2.6	2.7	2.8	2.8	2.8
8	3.1	3.3	3.7	4.0	3.9	4.1	4.0

表 5 不同线程数综合排名

线程数	搜索次数							排名
	80	160	320	640	960	1 200	1 600	总和
1	1	2	4	4	2	2	1	16
2	4	1	1	1	1	1	3	12
4	3	3	2	2	4	3	2	19
8	2	4	3	3	3	4	4	23

表 6 不同线程数并行效率

线程数	平均加速比	并行效率
2	1.6	0.800 0
4	2.7	0.675 0
8	3.7	0.462 5

为了综合评价不同线程数对搜索算法性能影响，笔者统计了搜索次数从 80～1 600 实验中，不同线程数在该次实验中评价函数均值排名情况如表 2 所示，利用排名总和对算法性能进行排名比较。由排名总和可看出：当线程数为 2 时，利用叶子节点并行的方式对通用机动框架进行加速，可提升算法性能。当线程数为 4 和 8 时，算法性能有所下降。

结合表 3 可知：随着线程数的增加，程序运行时间显著下降。当线程数为 8 时，随着树搜索次数增加(从 80 增加到 640)，程序加速比显著提高，其原因在于当树搜索次数较少时，程序总运行时间较

短，多线程技术需要在内存空间初始化 8 架飞机实体，其时间开销占程序总运行时间的比重，随着树搜素次数的增加(程序总运行时间增加)而减小。3 种线程数并行效率比较如表 3 所示。

表 6 对 3 种并行方式并行效率(加速比与并行度的比值)进行比较可知，随着线程数增加，程序并行效率降低。由于 Pthread 采用共享内存方式并行，随着线程数目增多，线程间通信时间开销增加，并且线程数越多，已经完成计算的线程等待其他线程完成计算的平均等待时间增加，导致程序并行效率降低。

4.2　CUDA 加速实验

笔者利用 CUDA[5]分别就叶子节点并行、根并行以及块并行方式对通用机动框架进行加速。飞机模型和机动指令设置与 Pthread 实验一致。

利用 CUDA 就叶子节点并行的方式对通用机动框架进行加速，总的树搜索次数分别为 1 024, 2 048, 3 072, 4 096 次，线程数分别为 4, 8, 16, 32，每种情况运行 20 次，结果取平均值，框架性能变化情况如表 7—9 所示。

表 7　评价函数均值变化情况　　min

评价函数	树搜索次数			
	1 024	2 048	3 072	4 096
CPU	48	17	9	12
线程数 4	25	15	10	5
线程数 8	20	18	11	12
线程数 16	32	19	8	10
线程数 32	70	27	12	11

表 8　程序运行时间变化情况　　s

程序运行时间	树搜索次数			
	1 024	2 048	3 072	4 096
CPU	77	150	224	295
线程数 4	65	109	160	220
线程数 8	27	63	82	115
线程数 16	16	27	46	54
线程数 32	11	16	25	31

表 9　程序加速比变化情况

线程数	树搜索次数			
	1 024	2 048	3 072	4 096
4	1.4	1.4	1.4	1.4
8	2.5	2.5	2.6	2.5
16	4.4	4.7	4.7	4.9
32	8.5	8.8	8.7	8.7

利用排名总和对算法性能进行比较可知，当线程数为 4 时，通用机动框架性能有所提升，同时，程序加速比为 1.4。当线程数为 8, 16, 32 时，程序加速比显著提升，但通用机动框架性能随之下降。考虑到引入并行计算的目的是在保证通用机动框架性能的前提下，减小框架运行时间；因此，综合框架性能与加速比因素，选取线程数 16 作为并行方式比较时，叶子节点并行方式典型代表[6]。

4.2.1　根节点并行结果分析

笔者利用根节点并行的方式，对通用机动框架进行加速，总的树搜索次数分别为 1 024, 2 048, 3 072, 4 096 次，线程数分别为 4, 8, 16, 32，每种情况运行 20 次，结果取平均值。

