DeepL
订阅DeepL Pro以翻译大型文件。
欲了解更多信息，请访问www.DeepL.com/pro。

2022年人工智能、信息处理和云计算国际会议(AIIPCC)

亚马逊棋牌的蒙特卡洛树搜索算法的GPU加速及其评估功能

伊凯孙
北京信息科技大学计算机学院，北京信息科技大学
北京，中国
sunyikai2021@outlook.com

高明
北京信息科技大学计算机学院，北京信息科技大学
北京，中国 gaomingshsf@hotmail.com

东元
北京信息科技大学计算机学院，北京信息科技大学
北京，中国
YUANDONG20211@outlook.com

朱鹏辉
北京信息科技大学计算机学院，北京信息科技大学
北京，中国
941509604@qq.com

***摘要*--评价函数是亚马逊国际象棋的蒙特卡罗树搜索（MCTS）算法中非常重要的部分。本文介绍了蒙特卡罗树搜索算法的原理，并基于CUDA计算平台，实现了亚马逊棋局评价函数的GPU加速计算。利用CPU代码和CUDA代码，进行了两个实验，评估了函数计算精度和内核函数调用参数对GPU加速性能的影响。结果表明，该评价函数对GPU加速是可行的、有效的，GPU对CPU的加速比达到20.0。**

关键词-组件；亚马逊国际象棋；人工智能；蒙特卡洛搜索树；并行操作

I. 简介

亚马逊象棋是阿根廷人Walter Zamkauskas在1988年发明的一种新的棋盘游戏[1][2]。虽然它的规则非常简单，但其玩法却非常困难[3]。这主要是因为在游戏的每一步中可以选择很多情况[4]，通常超过500个[6]，而每一局的步骤数可以达到80多个[7]。由于这个原因，亚马逊国际象棋已经成为测试搜索策略算法[9]的良好实验样本[8]。自发明以来[10]，它吸引了许多计算机科学家[11]和计算机学生[12]的注意。

亚马逊国际象棋的主要困难在于其复杂性：这主要是因为每一步棋可以选择的局面很多，通常超过500种，而步骤数可以超过80个。以每步500种情况为例，四步之后的情况总数将达到600多亿，这对传统的最小搜索算法无疑是一个巨大的挑战。此外，一些真正的技巧只有在10步甚至20步之后才能看到，而前几步看起来更像是坏的诡计。对于传统的最小化算法来说，要选择这样一个巧妙的技巧是非常困难的。

蒙特卡洛（MC）方法[13]是一种通过重复随机抽样获得目标数值结果的计算方法。通过结合树搜索（TS）算法和适当的形势评价函数，蒙特卡洛树搜索（MCTS）算法在形势分支比大的棋类游戏[14-16]（如围棋和亚马逊象棋）和实时战略游戏（如《星际争霸》[17]和Dota2）的算法中取得了非常好的结果。例如，2016年，英国人工智能公司DeepMind的AlphaGo程序[18]以4:1的比分击败了韩国围棋世界冠军李世石，2018年，美国人工智能公司openai的OpenAI Five程序[19]以压倒性优势击败了Dota2世界冠军，这些都是基于蒙特卡洛树搜索算法的。

蒙特卡洛树搜索算法的效果主要取决于它能搜索的情况数量。它能搜索的情况数量越多，算法就越容易找到平时难以发现的 "技巧"，并能战胜对手。换句话说，计算能力越强，蒙特卡洛算法的效果就越好。由于计算架构的不同，计算机图形处理单元（GPU）的计算效率要高于中央处理器（CPU）。使用GPU加速Monte Carlo树形搜索算法可以显著提高该算法的实际应用效果[20,21]。因此，本文选择用GPU来加速亚马逊棋牌的蒙特卡洛树形搜索算法中非常重要的评估函数，从而为提高算法的实际应用效果提供依据。

