操作系统,作业 5

该作业包括两个编程问题,一个是 C 语言,一个是 CUDA 问题,用于您在作业 4 中说明的 ARC 系统。与往常一样,确保您的代码在提交之前（或在第二个程序的圆弧系统）。另外,请记住,您的工作需要单独完成,并且您的程序需要进行注释和一致的缩进。您必须记录您的来源,如果您使用来源,您仍然必须自己从头开始编写至少一半的代码。

1. (16 pts) 分页内存模拟 考虑通过使用 TLB 分页来支持虚

拟内存。虚拟地址由 18 位 (256KB) 组成。物理内存大小为 64KB,页/帧大小为 4KB。使用如下所示的页表和 TLB,请 1) 计算物理地址； 2)指示页表和TLB的变化，也指示何时发生TLB未命中;您应该使用 FIFO 替换作为您的 TLB 替换策略（从第一个条目开始）； 3) 为以下每个虚拟内存引用提供对内存引用的影响（完成或访问冲突）。您应该假设以下访问是连续发生的,并且应该使用来自先前访问的更新页表和 TLB。我们还假设我们对 TLB 使用回写策略,这意味着 TLB 上的副本仅在需要被替换时才与页表上的副本同步。并且如果访问无效,则不应将页面带入 TLB。

(a) 000001000100000101（读权限） (b) 000000000011000100
（写权限） (c) 000011000101100100（写权限） (d)
000010000101010100（读权限）

页码以最高有效位编码。状态位（有效、脏、只读、引用)如果为真则为 1,否则为 0。

提示：

dirty bit:修改后,相应帧的dirty bit标记为“1” reference bit:最近访问过一个帧后,该帧的reference bit标记为“1”

页表有效脏 0

虚拟页面	框架			只读	参考
0	2个	1个		1个	0
1个	0	1个	0	0	1个
2个	1个	1个	0	1个	1个
3个	5个	1个	0	0	1个
4个	3个	1个	0	0	0

TLB

虚拟页框有效脏 1				只读	参考
2个		1个	0	1个	1个
1个	0	1个	0	0	1个

作为参考,您的回答可能采用以下形式：

1个

000100001100000000 (读访问) -> 000100 001100000000 页
号为4,映射到物理内存中的第3帧

TLB未命中

物理地址=000011 001100000000 分页完成

页表有效脏 0

虚拟页面	框架			只读	参考
0	2个	1个		1个	0
1个	0	1个	0	0	1个
2个	1个	1个	0	1个	1个
3个	5个	1个	0	0	1个
4个	3个	1个	0	0	1个

条目“虚拟页面 4”的“使用”位设置为 1

TLB

虚拟页框有效脏 3 0			只读 0 0	参考
4	1	0		1
1	1	0		1

发生 TLB 替换:用新条目“虚拟页面 4”替换条目“虚拟页面 2”

将您的答案写入文本文件,将其转换为名为 hw5 p1.pdf 的 pdf 文件,然后将其提交至 Gradescope 上的 HW5 Q1。

2. (40 pts) 我们将使用 TCP/IP 套接字进行通信,用 C 实现一个多线程 Unix 客户端/服务器程序。服务器将让用户查看和更新包含小写字母（和空格,如拼字板)的二维游戏板。用户将能够向板上添加单词,垂直或水平,但任何新单词的字母必须与已经放置在板上的单词的现有字母相匹配。

为您提供了服务器框架以帮助您入门,scrabbleServer.c。您将完成此服务器的实施,添加对多线程和同步的支持,并构建一些用于存储板和响应客户端命令的表示。您不需要编写客户端程序;为此,我们将使用一个用于网络通信的通用程序 telnet。

一旦启动,您的服务器将永久运行,接受来自客户端的连接,直到用户使用 ctrl-C 将其终止。我们用作客户端的程序 telnet 将服务器的主机名和端口号作为命令行参数。此问题末尾的示例执行显示了如何像这样运行 telnet。连接到服务器后,telnet 只会将服务器发送的任何文本回显到屏幕,并将用户输入的任何文本发送到服务器。

拼字板

当用户启动服务器时,他们将给出两个命令行参数,面板中的行数和列数。这些必须是大于零的整数。如果服务器以无效参数运行,它应该打印以下用法消息并以退出状态 1 终止。

用法:scrabbleServer <行> <列>

服务器将维护板的表示,向其添加单词以响应用户命令,在被要求时打印当前板的副本,并检测新单词何时与板上已经放置的字母冲突。

客户端/服务器交互

当客户端连接到服务器时,它会使用提示符“cmd>”反复提示用户输入命令。起始代码支持提示用户和读取命令,但它只是将每个命令回显给用户。您将修改它以支持以下命令:

