亚马逊国际象棋蒙特卡洛树搜索算法的 GPU 加速及其评估

函数

Yikai 太阳

北京信息科技大学计算机学院 BISTU

中国,北京

sunyikai2021@outlook.com

明高

北京信息科技大学计算机学院 BISTU

中国,北京

gaomingshsf@ hotmail.com

东元

北京信息科学与计算机学院

北京工业大学 BISTU，中国北京

YUANDONG20211@outlook.com

Penghui 朱

北京信息科学与计算机学院

北京工业大学 BISTU，中国北京

941509604 @qq.com

***摘要-评估函数是亚马逊国际象棋蒙特卡洛树搜索 (MCTS)算法中非常重要的一部分。*该文介绍了蒙特卡罗树搜索算法的原理，并基于 CUDA 计算平台，实现了对亚马逊棋态势评估函数的 GPU 加速计算。利用 CPU 代码和 CUDA 代码进行了两个实验，分别评估了函数计算精度和核函数调用参数对 GPU 加速性能的影响。实验结果表明，该评估函数对于 GPU 加速是可行有效的，GPU 对 CPU 的加速比达到 20.0。**

Keywords-component;亚马逊国际象棋;人工智能;蒙特卡洛搜索树

我的介绍。

亚马逊国际象棋是阿根廷人 Walter Zamkauskas 在 1988 年发明的一种新型棋类游戏[1][2]。虽然它的规则很简单，但它的打法却非常难[3]。这主要是因为有很多情况下[4]可以在游戏的每一步[5]中选择，通常在 500[6]以上，而每一局的步数可以达到 80[7]以上。正因如此，亚马逊国际象棋成为了测试搜索策略算法[9]的很好的实验样本[8]。自发明[10]以来，就吸引了众多计算机科学家[11]和计算机专业学生[12]的关注。

亚马逊象棋的主要难度在于它的复杂性:这主要是因为每一步棋都有很多情况可以选择，通常超过 500 步，步数可以超过 80 步。以每步 500 种情形为例，经过 4 步之后的情形总数将达到 600 多亿种，这无疑对传统的 minimax 搜索算法是一个很大的挑战。此外，一些真正的小把戏只有在 10 步甚至 20 步之后才能看到，前几步看更像是坏把戏。对于传统的极大极小算法来说，选择这样一个巧妙的小把戏是极其困难的。

蒙特卡洛(MC)方法[13]是一种通过反复随机抽样获得目标数值结果的计算方法。蒙特卡罗树搜索(MCTS)算法通过结合树搜索(TS)算法和适当的态势评估函数，在态势分支比较大的棋类游戏[14-16](如围棋和亚马逊国际象棋的对局)和实时策略游戏(如星际争霸[17]和 Dota2)的算法中取得了非常好的效果。例如，2016 年英国人工智能公司 DeepMind 的 AlphaGo 程序[18]以 4:1 的成绩击败韩国世界围棋冠军李世余，2018 年美国人工智能公司 OpenAI 的 OpenAI Five 程序[19]以压倒性的优势击败 Dota2 世界冠军，都是基于蒙特卡洛树搜索算法。

蒙特卡洛树搜索算法的效果，很大程度上取决于它能搜索的情境数量。能搜索的情况越多，算法就越容易找到平时很难找到的、能战胜对手的“招数”。也就是说，计算能力越强，蒙特卡洛算法的效果就越好。由于计算架构不同，计算机图形处理器(GPU)比中央处理器(CPU)具有更高的计算效率。利用 GPU 加速蒙特卡罗树搜索算法，可以显著提高算法的实际应用效果[20,21]。因此，本文选择使用 GPU 对亚马逊象棋蒙特卡洛树搜索算法中非常重要的评价函数进行加速，从而为提高算法的实际应用效果提供依据。

统一计算设备架构(CUDA)是由美国图形处理器(GPU)制造商 NVIDIA 推出的一种并行计算平台

计算机科学学院(5112210832)，北京信息科技大学大学生创新创业训练项目。

978-1-6654-6287-7/22/$31.00©2022 ieee doi
10.1109/ aiipcc57291.2022.00098

2007 年。它允许用户使用 NVIDIA 的 GPU 加速应用程序。因为 CUDA 计算平台的编程语言 CUDA-C 与标准的 C 语言非常相似，所以 CPU 代码通常只需要少量修改，就可以实现 CUDA-C 的 GPU 加速程序。因此，CUDA 计算平台在高性能计算领域得到了广泛的应用。早期版本的 AlphaGo 程序和 OpenAI Five程序也使用 CUDA 来加速程序。