计算统一设备架构（CUDA）是由美国图形处理单元（GPU）制造商NVIDIA推出的一个并行计算平台。

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

计算机学院（5112210832）、北京信息科技大学大学生创新创业培训项目。

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

2022年人工智能、信息处理和云计算国际会议（AIIPCC）| 978-1-6654-6287-7/22/$31.00 ©2022年IEEE | DOI: 10.1109/AIIPCC57291.2022.00098

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

美国，2007年。它允许用户使用NVIDIA的GPU来加速应用程序。由于CUDA计算平台的编程语言CUDA-C与标准的C语言非常相似，通常只需要对CPU代码进行少量修改，就可以实现CUDA-C的GPU加速程序。正因为如此，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程序也使用了CUDA来加速程序。

为了提高亚马逊棋牌的蒙特卡罗树搜索算法的运行性能，本文采用CUDA-C语言实现了形势评估函数的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。

II. 理论

A. 亚马逊国际象棋的规则

亚马逊国际象棋通常由两个人轮流在10×10的棋盘上移动，每边有四个棋子。棋子的移动规则与国际象棋中皇后的移动规则相同，即可以在以棋子为中心的八个方向自由移动，但不能跨越障碍物。棋子的初始位置如图1所示。亚马逊国际象棋的每一步都包括两个步骤。首先，选择自己的一个棋子在棋盘上移动，然后在以被移动的棋子为中心的八个方向上选择一个位置，放置一个箭头。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）。当对方的四个棋子都不能移动时，对方就被打败了。如果双方都不能移动，则为平局。

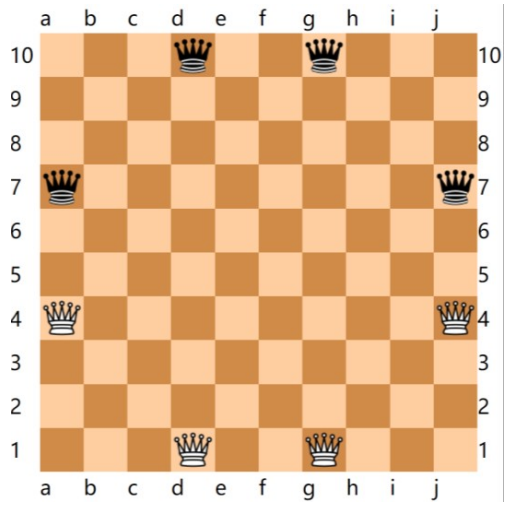


图1.亚马逊国际象棋的棋盘和棋子的初始位置

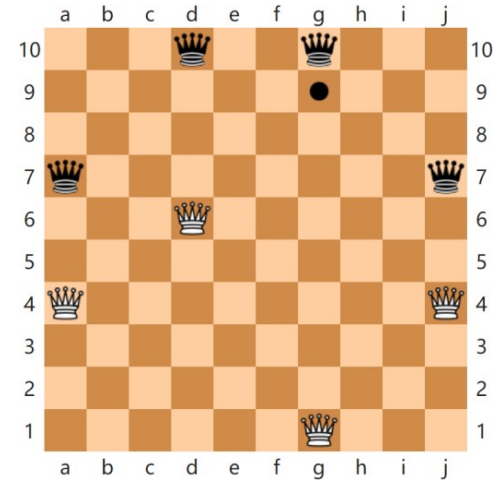


图2. 亚马逊国际象棋的棋盘和棋子。白棋从d1位置走到d6位置，一个箭头放在d9位置上

B. 蒙特卡洛树搜索算法

由于亚马逊象棋的每一步都有很大的分支比例，如果我们需要考虑当前情况的许多步之后的情况，搜索空间将急剧增加。因此，传统的最小搜索算法[12, 22, 23]可以计算的步数有限。这样一来，就很难找到一些需要在很多步之后考虑的 "小技巧"。相比之下，蒙特卡洛树搜索算法可以搜索更多的步骤。

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，从上到下选择一个子节点；在扩展阶段，在选择的节点上增加一个新的子节点；在模拟阶段，计算新的节点；最后，在反向传播阶段，计算值被更新到从底部到根节点的路径。