- 跨 rc 字

  此命令在板上水平放置一个新词。单词的左端从棋盘上给定的行 r 和列 c 开始。行号和列号都从零开始(板的顶行和左列)。给定的单词必须是 1 到 26 个小写字母的序列。它被放置在棋盘上,每个位置一个字符,从起始位置开始。

  如果给定位置在棋盘外,如果单词超出棋盘范围,如果单词包含小写字母以外的其他内容,或者单词中的某个字符与字符不一致,则命令无效已经放在板上了。

- 向下 rc 字

  此命令类似于 crover 命令,但它放置一个从给定起始位置向下的单词。它对板上的有效单词和单词放置有相同的规则。

- 木板

  此命令指示服务器为客户端打印板的当前状态。它的边缘应印有边框,边框由左右竖线、顶部和底部的破折号以及角落的加号组成。有关示例,请参见下面的示例执行。 · quit 这是请求服务器终止与该客户端的连接。此行为已为您实施。

无效的客户端输入

如果客户端发送无效命令,服务器将打印出一行“Invalid command”,忽略该输入行,然后提示输入下一个命令。

您可以假设每个用户命令都是在一行输入中给出的。您不必担心处理在同一行上给出的多个命令,或者一个命令分布在多个输入行。我们不会用这样的输入来测试你的程序。这应该使用户输入更容易解析。

远程登录客户端

对于这个程序,我们不需要编写单独的客户端程序。服务器只需要一个可以打开套接字连接、通过套接字发送用户输入并将从套接字读取的任何内容写入终端的客户端。有一些标准程序可以做这种事情。在此作业中,我们将使用一个名为 telnet 的工具,它已安装在大学机器上。如果您有自己的 Linux 机器,您应该能够安装一份 telnet。

您可以按如下方式运行 telnet。在这里，主机名是您运行服务器的系统的名称。端口号是您的服务器用于侦听连接的唯一端口号（见下文）。

telnet 主机名 端口号

客户端/服务器交互

客户端/服务器通信全部以文本形式完成。在服务器中，我们使用 fdopen() 从套接字文件描述符创建一个 FILE 指针。这让我们可以像使用 C 标准 I/O 库写入和读取文件一样发送和接收消息。当然，我们可以直接读取和写入套接字文件描述符，但使用 CI/O 函数应该会使发送和接收文本更容易一些。

部分服务器实现只是反复提示客户端输入命令并在客户端输入“退出”时终止连接。您将扩展服务器以处理上述命令。下面有一个示例执行，以帮助演示服务器应如何响应这些命令。

额外学分

您的服务器不应该检查用户输入的单词是否是真实单词。对于最多 8 分的额外信用，您可以让您的服务器检查以确保板上的所有单词始终有效，字典单词。

此作业的起始文件包括一个名为“words”的字典文件。这是通常安装在 Linux 机器上的字典文件的副本，位于路径 /usr/share/dict/words 下。

EOS Linux 机器没有安装拼写检查器，所以它们没有这个文件。因此，它包含在启动器中。如果你做了额外的功劳，在启动时，你的服务器应该在当前目录中查找一个名为“words”的文件，每行一个单词。该文件可以包含任意数量的单词，但每个单词的长度都不会超过 26 个字符。如果当前目录中没有 words 文件，您的程序应该仍然可以运行。它只是不会检查板上的单词以确保它们是有效的。

标准的 linux 字典有一些单词包含大写字母，甚至一些非 ASCII 字符。您可以在阅读“单词”文件时丢弃这些单词，也可以保留它们。无论哪种方式，他们都不会匹配拼字板上的任何单词，因为用户只能在板上放置小写字母。

为了获得额外的分数，您需要检查两个或多个字符的所有水平和垂直序列是否构成一个有效单词。您不需要检查在 across 或 down 命令中给出的字符序列是否构成有效单词，只需检查添加的字符在放置在板上后是否构成有效单词。例如，用户可以输入类似“across 0 1 ope”的命令。

“ope”本身不是一个词，但如果“ope”出现在已经放置的字母“c”、“h”、“m”或“r”之后，它仍然是一个合法的命令。

下面的示例执行中有一些额外信用行为的示例。

多线程和同步

在起始代码中，服务器仅使用主线程来接受新的客户端连接并与客户端通信。它一次只能与一个客户端交互。您将通过使其成为多线程服务器来解决此问题。每次客户端连接时，您都将创建一个新线程来处理与该客户端的交互。当它完成与客户端的对话时，该线程可以终止。使用

pthread 参数传递机制,为新线程提供与该客户端连接关联的套接字。1

由于每个客户端在服务器上都有自己的线程,当线程试图访问存储在服务器中的任何状态时（例如,当它们报告或修改面板时）,可能会出现竞争条件。您需要向服务器添加同步支持以防止潜在的竞争条件。您可以为此使用 POSIX 信号量或 POSIX 互斥锁。这样,如果两个客户端同时输入命令,服务器将确保它们的线程不能同时访问共享服务器状态。