为了提高蒙特卡洛树搜索算法对亚马逊象棋的运算性能，本文采用 CUDA-C 语言实现了态势评估函数的 GPU 加速程序，并通过实验分析了计算结果的准确性和不同参数下的计算性能，验证了 GPU 可以很好地应用于评估函数的加速计算。

2 理论

A.亚马逊象棋规则

亚马逊国际象棋通常是在 10 x 10 的棋盘上，由两个人轮流走，每边有 4 个棋子。棋子的移动规则与象棋中的皇后相同，即可以以棋子为中心在 8 个方向上自由移动，但不能跨越障碍。棋子的初始位置如图 1 所示。亚马逊象棋的每一步由两步组成。首先，选择自己的棋子在棋盘上移动，然后在以移动的棋子为中心的八个方向上选择一个位置来放置箭头。箭头类似于围棋中的棋子，棋子被移动后不能立即再次移动(图 2)，当对手的四枚棋子都不能移动时，对手就被击败了。如果双方都不能移动，则为平局。

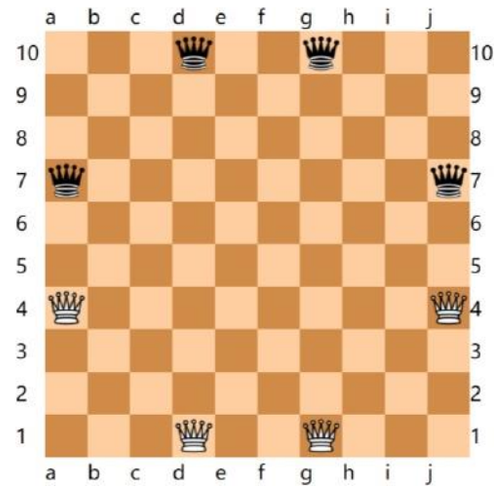


图1所示。亚马逊象棋的棋盘和棋子的初始位置

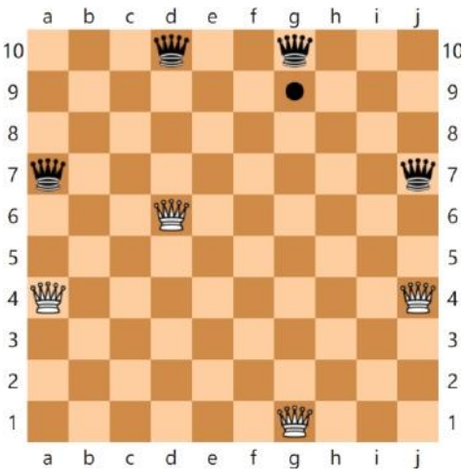


图2。亚马逊棋的棋盘和棋子。白棋从位置 d1 移动到位置 d6，位置 d9 放置箭头

B.蒙特卡洛树搜索算法

由于亚马逊棋的每一步都有很大的分支比，如果我们需要考虑当前情况下经过很多步后的情况，搜索空间会急剧增加。因此，传统的 minimax 搜索算法[12、22、23]只能计算有限步数。这样一来，就很难找到一些需要多步后再考虑的“小把戏”。相比之下，蒙特卡洛树搜索算法可以搜索到更多的步骤。

蒙特卡洛树搜索算法[24-26]包括“选择”、“扩展”、“模拟”和“反向传播”四个阶段(图 3)。在选择阶段，算法从当前构建的搜索树的根节点开始，自顶向下选择其中一个子节点;在“扩展”阶段，在选定的节点上添加新的子节点;在模拟阶段，计算新节点;最后，在反向传播阶段，将计算值更新为从底部到根节点的路径。