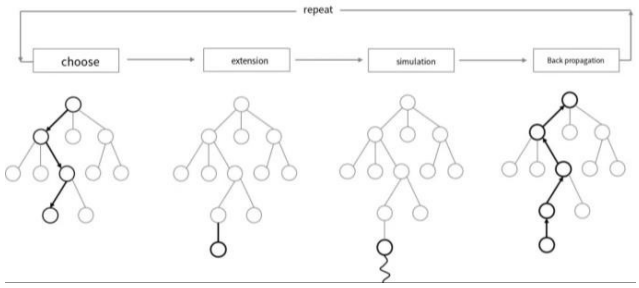


图3.MCTS算法的基本步骤

然而，纯粹的蒙特卡洛树搜索算法（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的策略可能会错过非最优子节点之后的潜在动作。为了解决这个问题，本文采用了 "上层信心
应用于树的界限 "算法[27]来指导选择
的节点。如公式（1）所示，UCT公式为
由两个项组成，其中第一个项x_i表示

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

子节点的成功率，表示算法对所构建的搜索树信息的利用率

(开发)，而第二项 $\sqrt{\frac{\ln N}{a}}$ 是指
在未访问的节点上进行算法的探索（探索）。常数C被用来调整算法在两种策略中的比例。请注意，由于无法到达的子节点，根公式的值是无限的，这保证了当前节点下无法到达的子节点至少会被访问一次。

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

其中 $\bar{X}_i$ 是子节点的获胜率

N是访问父节点的次数 n_i是访问子节点的次数 c

是一个常数

C. 评价功能

与许多棋类游戏类似，亚马逊象棋可以大致分为三个阶段：开局、中局和结局[28]。每个阶段的策略也略有不同。因此，形势评估函数也应该包括不同阶段的策略。亚马逊象棋的目标之一是比对手拥有更多的开放空间。当棋盘上的某个空间只能由一方到达时，该空间可被视为该方的领土。

根据这些想法，J. Lieberum提出了以下亚马逊国际象棋的评价功能[28]：

$D^j(a)$被定义为一个最小的步骤数。
运动员 j $(j=1, 2)$ 到达开放空间a，通过
国际象棋中的皇后走法，而$D^j(a)$是最小数量的
棋手通过国际象棋中的国王走法到达空位a的步骤。国王走法是在其3×3的范围内，在中心被移走的八个格子内移动。图4和图5分别显示了和在某种情况下。其中，开放空间的左上角是j=1，右下角是j=2。

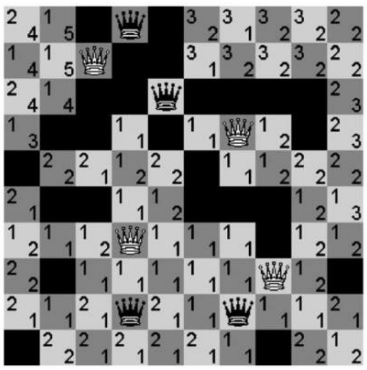


图4.女王到达指定空间的最小步数

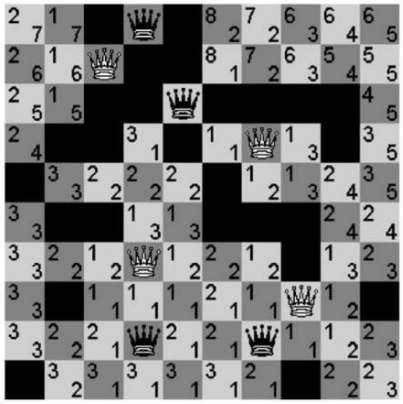


图5.国王到达指定空间的最小步数

当开放空间a上有$D^1(a) < D^2(a)$时，一般来说，这意味着棋手1比棋手2更容易到达a。然而，在开局阶段，由于少量的
箭头，棋盘上的空地可能有非常小的价值，但事实上他们无法达到这些空地上的
一个步骤。相比之下，$D^j(a)$ 具有更好的局部特征和更稳定的数值。