分离线程

以前,我们总是让主线程与其创建的线程连接。在这里,我们不需要这样做。主线程一旦创建它就可以忘记它。每个新线程都可以与其客户端通信,然后在完成后终止。

为了使其正常工作,在创建线程后,服务器需要使用 pthread detach() 将其分离。
这告诉 pthreads 在给定线程终止后立即为给定线程释放内存,而不是为了另一个预期加入它的线程的利益而保留它。

缓冲区溢出

在当前状态下,服务器在读取来自客户端的命令时容易受到缓冲区溢出的影响。
您需要修复现有代码中的此漏洞,并确保您添加的任何新代码中都没有潜在的缓冲区溢出。2您的服务器只需要能够处理上面列出的命令。如果用户试图输入一个很长的名字、一个很长的命令或一个很长的单词,你应该检测到这一点,忽略它并打印“Invalid command”消息。

输入刷新

由于我们创建文件指针的方式,当服务器开始将输出写入套接字时,缓冲输入似乎被丢弃了。这最终相当方便,但在您编写代码来解析和响应用户命令时需要牢记这一点。例如,在开始打印响应之前,您需要阅读用户命令的所有部分（否则,您将丢失尚未阅读的部分）。同样,假设用户输入一个很长的单词作为命令的一部分。一旦检测到单词太长,您可以打印出“无效命令”消息,其余的命令将被丢弃。您不必编写代码来读取和丢弃命令的其余部分。

端口号

由于我们可能在多用户系统上进行开发,因此我们需要确保我们都使用我们的服务器可以侦听的不同端口号。这样,一个学生编写的客户端就不会意外地尝试连接到另一个学生编写的服务器。

为了帮助防止端口号冲突,我为每个学生分配了自己的端口号。您可以通过在我们的 Moodle 页面上检查名为“端口号分配”的分配来找到您随机分配的端口号。这实际上不是您要上交的作业,但单击此作业应该可以让您下载名为 port-number.txt 的反馈文件。该文件应该告诉您应该使用的端口号。

1请记住,将参数传递给新线程的方式很容易出错。在这里要小心。
2如您所知,缓冲区溢出在服务器中非常糟糕。它可能允许世界任何地方的攻击者获得访问权限
到运行服务器的系统。

对该作业的评分将包括检查以确保您使用正确的端口号,因此请务必在开始开发之前从该文件中获取分配给您的端口号。

关于 EOS Linux 主机上套接字的悲惨事实

由于我们使用 TCP/IP 进行通信,我们终于可以在不同的主机上运行我们的客户端和服务器程序。你应该能够在一台主机上运行你的服务器,然后在世界上任何其他连接到 Internet 的系统上启动 telnet,在命令行上给它服务器的主机名和你的端口号。但是,如果您在典型的大学 Linux 机器上尝试这个,您可能会失望。这些系统具有阻止大多数 IP 流量的软件防火墙。因此,如果你想在大学系统上开发,你仍然需要在同一台主机上运行客户端和服务器。如果你有自己的 Linux 机器,你应该能够在上面运行你的服务器并从任何地方连接(取决于你的网络设置方式)。

如果您想在虚拟计算实验室 (VCL) 中使用一台机器,您应该能够以管理员身份运行命令,这样您就可以禁用软件防火墙。我成功预订了 CentOS 7 Base(64 位 VM)系统。登录、上传和构建我的服务器后,我运行以下命令以允许通过我的端口进行 TCP 连接。然后,如果我在虚拟机上运行服务器,我就能够连接到它,即使是在校外。

```
sudo /sbin/iptables -I INPUT 1 -p tcp --dport my-port-number -j ACCEPT
```

此外,如果您的服务器崩溃,您可能暂时无法绑定到您分配的端口号,并显示错误消息“无法绑定套接字”。只需等待一两分钟,您就可以再次运行服务器。

执行

您将通过从启动器扩展文件 scrabbleServer.c 来完成您的服务器。你需要编译如下:

```
gcc -Wall -g -std=gnu99 scrabbleServer.c -o scrabbleServer -lpthread
```