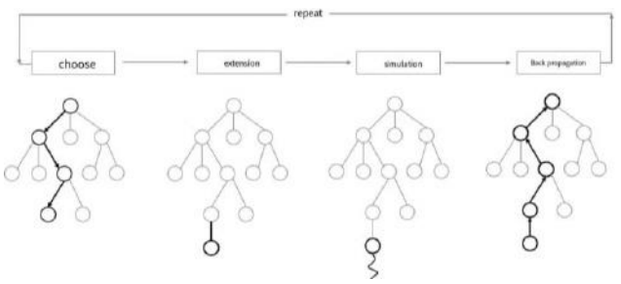


图3。MCTS 算法的基本步骤

然而，单纯的蒙特卡罗树搜索算法(选择当前节点得分最高的子节点)并不理想，部分原因是这种贪婪策略可能会错过非最优子节点之后的潜在移动。为了解决这个问题，本文采用了“应用于树的置信上界”算法[27]来指导节点的选择。如公式(1)所示，UCT 公式由两项组成，其中第一项 x表示

子节点的成功率，代表了算法对构建的搜索树信息的利用情况

(利用)，第二项 $\sqrt{\frac{\ln N}{n_i}}$ 表示

对未访问节点的算法探索(Exploration)。常数 C 用于调整算法对两种策略的比例。注意，根公式的值是无限的，因为有不可达的子节点，这保证了当前节点下不可达的子节点至少会被访问一次。

$$UCT = X_i + c\sqrt{\frac{\ln N}{n_i}} \quad (1)$$

其中 x 是子节点的胜率

N 为父节点被访问*的*次数 nis_i 子节点被访问的次数 c 为*常数*

C.评价函数

和很多桌游类似，亚马逊国际象棋大致可以分为三个阶段:开局、中期、结束[28]。每个阶段的策略也略有不同。因此，态势评估函数也应该包含不同阶段的策略。亚马逊棋的目标之一就是比对手拥有更多的开放空间。当棋盘上的一个空间只能被一方到达时，这个空间就可以被认为是该方的领土。

根据这些想法，J. Lieberum 提出了以下亚马逊国际象棋[28]的评估函数:

$D_1^j(a)$被定义为棋手 j(j = 1,2)通过国际象棋中的女王步法*到达开放空间 a 的*最小步数，$D_2^j(a)$*是棋手通过国际象棋中的国王步法到达开放空间 a 的最小步数*。国王步法是在 8 个方格内移动，在其 3×3 范围内移去中心点。图 4、图 5分别表示和在一定情况下。其中，开放空间左上角为 j = 1，右下角为 j = 2。

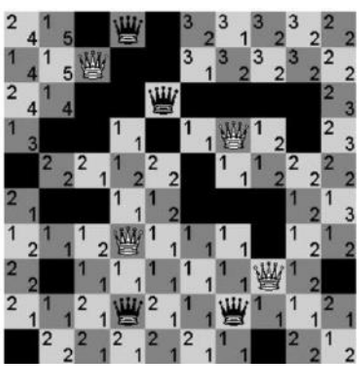


图4。女王到达指定空间所需的最少步数

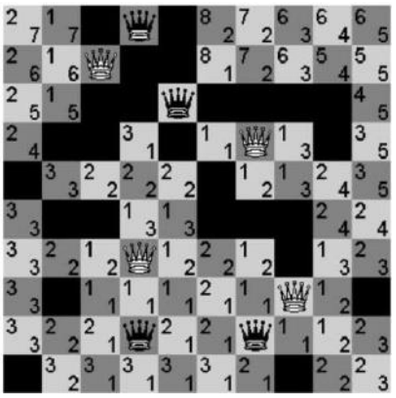


图5。国王到达指定空间所需的最少步数

当开放空间 a 上有 D1 1 $D12_{(a)}$*时，一般意味着参与人 1 比参与人 2 更容易到达 a*。但在开局阶段，由于箭头数量较少，棋盘上的空地可能值很小，但实际上无法一步到位到达这些空地。*相比之下，D(a)具有更好 $_2{}^j$的*局部*特征*和*更稳定的数值*。