表 10—13 为不同线程数时并行通用机动框架性能随树搜索次数变化情况。利用表 13 所示评价函数均值排名总和对框架性能进行比较可知：当线程数为 8 和 32 时，通用机动框架性能有所提升；考虑到线程数为 32 时，程序加速比为 4.3，比线程数为 8 时，拥有更好的程序加速效果；线程数为 32 时，通用机动框架性能更优。选取线程数 32 作为并行方式比较时，作为根节点并行方式典型代表。

表 10　评价函数均值变化情况　　min

评价函数	树搜索次数			
	1 024	2 048	3 072	4 096
CPU	49	14	9	12
线程数 4	36	13	12	8
线程数 8	17	17	11	7
线程数 16	16	17	13	9
线程数 32	34	13	12	7

表 11　程序运行时间变化情况　　s

程序运行时间	树搜索次数			
	1 024	2 048	3 072	4 096
CPU	75	150	225	290
线程数 4	83	175	270	360
线程数 8	62	110	170	235
线程数 16	34	65	100	130
线程数 32	17	25	50	70

表 12　程序加速比变化情况

线程数	树搜索次数			
	1 024	2 048	3 072	4 096
4	4.1	4.2	4.2	4.1
8	1.3	1.3	1.3	1.3
16	2.2	2.2	2.2	2.2
32	4.1	4.2	4.2	4.1

表 13　不同线程数综合排名

线程数	排名				
8	2	5	2	1	10
16	1	4	5	4	14
32	3	1	3	3	10

根并行方式需要由主线程同时对多棵搜索树进行维护。总树搜索次数不变情况下，相对于线程数为 8、16 和 32 的情况，线程数为 4 时，CPU 与 GPU 通信次数增多，增加了程序时间开销。线程数较小时，程序并行度较低，GPU 利用率较低；因此，当线程数为 4 时候，程序加速比出现小于 1 的情况。

4.2.2 块并行结果分析

笔者利用块并行的方式对通用机动框架进行加速，总的树搜索次数分别为 1 024, 2 048, 3 072, 4 096 次，叶子节点并行度分别为 4, 8, 16, 32，对应的根节点并行度为 4, 8, 16, 32，对框架性能变化情况进行分析。

随着叶子节点并行度和根节点并行度的增加，程序加速比呈增大趋势。利用表 14 所示评价函数均值排名总和对框架性能进行比较可知，根节点并行度为 4，叶子节点并行度为 4 时，通用机动框架性能最好；因此，选取根并行度 4、叶并行度 4 作为 4.2.3 小节并行方式比较时块并行方式典型代表。

表 14 各种并行情况下评价函数均值排名总和

根节点并行度	叶节点并行度			
	4	8	16	32
4	4	12	24	40
8	8	20	36	52
16	16	31	46	59
32	29	46	57	64

4.2.3 并行方式比较

前面小节分别就叶子节点并行、根节点并行和块并行方式对通用机动框架加速结果进行分析，在综合框架性能和程序加速比因素后，各选出一种情况作为该种并行方式的典型代表，分别为：线程数 16 的叶子节点并行方式，线程数 32 的根节点并行方式和根并行度为 8、叶并行度为 8 的块并行方式。笔者分别以上述 3 种并行情况为代表，就叶子节点并行、根并行和块并行 3 种并行方式对通用机动框架性能影响进行对比，对比结果如表 15—17。

表 15 不同并行方式评价函数均值变化情况 min

并行方式	树搜索次数			
	1 024	2 048	3 072	4 096
串行	49.1	14.8	9.7	11.6
叶子节点并行	31.5	16.5	9.4	9.9
根并行	33.6	11.8	10.7	8.3
块并行	26.7	14.4	9.2	9.0