样本执行

您应该能够使用任意数量的并发客户端运行您的服务器,每个客户端都由服务器中自己的线程处理。下面,我们将查看在三个终端会话中运行的命令,服务器在左侧栏中运行,客户端在其他两栏中运行(请务必使用我们自己的端口号,而不是 28123)。我交错输入/输出以说明与每个客户端交互的顺序,并且我添加了一些注释来解释发生了什么。如果您在自己的服务器上尝试此操作,请不要输入注释,因为服务器会认为它们是无效命令。

在这里,我在同一台主机上运行所有三个程序,但是如果您使用一台可以控制防火墙的机器,您应该能够在三台不同的主机上运行这个例子。

```
# 使用 8 x 12 板运行服务器。
$ ./scrabbleServer 8 12
```

```
# 运行一个客户端并放置一个词。 $ 远程登录本地
主机 28123
cmd> 跨越 3 4 阳光
```

运行另一个客户端并检查面板。 $ 远程登录本地主机 28123

命令>板
+------------+
|
|
|
| | |阳光| | | | |
|
|
|
|
+------------+

往下写一个词。 $ cmd> 向下 3 8 快乐

依次检查棋盘
客户的话。
命令>板
+------------+
|
|
|
| | |阳光| | p | p |
A
是 |
|
|
+------------+

尝试一个带有非法字符的单词。 cmd> 跨越 2 2 CAPITAL

命令无效

尝试从棋盘开始的单词 cmd> down 1 12 a

命令无效

尝试一个超出边缘的词。 cmd> down 5 2 过多

命令无效

尝试一个与 # 现有字符不匹配的词。 cmd> across 7 4
sleeping 无效命令

现在,一个确实匹配 # 现有字符的单词。 cmd> 跨越 6 4 睡
觉

检查板子,然后退出。命令>板

+------------+

| | |阳光|一个 | p
|睡觉|是 |

+------------+

命令>退出

其余的是额外的信用测试。

尝试一个不在字典中的单词。 cmd> 跨越 0 0 很多

命令无效

现在,一个有效的词。 cmd>
跨越 0 0 狂热者

一个有效的词,但它在板上创建了一个 # 无效的词。

cmd> 跨越 0 5 圆环
命令无效

走另一条路也没关系。 cmd> 向下 0 5 圆环

命令>板
+------------+
|狂热者| | | |
欧
r
| | |阳光| | p |睡
在
觉|是 |

| |
+------------+

to 是一个有效的词,但 ot（向下)不是。 cmd> 跨越 1 4 到

命令无效

但是,on 和 no 都是有效的。 cmd> 跨越 1 4 否

命令>板
+------------+

```
|狂热者| | | | |
          不
            r
          | | |阳光| | p |睡
            在
                  觉|是 |

| |
+-----------+

# 退出客户端,然后退出服务器。命令>退出
```

```
# 杀死服务器。
Ctrl-C
```

提交您的作品

完成后,在名为 HW5 Q2 的作业下提交 scrabbleServer.c on Grade – 范围。

3. (40 分)对于这个问题,您将在 GPU 上实现作业 1、2 和 3 中的 maxsum 程序,其中有很多线程。我们将在 NC State 的 arc 集群上使用 CUDA 框架。你知道电弧;作为作业 4 的一部分,你应该已经激活了你的 arc 帐户。Arc 有大约 100 台运行 Linux 的主机,每台主机都有一个通用 CPU 和一个用作高度并行协处理器的 GPU。

Arc 文件系统

Arc 不与其他大学机器共享用户帐户或文件系统。要连接到它,您需要从另一个大学系统(例如 remote.eos.ncsu.edu)ssh 登录,就像您为作业 4 激活 arc 帐户时所做的那样。要成功连接,您需要使用在作业 4 中制作密钥对的同一台机器,或者如果您在具有 NFS 的机器上制作密钥对,则使用可以访问 NFS 的机器。

到期日分布

arc 系统有很多节点,但它不足以让我们所有人同时拥有一个。为确保您有机会完成此问题,我正在为此作业实施软截止日期。尽早完成此问题可以获得最多 8 分的额外学分。或者,如果您迟交了您的解决方案,迟交罚金将逐步应用,而不是一次全部应用。我希望这将有助于确保我们不会在同一天晚上都尝试使用电弧系统。 Arc 有很多计算节点,但如果我们都试图同时使用它,一些用户会感到失望。

下表显示了根据您早晚提交解决方案的额外学分或迟交罚金。

奖金/罚款提交时间	
	至少提前 96 小时 +8 +6
	至少提前 72 小时
	至少提前 48 小时 +4 +2
	至少提前24小时-2
	最多迟到24小时
-4	最多迟到48小时