定义代表疆域的评价变量 t_j，如公式(2)所示，当参与人 1更容易到达开放空间 a时，t_j为 1，反之为-1，其中 k为常数，用于表示 $Di_j(a)$相同时第一个掉落方块的优势。一般取|k|≤1/5，本文取 k=0.1。

$$t_i = \sum_a \Delta(D_i(a) - D_i(a)) \quad (2)$$

在哪里

$$\Delta(n,m) = \begin{cases} 0 & if\ n = m = \infty \\ k & if\ n = m < \infty \\ 1 & if\ n < m \\ -1 & if\ n > m \end{cases}$$

但是，$(Di_1(a), Di_2(a))$没有考虑到的值代表了玩家之间的优势差异，因此定义的评价变量 c 和 $care_1$，$_2$如公式(3)和(4)所示。

$$c_1 = \sum_a (D(a) - D(a)) \quad (3)$$

$$c_2 = \min(1, \max(-1, (D_{22}(a) - D1_{(2a)})/6)) \quad (4)$$

一个

然后，如式(5)所示，定义权重变量 w。权值变量的值通常会随着棋局的进展而减小。

$$w = \sum_{a} \square^{-|D\square a \square - D\square a \square|} \quad (5)$$

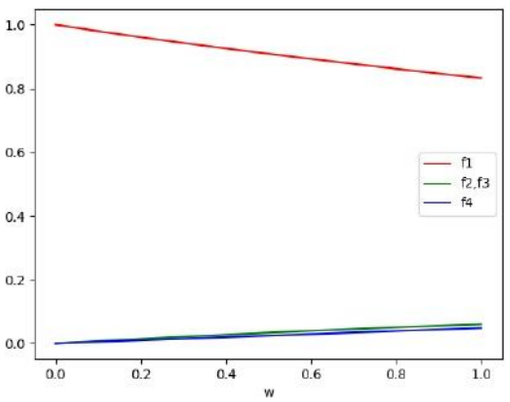
最后如式(6)所示，将 t_1、t_2、c_1、c_2 into 组合为一个整体评价函数:

$$T = f_1(w)T_1 + f_2(w)c_2 + f_{13}(w)c + f(w)T_2 \quad (6)$$

式中，f_i is 为各变量间权重的除法函数，即 $f_i = 1$。本文使用的配分函数如式(7)所示:

$$\begin{cases} f_{\square} = \dfrac{\square}{w + \square} \\ f_{\square} = f_{\square} = \dfrac{\square}{\square} \square - \dfrac{\square}{w + \square} - \dfrac{w}{\square w + \square\square\square} \square \\ f_{\square} = \dfrac{w}{\square w + \square\square\square} \end{cases} \quad (7)$$

(6)式中配分函数与权值变量 w 的关系如图 6 所示。*由于 w 的值会随着棋局的进行而减小，所以 f 的值* $_i$ 会在图上从 x 轴的右侧移动到左侧。即 f逐渐 $_1$ 减小，f_2，f_3，f_4 逐渐增大。



图。6。配分函数随权值变量 w 的变化

D.评价函数的 GPU 加速

相比于 CPU 一般有几个到十几个计算核心，带有 CUDA 功能的 GPU 通常有上千个计算核心[29]。因此，对于计算密集型任务，GPU 的计算速度大大超过了 CPU[30-34](图 7)，然而，虽然 CUDA GPU 可以同时执行很多指令，但单个 CUDA 核的计算速度要比传统 CPU 核慢得多。这是因为，与传统 CPU 不同的是，CUDA 采用了一种叫做单指令多线程(SIMT)的系统架构。一组被称为 Warp 的线程执行同一段代码。如果一个线程中的代码执行在一个 warp 中发生分支，一些线程会被挂起来执行分支代码。因此，如果 CUDA 代码中有很多控制分支，效率会大大降低[35]。

CUDA 程序由主机端(CPU)代码和设备端(GPU)代码两部分组成。主机端代码主要负责输入输出，以及整个程序的调度。

在 CUDA 的编程模型中(见图 8)，在主机端调用的设备端函数称为内核函数。内核函数在 GPU 中由许多线程组成的线程块中执行。不同的线程块不会相互干扰。线程块中的线程可以通过共享内存高速交换数据[36,37]。一个内核函数的所有线程块的集合称为网格。内核函数在网格中共享相同的 GPU 全局内存。