表 16 不同并行方式程序运行时间变化情况 s

并行方式	树搜索次数			
	1 024	2 048	3 072	4 096
串行	75.8	150.7	224.1	296.6
叶子节点并行	17.4	32.2	47.8	61.0
根并行	18.4	36.3	53.8	71.8
块并行	9.6	18.7	27.4	36.3

表 17 不同并行方式程序加速比变化情况

并行方式	树搜索次数			
	1 024	2 048	3 072	4 096
串行	1.0	1.0	1.0	1.0
叶子节点并行	4.4	4.7	4.7	4.9
根并行	4.1	4.2	4.2	4.1
块并行	7.9	8.1	8.2	8.2

由 3 种并行方式在不同树搜索次数下性能变化情况可知：在树搜索次数较低时，3 种并行方式能明显提升通用机动框架性能，随着树搜索次数增加，搜索结果趋于收敛。当树搜索次数为 4 096 次时，串行和叶子节点并行方式评价函数均值不降反增，表明 2 种方法稳定性不如根节点并行和块并行方式，容易陷入局部最优解。

结果表明：块并行的通用机动框架性能最优，其次为根并行方式。由于块并行和根并行方式都同时构建了多棵搜索树，在机动问题解空间从多个点出发开始搜索，增大框架全局搜索能力，陷入局部最优解可能性降低。同时，块并行方式每棵搜索树采用叶子节点并行方式进行搜索，能同时对多个节点进行评价，当问题动作空间较小(筋斗动作数为 4)时候，能对当前节点所有子节点进行搜索，对该分支节点开发程度较高，能以更快速度找到问题最优解。

表 16、表 17 分别就程序运行时间和程序加速比变化情况进行比较。由表 16、表 17 可知，利用并行计算的方式能大幅减少程序运行时间。表 18 对 3 种并行方式并行效率进行比较。由表 18 可知：叶子节点并行效率最高，其次为根并行方式，块并行方式并行效率最低。究其原因，MCTS 算法树内游走过程时间开销远大于叶子节点扩展过程，叶子节点并行每次搜索只用进行一次树内游走，叶子节点扩展时间开销较小。树并行方式每次搜索时，每棵搜索树需要进行一次树内游走，且只扩展一个叶子节点，相对于叶子节点并行方式，树内游走次数更多，程序时间开销增加导致程序并行效率较低。块并行方式同根节点并行一样，同时开发多棵搜索树，程序并行效率同样较低。

表 18 3 种并行方式并行效率比较

并行方式	并行度	平均加速比	并行效率
叶子节点并行	16	4.15	0.26
根并行	32	4.68	0.15
块并行	64	8.10	0.13

4.3 最大加速比测试

由实验可知：块并行的并行通用机动框架性能最优，为此，笔者利用块并行方式，探究 1080Ti GPU 对通用机动框架的最大加速比。

由于 1080Ti GPU[7]中有 3 584 个 CUDA 核心，其最大可并行度为 3 584；因此，笔者设定通用机动框架总树搜索次数为 3 584 次，利用 F16 飞机模型，

以筋斗动作为例，每次搜索重复执行 50 次，实验数据取平均值，实验结果如表 19 所示。

表 19　并行度为 3 584 时加速比实验结果

并行方式	并行度	评价函数均值	程序运行时间/s	程序加速比
串行	1	9.724 3	263.199 2	1.0
树 4 叶 896	3 584	13.630 3	2.486 9	105.8
树 8 叶 448	3 584	10.639 9	2.451 2	107.4
树 16 叶 224	3 584	9.436 7	2.459 8	107.0
树 28 叶 128	3 584	9.195 9	2.459 5	107.0
树 56 叶 64	3 584	8.654 4	2.489 1	105.7
树 112 叶 32	3 584	9.471 8	2.511 5	105.8

由表可知：总树搜索次数为 3 584 次时，串行通用机动框架求解时间较长，需要 263 s，基于 CUDA 并行的通用机动框架，当并行度为 3 584 时，求解时间缩短到 3 s 以内，程序加速比达 100 倍以上。相比于单独对飞机仿真模块加速比最大为 462.57 的情况，基于 CUDA 并行的通用机动框架加速比有所下降，分析其原因如下：