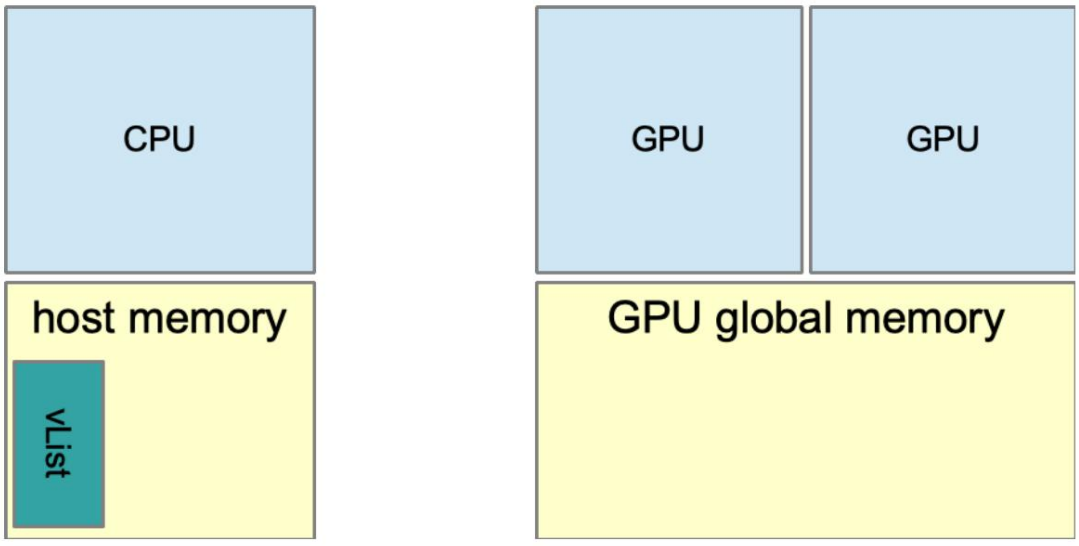
工作分区

对于此任务,我们将在线程之间静态分配工作。您可以根据需要划分工作,并且与家庭作业 3 不同,您可以在任何工作线程开始运行之前访问整个值序列。每个线程都有一个唯一的索引,i。它会在线程启动后立即根据网格和块维度计算该索引。你可以让线程号 i 查看序列中从位置 i 开始的序列,或者它可以检查所有以位置 i 结束的序列,或者如果你有更好的主意的话。如果在命令行上给出报告选项,您的线程应该在发现它们时报告范围,就像我们在之前的作业中所做的那样。 CUDA 对此提供支持,即使 GPU 线程在不同的设备上运行。

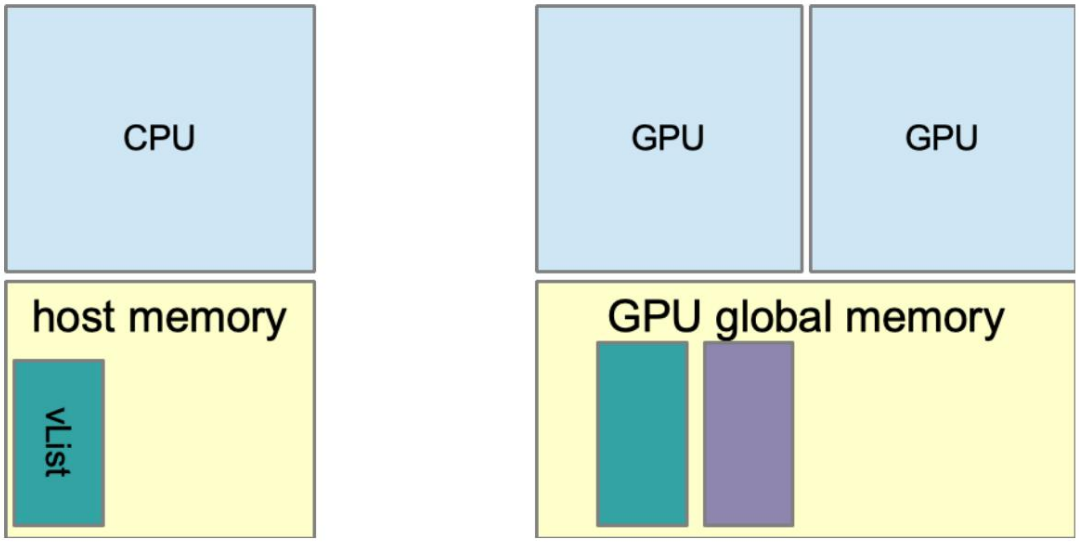
当一个 worker 完成后,它将把它找到的最大总和复制到 CUDA 全局内存中分配的数组的元素 i 中。所有线程执行完毕后,main 中的代码会将所有本地最大和复制到主机内存中,并进行比较以得到最终的最大和。