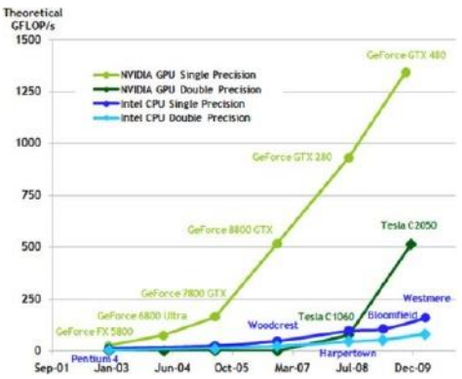


图7。CPU 和 GPU 计算能力对比

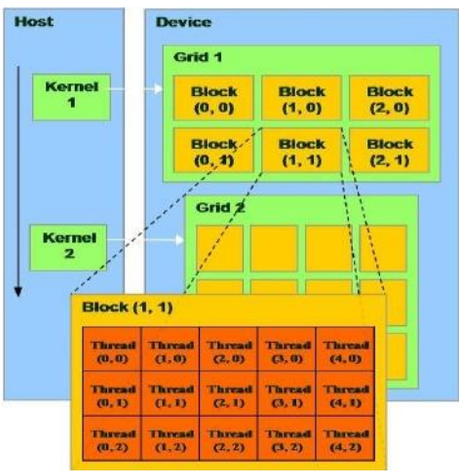


图8。CUDA 的基本系统架构

3 实验

实验分为两部分，分别评估函数的计算精度和核函数的调用参数对 GPU 加速性能的影响。代码在笔记本电脑上运行。CPU 为 Intel Core2 i7-8850h, 6 核 CPU，内存为 32GB。运行在 windows 10 操作系统上。GPU 加速卡是 NVIDIA Quadro p3200，有 1792 个 CUDA 内核，14 个多处理器运行在 1.3GHz。显存为 6GB。软件编程开发环境使用 Microsoft Visual Studio 2015和 CUDA计算平台软件包 10.1(图 9)。

为了比较分析评估函数的 GPU 加速效果，编写程序随机生成要计算的情况并保存在文件 moves.txt 中(图 10)。为简单起见，本文从头开始，在本文中，从一开始，棋子是随机选取的，合法的走法也是随机选取的。走法的坐标被保存在文件中。然后分别编写 CPU和 CUDA程序，依次计算所有情况的值，并保存在输出文件 scores_ CPU .txt 和 scores_ Gpu.txt 中，以便后续分析(图 11)。

