并行 Floyd 最短路径算法

一、项目成员

MC

ZHY

二、难点与优化思路

首先，对 N 个节点的有向图，我们定义数据结构如下：

①邻接矩阵 A 。 A_{ij} $(0 \le i \le N-1, 0 \le j \le N-1)$ 存储节点 v 到节点 w 的权值。

②距离矩阵 D 。 D_{ij} $(0 \le i \le N-1, 0 \le j \le N-1)$ 存储节点 v 到节点 w 的最短路径长度，

③路径矩阵 P 。 P_{ij} $(0 \le i \le N-1, 0 \le j \le N-1)$ 存储节点 v 到节点 w 间最短路径经过的节点。

难点一： 确定 SIMD 可并行化部分。

Floyd 算法时间复杂度为 O（N^3），耗时巨大，我们需要确定可并行化部分。

根据 Floyd 算法的递推式 $D_{ij} = \min_{0 \le k \le N-1} (D_{ik} + D_{kj}, D_{ij})$ ，我们确定迭代变量 k 的 N 次迭代之间有依赖关系无法并行，然而每次迭代内部变量 i 和 j 的 N^2 次迭代却可以并行化，依据 GPU 异构平台 SIMD 的特点，我们接下来对这一部分进行并行化，时间复杂度由 O（N^3）优化至 O（N^2）。

难点二：提高算法空间局部性

我们发现，若每轮 k 迭代直接采用 $(N\text{-}1)^2$ 个 GPU 线程计算，会导致访问 GPU 的内存空间跨度太大，造成了大量的读写开销，如何提高算法的空间局部性便成为关键问题。

基于此，我们提出提出通过分块技术提高算法的空间局部性，将每一个块存入 GPU 的共享内存中，使每一次都从块中读取所需要的数据，从而使数据的读取速度尽可能快。算法描述如下：

先假设 GPU 能存下全部节点。我们规定每个分块的大小为 $TILE\_WITH$ ，注意，N 可能不会被 $TILE\_WITH$ 整除，因此特殊地，我们令在第 $N\text{-}1$ 列的分块宽度为 $knum(0 < knum <= TILE\_WITH)$,令在第 $N\text{-}1$ 行的分块高度为 $knum(0 < knum <= TILE\_WITH)$ 。最终的分块个数 B 为 $(N + TILE\_WITH\text{-}1)/TILE\_WITH$ 。这样，原有的 N 次迭代的算法被我们转为 B 次迭代的算法，每次迭代一个分块矩阵，设迭代变量为 k 。

第 k 次迭代内部分为 3 步：
①依据 k -1 次计算得到的主模块，来计算 k 次的主模块。
②依据 k 次计算的主模块，来计算与主模块同行、同列的其它模块。
③依据 k 次计算的主模块与更新后的与主模块同行、同列的其它模块，来计算剩余的其他模块。
我们以 $B=3$ 时为例，算法如下图所示：

难点三：GPU 内存空间不足

我们的算法假设距离矩阵 D 和路径矩阵 P 都存在 GPU 全局内存中，这显然是不现实的，因为我们的节点数的部分测试数据达到了了 50000 个节点，这已经超出了 GPU 最大的存储容量。

故所以我们接下来在分块并行算法的基础上考虑分批次迭代读入 GPU 进行运算。

难点四：提高算法的时间局部性

若采用最原始的分批次处理方法，即本次子矩阵运算完后，又读下一部分未运算的子矩阵将其覆盖，然后开始下一批次的计算。这种方法显然会在主机和设备之间的内存拷贝上耗费大量的时间。如何增强算法的时间局部性，使每次在主机和设备之间传输的数据尽可能得到充分的使用，从而减小每次主机和设备之间来回传送的数据量便成为问题。

由此，我们提出内存空间复用技术。即在迭代过程中，始终保证大多数数据存储在设备内存中，并在不同的k轮迭代中得到反复使用，并通过少部分数据在主机和设备之间的来回传送完成分批次更新任务，从而提高算法的时间局部性。该算法描述如下：

我们允许GPU在列方向上存储所有的N个节点，规定GPU在行方向上最多只能存储$BigPart$个$Block$（这里的$Block$即$HipC$设置的$Block$，每个$Block$维度为$TILE_WIDTH \times TILE_WIDTH$。总共需要的$Block$数为$B$，剩下的不能一次性存下的$Block$数设为$Smallpart = B - Bigpart$，又设$Midpart = BigPart - SmallPart$。这样，我们将储存在主机内存中的矩阵划分为行方向上的块长度分别为$SmallPart$，$Midpart$，$SmallPart$的3个矩阵，编号为$1,2,3$。其中第3号矩阵的实际数据可能并没有存储满。我们的算法是的2号矩阵（也就是体积最大的$Midpart$）从初始化时拷贝进入设备内存之后，将始终保存在设备内存中，通过1号矩阵和3号矩阵不断地在主机内存和设备内存之间来回传送完成分批次读写的工作。算法描述如下：