对于之前的分配,我们让每个进程或线程负责序列中的几个元素。我们不想创造太多工人。这在 GPU 上不是问题。它可以同时处理大量线程,如果我们请求的线程数量超过 GPU 一次可以执行的数量,CUDA 软件将负责在线程之间共享处理元素。

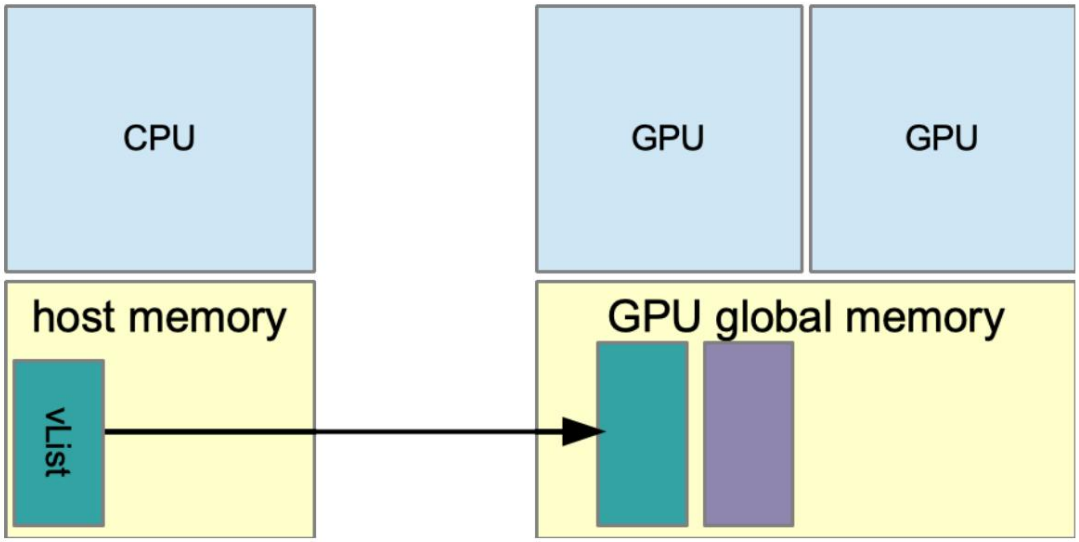
我正在为您提供框架 maxsum.cu,以帮助您入门。该程序将序列读入主机 vList 上的数组。