定义代表领土的评价变量t_j，如公式（2）所示。当棋手1更容易到达空地a时，t_j为1，反之为-1，其中k是一个常数，用来表示第一落子方的优势
当$D^j(a)$是相同的。一般来说，|k|≤1/5是取的，而
$k=0.1$在本文中被采纳。

$$t = \sum_a \Delta(D_i^1(a), D_i^2(a)) \quad (2)$$

其中

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

然而、 $\Delta\left(D_i^1(a), D_i^2(a)\right)$ 不考虑
的值代表玩家之间的优势差异，因此定义了评价变量c_1和c_2，如公式（3）和（4）所示。

$$c = 2\sum_a (2 - D_i^1(a) - 2 - D_i^1(a)) \quad (3)$$

$$c = \sum_a (1, \max(-1, (D^2(a) - D^1(a)) / 6)) \quad (4)$$

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

然后，如公式（5）所示，定义了权重变量w。权重变量的值通常随着棋局的进展而减少。

$$w = \sum_{a} 2^{-|D_1(a) - D_2(a)|} \quad (5)$$

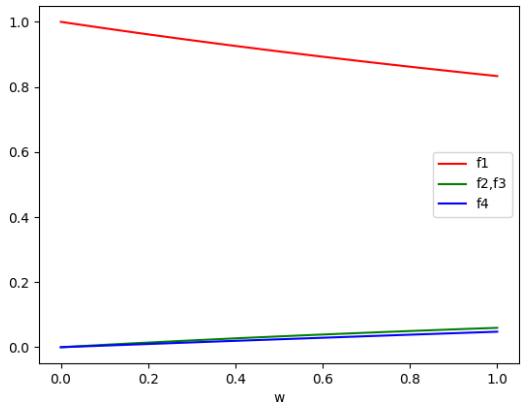
最后，如公式（6）所示，将$_{t1}$，$_{t2}$，$_{c1}$，$_{c2}$合并为一个整体评价函数：

$$t = {}_{f1}(w)_{t1} + {}_{f2}(w)_{c1} + {}_{f3}(w)_{c2} + {}_{f4}(w)_{t2} \quad (6)$$

其中，$_{fi}$是每个变量之间权重的划分函数，即$\sum_{fi}=1$.本文中使用的划分函数如公式（7）所示：

$$\begin{cases} f_1 = \dfrac{5}{w+5} & 1 \quad 5 \quad w \\ {}_{f2} = {}_{f3} = \dfrac{1}{2}\left(1 - \dfrac{W}{W+5} \cdot \dfrac{W}{(W+20)}\right) \\ {}_{f4} = \dfrac{w}{(w+20)} \end{cases} \quad (7)$$

公式（6）中分区函数和权重变量w之间的关系如图6所示。由于w的值随着棋局的进展而减少，所以价值的$_{fi}$ 将从X轴的右侧移动到图中的左侧。就是说、 f_1 逐渐减少，并且 $_{f2}$ ， $_{f3}$ ， $_{f4}$ 逐渐增加。



图。6.分区函数随权重变量w的变化情况

D. GPU对评估功能进行加速

与CPU一般有几个到十几个计算核心相比，具有CUDA功能的GPU通常有成千上万个计算核心[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，尽管CUDA GPU可以同时执行许多指令，但单个CUDA核心的计算速度要比传统CPU核心慢得多。这就是因为，与传统的CPU不同，CUDA采用了一种叫做单指令多线程（SIMT）的系统架构。一组称为Warp的线程执行同一块代码。如果在经线中的线程执行代码时出现分支，一些线程就会暂停执行分支代码。因此，如果CUDA代码中存在许多控制分支，效率将大大降低[35]。

一个CUDA程序包括两部分：主机侧（CPU）代码和设备侧（GPU）代码。主机代码主要负责输入和输出，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被称为内核函数。内核函数在一个由GPU中的许多线程组成的线程块中执行。不同的线程块之间不会相互干扰。一个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是称为网格。内核函数共享相同的GPU全局格中的记忆。