①将经过初始化的$1,2$号矩阵读入设备。

②开始迭代总次数为B的迭代步骤。其中迭代变量为k，k从0开始计数：

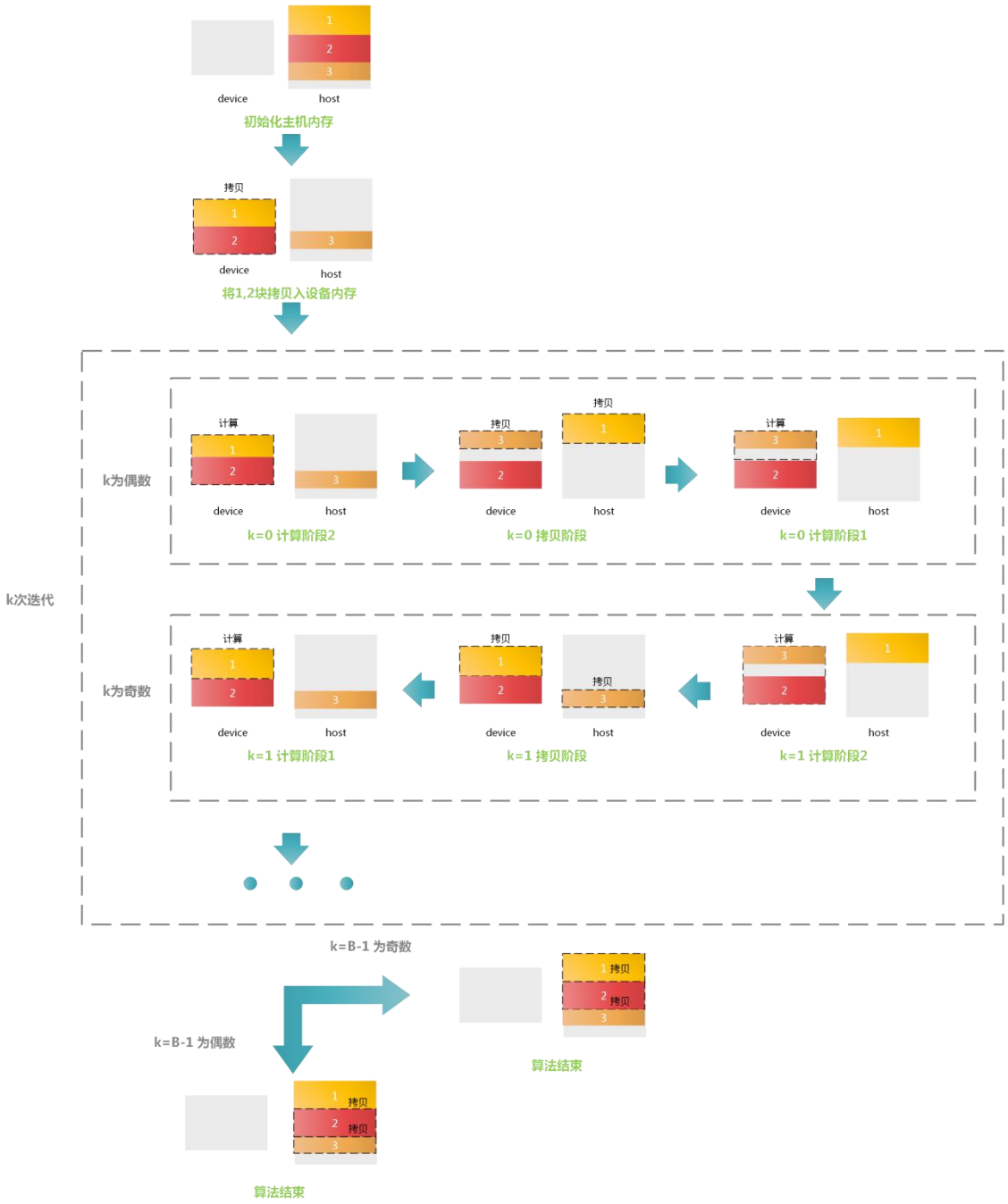
若k为偶数，则计算设备内存中的$1,2$矩阵，然后交换设备内存中的2矩阵与主机内存中的3矩阵，然后计算设备内存中的3矩阵。

若k为奇数，则计算设备内存中的$3,2$矩阵，然后交换设备内存中的3矩阵与主机内存中的1矩阵，然后计算设备内存中的1矩阵。

③当最后一次迭代完后，若$k = B\text{-}1$为奇数，则将设备内存中的$1,2$矩阵拷贝入主机内存，算法结束。

若$k = B\text{-}1$为偶数，则将设备内存中的$3,2$矩阵拷贝入主机内存，算法结束。

该算法示意图描述如下：



三、源代码与文档

我们给出的 Floyd.cpp 源代码已经包含了数据的生成，算法本身，和算法计时的部分，由随机数随机生成图的数据，然后调用 shortestPath_floyd 函数完成计算并计时，只需修改宏定义中的 NUM_NODE 即可进行运行，算法的注释附在源代码中。

四、建议代码运行的步骤

4w 点计算过程中，会占用几乎所有内存，建议在节点负荷小的时候进行

五、答题效果

节点个数	80	1024	5760	40000
耗时(h)	0.00015833	0.00016389	0.00021111	0.53333333

六、创新性与意义

创新性：本算法充分发挥 GPU 的 SIMD 特性，对算法的部分循环进行并行化。具有以下创新性：

①通过分块技术提高算法的空间局部性，将每一个块存入 GPU 的共享内存中，使每一次都从块中读取所需要的数据，从而使数据的读取速度极可能快。

②通过内存复用技术提高算法的时间局部性，使每一次数据的传输尽量完成更多的计算，使数据传送的效益最大化，从而进一步提高并行加速比。

意义：该算法在异构 GPU 的基础上，在保持图的基本特性的情况下，不断改进现有的图的存储结构，以充分发挥 GPU 计算平台的计算性能，完成了大规模数据的计算，同时具备高并行加速比，从而对于大规模数据的高性能计算的实践应用具有重要意义。

七、成果可应用到哪些行业应用

该算法可以完成大规模最短路径的查找，也可以为大规模物流配送中心的选址提供重要依据，预计在地图导航，物流配送领域,电商领域和计算资源调度领域有广阔的应用前景。