您需要添加代码以在设备上分配两个数组,一个数组用于保存主机序列的副本,另一个整数数组用于保存每个线程计算的本地最大总和。

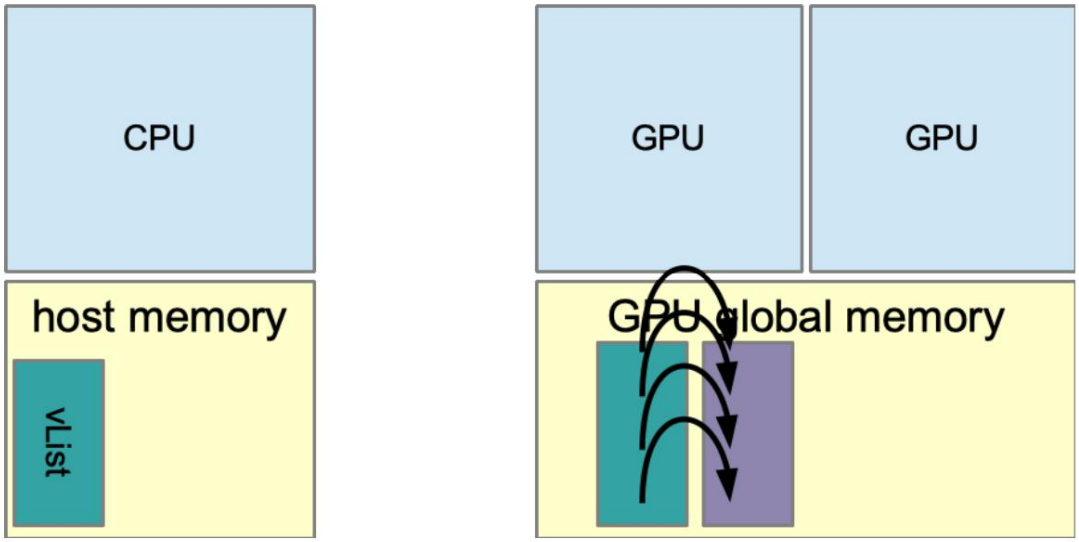


然后您需要代码将序列从主机复制到设备,以便 GPU 上的线程可以访问它。

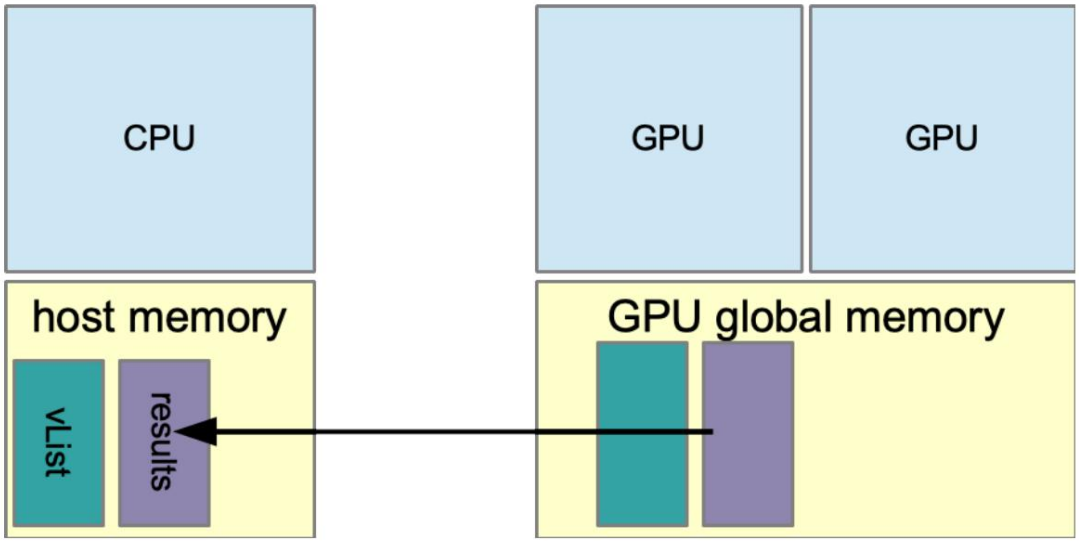


现在设备已准备就绪,您需要完成 checkSum() 内核的实现。
每个线程将负责检查序列中的某些范围。完成后,它将最大的总和存储在 i 当您从主机运行该内核时,它将为每个元素
启动一个线程并等待它们完成计[1] 结果数组在设备内存中的索引。

算它们的部分。



然后,主机将需要分配主机内存 (malloc) 并将结果复制到其中。



然后,主机上的主线程可以快速比较所有线程的总和,并以以下格式打印出最终的最大总和:

我是最伟大的:n

线程实现

您可以编写大部分 checkSum() 函数,即实际在 GPU 上运行的代码。我已经为此功能编写了一些起始代码。现在,每个线程只计算一个唯一的索引,并确保它有一个可以处理的值,然后返回。您需要向内核函数添加参数,以传递指向输入和输出数组的指针,找到每个线程部分的最大总和并记录结果。

3如果block size没有将sequence中的number元素均分,那么最后一个block中可能会有一些thread这实际上没有任何关系。

在弧上工作

要访问 arc,您必须从另一台大学机器登录,您已经在其中存储了用于激活 arc 帐户的密钥对（可能是 remote.csc.ncsu.edu）。使用您的 ssh 客户端（例如,putty 或 mobaXterm 或 ssh)连接到 remote.csc.ncsu.edu,然后您可以从那里访问 arc。

上传您的文件

arc 集群上的节点共享一个文件系统,但它们不使用大学 NFS。要将文件放到机器上,您需要从另一台大学机器上传它们。你可以只将你的文件上传到大学的一台 Linux 机器上,然后使用 sftp 程序从那里传输它们。从一台远程 Linux 机器（不是来自 ARC）,像下面这样的命令应该可以让你上传文件。从包含 maxsum.cu 和输入文件的目录开始：

sftp arc.csc.ncsu.edu 网站

sftp> put maxsum.cu 上传
maxsum.cu ... sftp> put input-1.txt
上传 input-1.txt sftp> put
input-2.txt 上传 input-2.txt
<you get the idea> sftp> 再见

登录 Arc

在 remote.eos.ncsu.edu 登录到其中一台机器后,您应该能够输入以下内容以登录到 arc 上的头节点：

ssh arc.csc.ncsu.edu 网站

编辑、编译和运行

当您登录到 arc 时,您连接到一个甚至没有 GPU 的头节点。您需要从头节点连接到其中一个计算节点。 Arc 使用批处理软件将作业分配给计算节点。以下命令可让您从头节点连接到其中一个计算节点。每次你想在 arc 上工作时,你应该只需要运行这些命令之一。一旦一个可用,它就会在其中一个计算节点上给你一个交互式提示。下面的评论告诉你每个节点上你会得到什么样的 GPU。根据系统的繁忙程度,您可能需要稍等片刻才能收到提示。还要确保从 shell 提示符运行它（而不是从上面描述的 sftp 命令内部）：