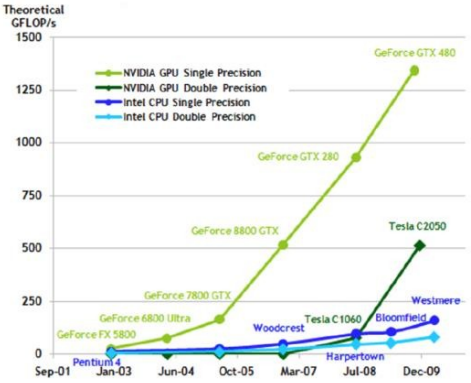


图7.CPU和GPU之间的计算能力比较

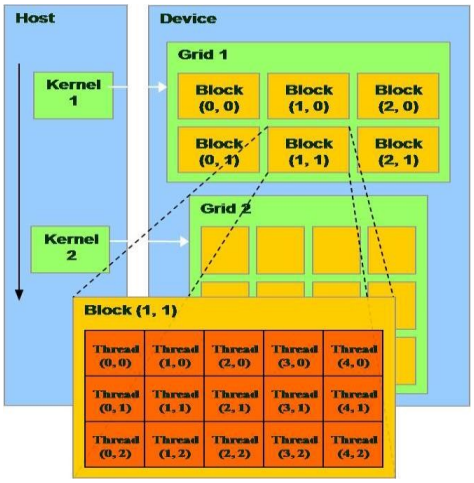


图8.CUDA的基本系统结构

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

III. 实验

实验分为两部分，评估函数的计算精度和内核函数的调用参数对GPU加速性能的影响。该代码在一台笔记本电脑上运行。CPU是英特尔Core2 i7-8850h，有六个CPU核心，内存是32GB。它运行在Windows 10操作系统上。GPU加速卡是NVIDIA Quadro p3200，配备有1792个CUDA核心和14个多处理器，运行频率为1.3GHz。视频内存为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）。

为了比较和分析评价函数的GPU加速效果，编写一个程序，随机生成要计算的情况，并保存在文件moves.txt中（图10）。为了简单起见，本文从头开始，在本文中，从一开始就随机选择棋子，合法的棋步也是随机选择的。棋子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数值，并保存在输出文件scores_ Cpu.txt和scores_ Gpu.txt中，以便后续分析（图11）。