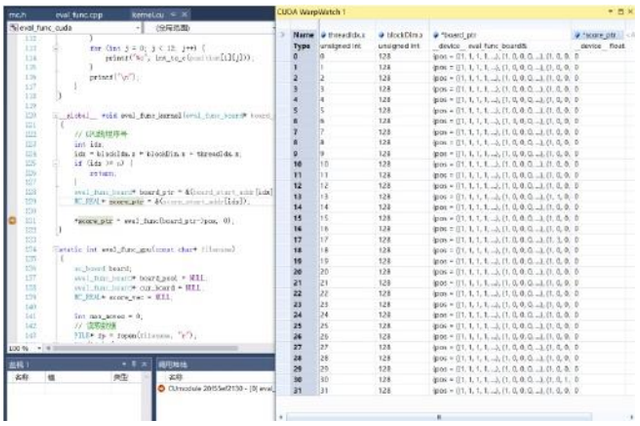


图9。用 visual studio 2015 调试 CUDA 程序

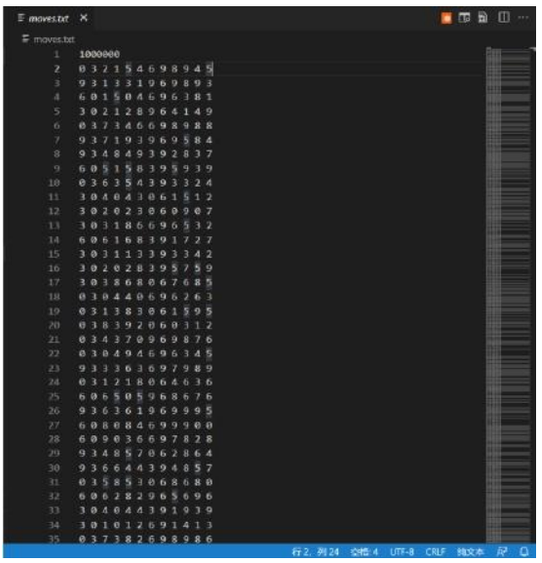


图10。用于评估函数计算的输入文件

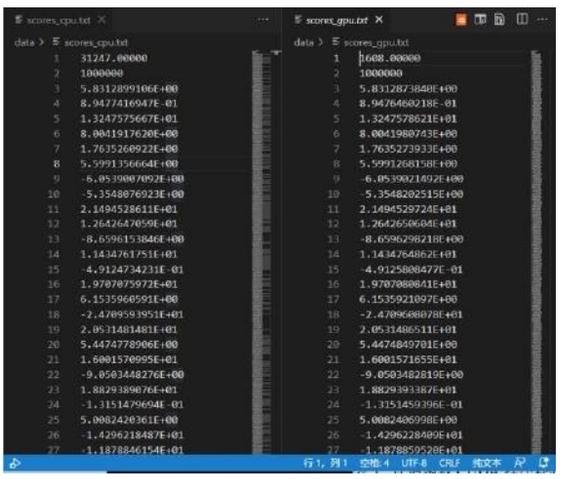


图11。CPU 和 GPU 计算结果的误差

四、结果与讨论

A.*评价函数计算的准确性*

通用 CPU 使用双精度浮点数进行计算，而 GPU 一般使用单精度浮点数，速度比双精度浮点数快。以 NVIDIA Quadro P3200 为例，其理论单精度浮点运算速度约为 5.5 Tflops，即每秒可完成 5.5 万亿次浮点运算，而双精度浮点运算速度仅为 0.17 Tflops，约为单精度浮点运算速度的 1 / 32。因此，GPU 加速的运算通常都是用单精度浮点数来执行的。为了验证 GPU 运算的结果，我们使用了 CPU 版本程序和 CUDA 版本程序来计算随机生成的 sub 的评估函数

棋局，并比较了二者的误差。如图 12 所示，大多数结果的误差都在和之内，说明 CUDA 程序可以准确计算评价函数。

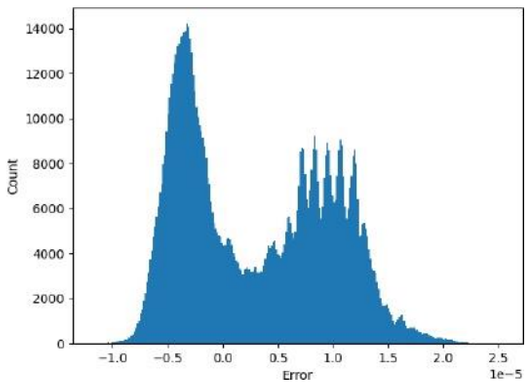


图12。CPU 和 GPU 计算结果的误差

B.核函数的调用参数对性能的影响

由于 GPU 架构的原因，单个 GPU 核的运算速度要比 CPU 慢很多。为了达到加速效果，很多线程需要同时在 GPU 上运行。表 1 和表 2 显示了在不同 CUDA 调用内核函数的网格参数下计算评估函数次数的时间消耗，其中为网格中的线程块数量，为单个线程块中包含的线程数量。

从表 1 中可以看出，随着同时运行的线程数的增加，计算时间消耗逐渐减少，在时达到最小值，但在此后时间消耗略有增加。这主要是因为实验中使用的 GPU 加速器有 1792 个 CUDA 核。线程数较少时，GPU 没有被充分利用，因此运算速度变慢。当线程数达到一定数量时，GPU 的所有内核都被充分利用，运算速度达到最低。我们注意到，网格中最高效率的线程数为 16384 个，远远高于 GPU 物理核的数量。这表明，调用内核函数需要比物理核数量更多的线程，才能达到 GPU 最高的工作效率。继续增加线程数无法继续减少计算时间。可能是因为过多的线程会增加系统在分配线程资源上花费的时间，从而增加总的计算时间。

表1。cuda 程序的评估函数时间随核函数调用参数的变化(在网格逐渐增加)