有 13 个节点带有 nVidia RTX 2060
srun -p rtx2060 --pty /bin/bash

有 17 个节点带有 nVidia rtx2070。
有一次,当我尝试使用其中一个 rtx2070 系统时,它在我第一次调用 cuda 函数时失败了。后来,当我重新登
录到另一个 rtx2070 系统时,它工作正常。这让我觉得

可能这些节点之一的 GPU 有问题,或者可能 # 它没有正确配置。如果你有这个问题,你可能 # 想尝试另一个系统。 srun -p
rtx2070 --pty /bin/bash

有 2 个节点带有 nVidia RTX 2080
srun -p rtx2080 --pty /bin/bash

有 21 个节点带有 nVidia RTX 2060 Super srun -p rtx2060super --pty /bin/bash

有 2 个节点带有 nVidia RTX 2080 Super # 当我尝试其中一个系统时,我的第一次 CUDA 调
用也出现错误。 srun -p rtx2080super --pty /bin/bash

仍然有一些系统使用较旧的 nVideo gtx480 显卡 # 卡。如果您无法连接到上述较新的系统之一,您 # 可以尝试使用这些较旧的系统
之一。 srun -p gtx480 --pty /bin/bash

在计算节点上出现提示后,您应该仍然能够看到您上传的文件。 Arc 过去要求您在使用 CUDA 进行任何工作之前输入以下命令,但现在似乎不再需要这样做。如果您尝试运行 nvcc 编译器并收到一个抱怨,即您的 shell 无法找到具有该名称的程序,您可以尝试运行此命令以查看是否修复了它。

模块加载cuda

要编辑您的程序,您可以在 NFS 中进行编辑,并在每次进行更改时使用 sftp 将修改后的文件复制到 arc。如果您习惯于直接在 Linux 机器上编辑,那么直接在 arc 机器上编辑您的源文件可能更容易。那里有编辑器 vim 和 emacs,但看起来并没有安装真正简单的编辑器,如 pico 或 nano（无论如何,vim 和 emacs 是更好的编辑器,但需要一点脑力才能准备好使用它们）。

准备好构建程序后,您可以按如下方式编译:

nvcc -I/usr/local/cuda/include -L/usr/local/cuda/lib64 maxsum.cu -o maxsum

当您准备好运行时,请确保您在计算节点上（即,您已经运行了上面的 svrun 命令之一）。您可以像运行普通程序一样运行您的程序。这个版本的程序仍然有可选的报告标志,但不需要告诉它使用多少线程。它只是为输入序列中的每个值创建一个线程。像往常一样,如果启用了报告,那么您的线程将报告他们找到的最大总和。请注意,在这里,您可以使用从 0 开始的整数为每个线程编号,而不是使用 tid。

$ ./maxsum report < input-1.txt 我是线程0。我找到的最
大总和是2。
我是线程 1。我找到的最大总和是 5。
我是线程 2。我找到的最大总和是 3。
我是线程 3。我找到的最大总和是 7。

我是线程 4。我找到的最大总和是 8。
我是线程 5。我找到的最大总和是 15。
我是最高的:15

当您想要测量程序的执行时间时（尤其是在较大的输入上），可以使用 time 命令。

$ time ./maxsum < input-5.txt 最大总
和:227655

真正的?米？ 。？ ？ ?s
用户 ?m?.???s
系统 ?m?.???s

记录执行时间

在 maxsum.cu 文件的顶部，我在 input-5.txt 文件上添加了用于记录该程序运行时间的注释。一旦您的程序开始运行，请在该输入文件上计算其执行时间，并输入您在此注释中获得的（实际）执行时间。此外，报告您使用的计算节点类型（即它具有的 GPU 类型）。测量执行时间时不要使用报告标志；那只会减慢速度。您可能会注意到启动时间似乎很长，因此将工作推到 GPU 上可以获得最大的输入文件。 arc 上不同的计算节点可能有不同的能力（至少，过去是这样），所以如果你从一些同学那里得到不同的执行时间，不要担心。无论哪种方式，我希望您会发现这些运行时间比您在通用 CPU 上获得的运行时间更快。

注销

当您准备好注销时，您将有多个级别可供注销。从计算节点，您可以键入 exit 以回退到 arc 集群的头节点。从那里输入 exit 会让你回到你连接的任何机器（可能是 remote.eos.ncsu.edu）。多一个出口应该能让你出去。

提交您的作品

完成后，将工作程序的副本 maxsum.cu 提交到 Gradescope 上名为 HW5 Q3 的作业，并在顶部填写运行时。