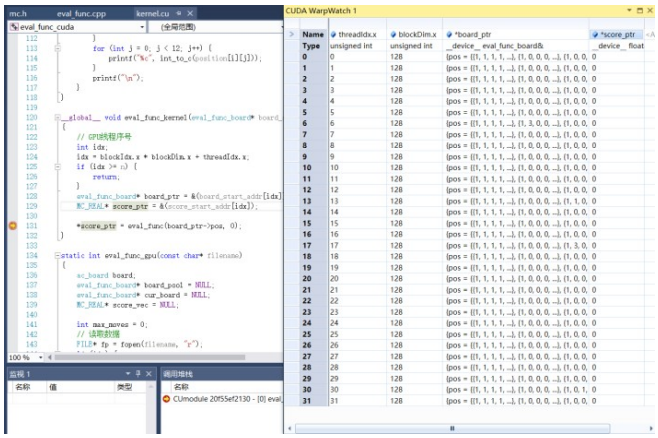


图9.用visual studio 2015调试CUDA程序

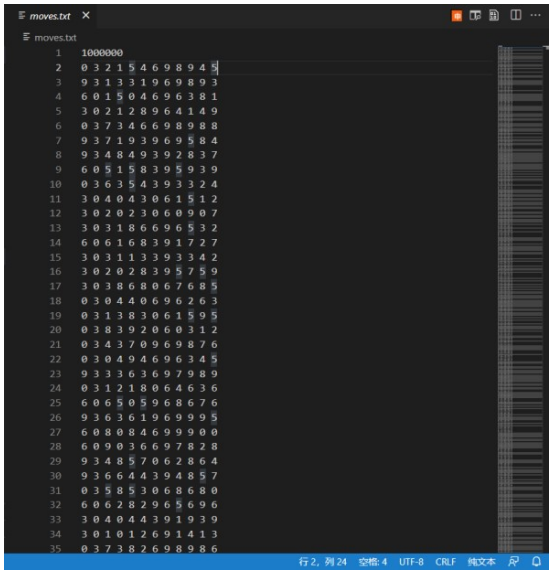


图10.评价函数计算的输入文件

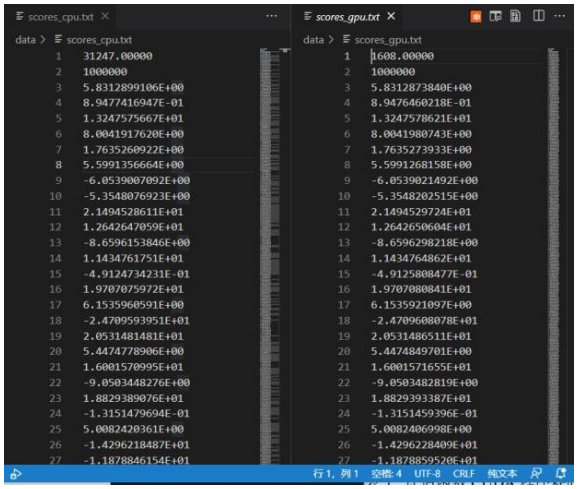


图11.CPU和GPU计算结果的误差

IV. 结果和讨论

A. 评价函数计算的准确性

一般CPU采用双精度浮点运算，而GPU一般采用单精度浮点运算，其运算速度比双精度浮点运算要快。以NVIDIA Quadro P3200为例，其理论上的单精度浮点运算速度约为5.5Tflops，即每秒可完成5.5万亿次浮点运算，而其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加速运算通常使用单精度浮点数进行。为了验证GPU操作的结果，我们用CPU版本的程序和CUDA版本的程序来计算次随机生成的评价函数

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

棋局，并比较了两者的误差。如图12所示，大多数结果的误差都在和之内，这说明CUDA程序能够准确地计算出评价函数。

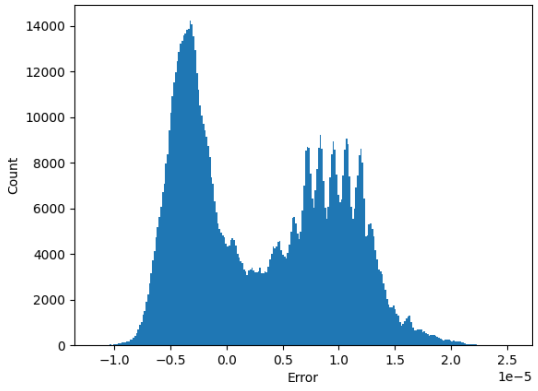


图12.CPU和GPU计算结果的误差

B. 核心函数的调用参数对业绩

由于GPU架构的原因，单个GPU核心的运行速度要比CPU慢得多。为了达到加速效果，许多线程需要同时在GPU中运行。表1和表2显示了不同CUDA调用内核函数的网格参数下计算评价函数的时间消耗，其中是网格中的线程块数，是单个线程块中包含的线程数。

从表1可以看出，随着同时运行线程数的增加，计算时间消耗逐渐减少，在达到最小，但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核心。当线程数较少时，GPU没有得到充分的利用，所以运行速度会减慢。当线程数达到一定数量时，GPU的所有内核都被完全利用，运算速度达到最低。我们注意到，在最高效率下，网格中的线程数为16384，远远高于GPU物理核心的数量。这表明需要比物理核心数量更多的线程来调用内核函数以达到GPU的最高工作效率。继续增加线程的数量不能继续减少计算时间。这可能是因为过多的线程会增加系统分配线程资源的时间，从而增加总的计算时间。