注	N_t	计算时间(秒)
16	16	14.489
32	32	5.056
64	64	2.009
128	128	1.603
256	256	1.835
512	512	1.804
1024	1024	1.756

表2。cuda 程序的评估函数时间随核函数调用参数的变化(在网格固定为 4096)

注	N_t	计算时间(秒)
4	1024	2.237
16	256	2.058
64	64	2.009
256	16	2.004
1024	4	4.238
4096	1	10.534

表 2 显示了在不同和条件下，总线程数不变的情况下，计算评估函数所花费的时间。从表中可以看出，计算时间在逐渐增加时逐渐减少，达到 1024 时略有增加。这主要是因为，与 CPU 内核不同，单个 CUDA 内核无法独立执行计算，而是以 32 warp 的形式运行相同的代码。因此，是小而整体运算速度会严重变慢。表中注意，达到 16 后，计算时间不再减少。这可能主要是因为评估函数的计算比较复杂，代码有很多控制分支，所以 warp 中的很多线程在运行过程中都处于等待状态。

作为对比，CPU 代码对评估函数进行多次计算需要 31.247 s，远慢于 GPU 加速计算速度 1.603 s，说明我们编写的 GPU 加速程序取得了很好的加速效果，GPU: CPU 加速比达到了 20.0。

五、结语及下一步

本文主要介绍了亚马逊棋蒙特卡洛树搜索算法的原理，并基于 CUDA 计算平台实现了亚马逊棋态势评估函数的 GPU 程序。最后，通过实验验证了评估函数的 GPU 加速的可行性和有效性。

从本文的实验结果可以看出，GPU 加速对于亚马逊国际象棋的评估函数计算具有非常明显的性能优势。下一步就是将 GPU 加速应用到整体的蒙特卡洛树搜索算法中。

参考文献

[1]郭 q，舒琴 L I。亚马逊的评价函数计算机博弈研究[J]。计算机工程与应用，2012,48(34):50- 54+87。

李锐，高明。基于 CNN 模型的亚马逊棋类搜索算法设计[J] .数字技术与应用，2022,40(02):164-166。

郭涛，邱浩，童波，等。亚马逊国际象棋中多种博弈算法的优化与比较[C]。2019 中国控制与决策会议(CCDC)， 2019: 6299-6304。

[4]建宁 Q，洪坤 Q, W 亚杰等。UCT 技术在亚马逊电脑游戏中的应用[C]。2016 中国控制与决策会议(CCDC)， 2016: 6896-6899。

[5]鞠杰，邱浩，王峰，等。亚马逊国际象棋中基于并行 PVS 算法的线程优化与开放库研究[C]。2021 年第 33 届中国控制与决策会议(CCDC)，2021: 2207-2212。

李泽，宁超，曹军，等。基于 UCT-PVS 混合算法的亚马逊象棋[C]。2021 年第 33 届中国控制与决策会议(CCDC)， 2021: 2179-2183。

[7]孟 D, Jianbo B, yizzhong Q，等。基于强化学习的亚马逊棋类对弈系统设计[C]。2019 中国控制与决策会议(CCDC)， 2019: 6337-6342。

童斌，邱慧，郭涛，等。基于亚马逊人机博弈的 PVS 算法并行计算研究与应用[C]。2019 中国控制与决策会议(CCDC)， 2019: 6293-6298。

陈曦，杨莉。亚马逊国际象棋中的评价函数研究[J] .计算机知识与技术，2019,15 (08):224-226

[10]李泽，李 Y，冉 g，等。基于 PVS 搜索算法的亚马逊棋类对弈系统设计[J] .智能计算机与应用，2018,8(05):86-88。

王聪，丁明。个性化亚马逊象棋界面设计与实现[J] .智能计算机与应用，2017,7(02):78- 80。

张磊。基于 minimax 搜索算法的亚马逊象棋对弈系统研究[D]东北大学，2010。

Metropolis N, Ulam S.蒙特卡罗方法[J]。美国统计学会学报，1949,44(247):335-341。

[14] Coulom R.蒙特卡洛树搜索中的高效选择和备份算子[C]。计算机与游戏，2007:72-83。

[15] Gelly S, Kocsis L, Schoenauer M，等。计算机围棋的重大挑战:蒙特卡洛树搜索与扩展[J]。Commun。Acm, 2012, 55(3): 106-113。