与串行飞机仿真模块相比，串行通用机动框架解算时间变短。二者都需对 3 584 个飞机架次进行机动仿真，且最大仿真步数同为 3 000 步，然而，通用机动框架在机动仿真时，会根据边界保护、目标机动达成标准等对飞机状态进行实时判断，对不符合要求的情况及时终止并舍弃该次仿真；因此，其平均机动仿真步数远小于飞机仿真模块，使得串行通用机动框架解算时间更短。

与并行飞机仿真模块相比，并行通用机动框架解算时间变长。在模拟过程中，GPU 内 3 854 个线程[8]同时开始解算，每个线程解算时间不同，其最终解算时长取决于所有线程中耗时最长的线程。再者，GPU 以单指令多线程的方式运行，在模拟过程，引入逻辑分支对飞机状态进行判断会降低线程束指令执行效率，解算时间增加；因此，并行通用机动框架模拟过程比并行飞机仿真模块耗时更多。同时，还需要对搜索树进行维护和节点信息更新等操作，使得并行通用机动框架耗时更长。

当搜索树数量较少(4 或 8)时候，并行通用机动框架性能略逊于串行通用机动框架；当搜索树数量较多(小于等于 16)时，并行通用机动框架搜索性能优于串行框架。这是因为串行 MCTS 算法容易陷入局部最优解，块并行方式通过构建多棵搜索树，能以更大概率找到问题最优解；因此，搜索树数量较多时，并行通用机动框架性能优于串行框架。

利用 NVIDIA 公司提供的 profiler 性能分析工具[9]对此时 GPU 资源使用情况进行统计。其结果如表 20 所示。

表 20　GPU 资源利用率

资源	资源使用情况
每个线程块共享存储器使用/kB	15.488
全局存储器读取效率/%	31.8
全局存储器存储效率/%	25.2
共享存储器效率/%	20.8
线程束执行效率/%	40.6
计算能力使用率/%	22

如表所示：每个线程块共享存储器使用量为 15.488 kB，相对于最大可用量 96 kB，资源还有很大富裕度，便于后期对框架任务功能进行扩展。由于并行度为 3 584 时，主机端(CPU)和设备端(GPU)间数据传输规模较小，全局存储器在小规模数据传输时实际带宽较低，使得其读取效率较低。GPU 单指令多线程的运行方式使得程序内部逻辑分支的数量会显著影响其指令执行效率，线程数执行效率较低，可通过减少分支数的方式提高其执行效率；因此，并行框架还有进一步优化空间，通过对程序进行优化，加速比还可进一步提高。

综上所述：基于 CUDA 的并行通用机动框架，可在保证框架搜索性能的前提下，有效提高框架解算速度。在 1080Ti GPU 上，加速比可达 100 倍以上。且框架还有优化空间，优化后的框架可进一步提高加速比。

5　结束语

笔者利用 CUDA 分别就叶子节点并行、根节点并行和块并行方式对通用机动框架进行加速，得出以下结论：1) 串行通用机动框架容易陷入局部最优解，根节点并行和块并行的方式，可增大并行框架找到最优解的概率，同时，3 种并行方式在选择合适的并行度时，框架性能都优于串行框架程序；2) 加速后的通用机动框架性能由好到坏依次为块并行方式、根节点并行和叶子节点并行；3) 3 种并行方式程序加速比随程序并行度增加而增大，其中，叶子节点并行方式并行效率明显优于根并行和块并行方式；4) 由加速比实验可知，树搜索次数为 3 584 次时，串行通用机动框架解算时间为 263 s，利用 1080Ti GPU 可有效对通用机动框架进行加速，机动解算时间缩短到 3 s 以内，最大加速比可达 100 倍以上，为通用机动框架在机载计算平台实现实时解算提供了可能。

(下转第 36 页)