表1.cuda程序的评估函数时间随内核函数调用参数的变化（网格中的线程数逐渐增加）

Nb	*Nt*	**计算时间(s)**
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内核函数调用参数的变化（网格中的线程数固定为4096）。

Nb	*Nt*	**计算时间(s)**
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显示了在总线程数不变的情况下，不同和条件下计算评价函数的时间。从表中可以看出，当逐渐增加时，计算时间逐渐减少，而当达到1024时，计算时间略有增加。这主要是因为，与CPU内核不同，单个CUDA内核不能独立进行计算，而是以32个经线的形式运行相同的代码。因此，很小，整体运行速度会严重减慢。在表中注意到，在达到16之后，计算时间就不再减少了。这可能主要是因为评估函数的计算很复杂，而且代码中有很多控制分支，所以在运行过程中，经线中的很多线程都处于等待状态。

作为对比，CPU代码计算评价函数的次数需要31.247秒，远远慢于GPU加速计算速度1.603秒，说明我们编写的GPU加速程序取得了良好的加速效果，GPU：CPU的加速比已经达到了20.0。

V. 结论和下一步

本文主要介绍了亚马逊国际象棋的蒙特卡洛树形搜索算法的原理，并基于CUDA计算平台实现了亚马逊国际象棋形势评价功能的GPU程序。最后，通过实验验证了GPU加速评估功能的可行性和有效性。

从本文的实验结果可以看出，GPU加速对于亚马逊象棋的评价函数计算具有非常明显的性能优势。下一步是将GPU加速应用于整个蒙特卡洛树搜索算法。

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

参考文献

[1] Guo Q , Shuqin L I .亚马逊计算机游戏的评价功能研究[J].计算机工程与应用, 2012, 48(34)：50-54+87.

[2] Li R, Gao M. 基于CNN模型的亚马逊象棋搜索算法设计[J] 数字技术与应用, 2022, 40 (02)：164-166.

[3] Guo T, Qiu H, Tong B, et al. 亚马逊国际象棋中多种游戏算法的优化与比较[C].2019年中国控制与决策大会(CCDC), 2019: 6299-6304.

[4] Jianning Q, Hongkun Q, Yajie W, et al. UCT技术在亚马逊计算机游戏中的应用[C].2016中国控制与决策大会(CCDC), 2016：6896-6899.

[5] Ju J, Qiu H, Wang F, et al. Research on Thread Optimization and Opening Library Based on Parallel PVS Algorithm in Amazon Chess[C].2021年第33届中国控制与决策大会(CCDC), 2021: 2207-2212.

[6] Li Z, Ning C, Cao J, et al. Amazon Chess Based on UCT-PVS Hybrid Algorithm[C].2021年第33届中国控制与决策大会(CCDC), 2021: 2179-2183.

[7] Meng D, Jianbo B, Yizhong Q, et al. Design of Amazon Chess Game System Based on Reinforcement Learning[C].2019中国控制与决策大会, 2019: 6337-6342.

[8] Tong B, Qiu H, Guo T, et al. Research and Application of Parallel Computing of PVS Algorithm Based on Amazon Human-Machine Game[C].2019年中国控制与决策大会(CCDC), 2019: 6293-6298.

[9] Chen X, Yang L. 亚马逊国际象棋中评价功能的研究[J]。计算机知识与技术, 2019, 15 (08)：224-226

[10] Li Z, Li Y, Ran G, et al. Design of Amazon chess game system based on PVS search algorithm [J] Smart computers and applications, 2018, 8 (05)：86-88.

[11] Wang C, Ding M. Interface design and implementation of personalized Amazon chess [J] Intelligent computer and application, 2017, 7 (02)：78-80.

[12] Zhang L. 基于最小搜索算法的亚马逊棋牌游戏系统研究[D] 东北大学, 2010.

[13] Metropolis N, Ulam S. The Monte Carlo Method[J].美国统计学会杂志, 1949, 44(247)：335-341.

[14] Coulom R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search[C].Computers and Games, 2007：72-83.