李志强，李志强。基于 Monte-Carlo 树搜索的计算机围棋快速动作值估计[J].中文信息学报，2016,35(6):1102 - 1102。人工智能，2011,175(11):1856-1875。

[17] Vinyals O, Babuschkin I, Czarnecki W M，等。基于多智能体强化学习的星际争霸 II 中的 Grandmaster 级别[J]。自然，2019,575(7782):350-354。

[18] Silver D，黄 A, Maddison C J，等。利用深度神经网络和树搜索来掌握围棋棋局[J]。自然，2016,529(7587):484- 489。

[19] Berner C, Brockman G, Chan B，等。基于大规模深度强化学习的 Dota 2 [J]。arXiv，2019 年。

[20] 陈建新。计算的进化:AlphaGo[J]。科学与工程中的计算，2016,18(4):4-7。

李志强，李志强。基于 GPU 的大规模并行蒙特卡洛树搜索算法[j].计算机工程，2016,35(6):1182 - 1182。2011 IEEE 并行与分布式处理研讨会暨博士论坛国际研讨会，2011:2034-2037。

[22] Shannon C E. XXII。计算机下国际象棋的程序设计[J]。伦敦、爱丁堡和都柏林哲学杂志和科学杂志，1950,41(314):256-275。

2004 年[23]。冯·N，摩根斯顿，王伟等。博弈论与经济行为[M]北京:生活、阅读与新知三联书店，2004。

[24] C, B, w; E, w;基于蒙特卡洛树搜索算法的研究进展[J]。计算智能与游戏 AI, 2012,4(1): 1-43。

[25] Chaslot G, Bakkes S, Szita I，等。蒙特卡洛树搜索:一种新的游戏 AI 框架[J]。人工智能与交互式数字娱乐 AAAI 会议论文集，2021,4(1):216-217。

[26] Ś wiechowski M, Godlewski K, Sawicki B，等。蒙特卡洛树搜索:近期改进与应用综述[J]。人工智能综述，2022。

[27] Kocsis L, Szepesvári C.基于 Bandit 的蒙特卡洛规划[C]。机器学习:ECML 2006, 2006: 282-293。

李志强，李志强。一种改进的亚马逊游戏评价函数[J]。理论计算机科学，2005,349(2):230-244。

刘志强，刘志强。基于 CUDA 的并行计算[j].计算机工程，2016,35(6):1172 - 1172。2010 IEEE 国际并行与分布式处理研讨会(IPDPS)， 2010。

张建军，张志强，张志强，等。基于 gpu 的医学图像重建技术综述[J].计算机工程与应用，2010。医学物理，2017,42:76-92。

邢莉。GPU 计算在医学物理中的应用综述[J].物理学报，2016,35(6):1102 - 1102。医学物理，2011,38(5):2685-2697。

张志强，李志强，李志强，等。基于 GPU 的 GPU 功耗分析方法研究[J].计算机工程学报，2016,35(6):1172 - 1172。ACM 第一版。测量员，2016, 49(3):第 41 条。

贾欣，Ziegenhein P，蒋绍波。基于 gpu 的放射治疗高性能计算[J].中国放射医学杂志，2016,35(6):1102 - 1102。医学与生物物理，2014,59(4):R151-R182。

张志强，张志强，张志强，等。GPU 能效分析与优化方法研究[J].计算机工程，2015,35(6):1102 - 1102。ACM 第一版。测量员， 2014, 47(2):第十九条。

韩涛 D, Abdelrahman T S.减少 GPU 程序分支分歧[j].计算机应用与开发，2016,35(6):1172 - 1172。第四届图形处理器通用处理研讨会论文集，2011:第 3 条。

Kim Y, Shrivastava A A. CuMAPz:一种分析 CUDA 访存模式的工具[C]。2011 第 48 届 ACM/EDAC/IEEE 设计自动化会议(DAC)， 2011: 128-133。

刘建军，王晓东，王晓东。基于 CUDA 架构的多核处理器设计与实现[J].计算机工程与应用，2016,35(6):1102 - 1102。计算机图形学论坛，2013,32(3pt2):161-170。