[3] 廖飞, 陈捷, 肖云峰. 云计算安全架构及防护机制研究[J]. 通信技术, 2019, 52(10): 2472−2482.
[4] 习阳, 李凯, 王潇. 军工行业工业控制系统信息安全风险与对策[J]. 兵工自动化, 2019, 38(9): 13−15.
[5] 李明, 曲洁. 《信息系统安全等级保护定级指南》修订要点解析[J]. 信息网络安全 2016(S1): 19−21.
[6] 马力, 祝国邦, 陆磊. 《网络安全等级保护基本要求》(GB/T 22239−2019) 标准解读[J]. 信息网络安全, 2019(2): 77−84.
[7] 国家市场监督管理总局, 中国国家标准化管理委员会. 信息安全技术 网络安全等级保护基本要求: GB/T 22239−2019[S]. 北京: 中国标准出版社, 2019.
[8] 国家市场监督管理总局, 中国国家标准化管理委员会. 信息安全技术 网络安全等级保护安全设计技术要求: GB/T 25070−2019[S]. 北京: 中国标准出版社, 2019.

****

（上接第 21 页）

[10] ZHAO H P, FU X G, GAO M G, et al. Research on the visibility of low-orbit debris using space-borne radar[J]. IET Radar, Sonar & Navigation, 2015, 9(1): 31−37.
[11] OLIVER M, EBERHARD G. 卫星轨道−模型、方法和应用[M]. 北京: 国防工业出版社, 2012: 34−35.
[12] MERRILL I. Skolnik. 雷达手册 [M]. 3 版. 北京: 电子工业出版社, 2010: 1069.
[13] Wikipedia. North American Aerospace Defense Command [OL]. (2018−07−09)[2019−10−25]. https://en.wikipedia.org/wiki/North_American_Aerospace_Defense_Command.
[14] 刘兴. 防空防天信息系统及其一体化技术[M]. 北京: 国防工业出版社, 2009: 81−90.
[15] Wikipedia. Boeing E-3 Sentry [OL]. (2018−06−01)[2019−10−25]. https: // en. wikipedia. org/ wiki/ Talk: Boeing_E-3_Sentry.
[16] 王群. 美国新一代导弹预警卫星系统及其能力分析[J]. 国防科技, 2012(2): 7−12.

****

（上接第 31 页）

参考文献:

[1] 刘佩. 空战机动飞行仿真研究[C]. 中国自动化学会控制理论专业委员会, 第 37 届中国控制会议论文集, 中国自动化学会控制理论专业委员会: 中国自动化学会控制理论专业委员会, 2018: 5.
[2] 陈向, 王维嘉, 魏文领, 等. 基于蒙特卡罗搜索树的自动飞行机动[C]//2016 年航空科学与技术全国博士生学术论坛, 西安: 西北工业大学研究生院, 2016.
[3] 林清, 梁争争, 许少尉. 基于自主可控的机载嵌入式计算机现状与展望[J]. 航空计算技术, 2018(5): 2−4.
[4] 牛文生, 王乐. 机载计算技术的新进展[J]. 航空科学技术, 2012(4): 1−4.
[5] "Nvidia Workstation Products" [Z]. Nvidia.com.Retrieved October 2, 2007.
[6] 薛杨, 孙永荣, 赵科东, 等. 基准地图测绘下的视觉导航算法[J]. 兵工自动化, 2019, 38(10): 22−27.
[7] LIU Y, JIAO S, WU W, et al. GPU accelerated fast FEM deformation simulation[C]//IEEE Asia Pacific Conference on Circuits & Systems. IEEE, 2008.
[8] PIRJAN A. Improving Software Performance in the Compute Unified Device Architecture[J]. Informatica Economica, 2010, 14(4): 2−3.
[9] CHU A, FU C W, HANSON A, et al. GL4D: A GPU-based Architecture for Interactive 4D Visualization[J]. IEEE Transactions on Visualization & Computer Graphics, 2009, 15(6): 1587−1594.