[15] Gelly S, Kocsis L, Schoenauer M, et al. The grand challenge of computer Go：蒙特卡洛树搜索和扩展[J].Commun.acm, 2012, 55(3)：106-113.

[16] Gelly S, Silver D. 计算机围棋中的蒙特卡洛树搜索和快速行动值估计[J].人工智能, 2011, 175(11): 1856-1875.

[17] Vinyals O, Babuschkin I, Czarnecki W M, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning[J].Nature, 2019, 575(7782)：350-354.

[18] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J].自然, 2016, 529(7587)：484-489.

[19] Berner C, Brockman G, Chan B, et al. Dota 2 with Large Scale Deep Reinforcement Learning[J]. arXiv, 2019.

[20] Chen J X. The Evolution of Computing： AlphaGo[J].Computing in Science & Engineering, 2016, 18(4)：4-7.

[21] Rocki K, Suda R. Large-Scale Parallel Monte Carlo Tree Search on GPU[C].2011年IEEE平行和分布式处理研讨会和博士论坛国际研讨会, 2011: 2034-2037.

[22] Shannon C E. XXII.为下象棋的计算机编程[J].The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 1950, 41（314）：256-275.

[23] 2004.冯恩，摩根斯顿，王维，等.博弈论与经济行为[M]北京：生活-读书-新知三联书店，2004.

[24] Browne C B, Powley E, Whitehouse D, et al. A Survey of Monte Carlo Tree Search Methods[J].IEEE Transactions on Computational Intelligence and AI in Games, 2012, 4(1)：1-43.

[25] Chaslot G, Bakkes S, Szita I, et al. Monte-Carlo Tree Search：游戏人工智能的新框架[J].AAAI人工智能和互动数字娱乐会议论文集, 2021, 4(1)：216-217.

[26] Świechowski M, Godlewski K, Sawicki B, et al. Monte Carlo Tree Search: a review of recent modifications and applications[J].Artificial Intelligence Review, 2022.

[27] Kocsis L, Szepesvári C. Bandit Based Monte-Carlo Planning[C].机器学习：ecml 2006, 2006: 282-293.

[28] Lieberum J. An evaluation function for the game of amazons[J].Theoretical Computer Science, 2005, 349(2)：230-244.

[29] Garland M. Parallel computing with CUDA[C].2010年IEEE并行与分布式处理国际研讨会(IPDPS), 2010.

[30] Després P, Jia X. A review of GPU-based medical image reconstruction[J].Physica Medica, 2017, 42: 76-92.

[31] Pratx G, Xing L. 医学物理学中的GPU计算： A review[J].Medical Physics, 2011, 38(5)：2685-2697.

[32] Bridges R A, Imam N, Mintz T M. Understanding GPU Power： A Survey of Profiling, Modeling, and Simulation Methods[J].ACM Comput.Surv., 2016, 49(3)：第41条。

[33] Jia X, Ziegenhein P, Jiang S B. 基于GPU的高性能计算的放射治疗[J].Physics in Medicine and Biology, 2014, 59(4)：R151-R182.

[34] Mittal S, Vetter J S. A Survey of Methods for Analyzing and Improving GPU Energy Efficiency[J].ACM Comput.Surv., 2014, 47(2)：第19条。

[35] Han T D, Abdelrahman T S. Reducing branch divergence in GPU programs[C].第四届图形处理单元上的通用处理研讨会论文集, 2011：第3条。

[36] Kim Y, Shrivastava A. CuMAPz：一个分析CUDA内存访问模式的工具[C].2011年第48届ACM/EDAC/IEEE设计自动化会议（DAC），2011年：128-133.

[37] Rosen P. A Visual Approach to Investigating Shared and Global Memory Behavior of CUDA Kernels[J].Computer Graphics Forum, 2013,32(3pt2):161-170.

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。

授权许可使用仅限于：吉林大学。于2023年5月5日02:38:49 UTC从IEEE Xplore下载。限制条件适用。