大作业报告: Hines 算法的 CUDA 实现

鄢语轩

June 28, 2020

1 Hines 算法

1.1 背景和原理

Hines 算法是一种用来求解神经元电压传导的算法, 是计算机脑模拟的一个步骤. 考虑一组树形神经元构成的 Hines 系统, 其中的每个神经元 i 有母节点 p_i 和子节点 $\{s_{i,j}\}$, 它的第 $n+1$ 个时间步的电压 V_i^{n+1} 满足方程:

$$u_i V_{p_i}^{n+1} + d_i V_i^{n+1} + l_i \sum_j V_{s_{i,j}}^{n+1} = r_i. \tag{1}$$

方程中的系数和上一个时间步的电压有关, 满足如下关系:

$$\begin{aligned} d_i &= c_i - u_i - l_i, \\ r_i &= c_i V_i^n - u_i (V_{p_i}^n - V_i^n) - l_i \sum_j (V_{s_{i,j}}^n - V_i^n). \end{aligned} \tag{2}$$

Hines 算法的作用是求解 Hines 系统的更新, 即 d_i 和 r_i 的更新.

1.2 算法的串行实现

一种可行的串行算法实现的伪代码展示如下. [1]

Algorithm 1 Hines algorithm.

```
1: void solveHines(double *u, double *l,   double *d,
2:                 double *rhs,      int *p, int cellSize)
3: // u → upper vector, l → lower vector
4: int i;
5: double factor;
6: // Backward Sweep
7: for i = cellSize − 1 → 0 do
8:     factor    = u[i] / d[i];
9:     d[p[i]]   -= factor × l[i];
10:    rhs[p[i]] -= factor × rhs[i];
11: end for
12: rhs[0] /= d[0];
13: // Forward Sweep
14: for i = 1 → cellSize − 1 do
15:    rhs[i] -= l[i] × rhs[p[i]];
16:    rhs[i] /= d[i];
17: end for
```

2 CUDA 并行实现的思路

我的 CUDA 并行针对的是多组 Hines 系统同时运行的情况, 并不能在单个 Hines 系统上取得加速. 代码中, 我要求运行的组数必须是 32 的倍数.

在我的实现中, GPU 的每个线程处理一个 Hines 系统. 之所以不用多个线程处理一个系统, 是因为 Hines 算法的执行中, 各个节点之间不独立. 在 “backward sweep” 部分, 子节点需要比母节点先更新; 而在 “forward sweep” 部分, 依赖关系相反. 如果用多个线程执行单个的系统, 会有较大的同步开销.

CUDA 程序设计中, 线程块数量和大小的选取对于性能意义重大. 为了最大程度发挥 GPU 的性能, 线程块的数量应当不小于 GPU 的流处理器 (streaming multiprocessors, SM) 的数量, 线程块中线程的数量应当不小于 32, 并是 32 的倍数. 实际操作中, 常常将线程块中线程的数量选取为 128 或 256. [2]

在我的实现中, 线程块中线程的数量选取范围为 32 或 64 或 128 或 256. 在线程块的数量不小于 GPU 的 SM 数量的情况下, 线程块中线程数量取可能的最大值. 不过不能满足线程块数量的条件, 则选取 32 个 threads.

3 性能分析

3.1 运行时间

我选取了几个 test case, 比较了串行和并行的运行时间. 测量的运行时间只包括 Hines 算法的执行和内存拷贝. 对于较小的 test case, 结果如 Table 1 所示. 较小的 test

Table 1: 较小的 test case 运行时间

Test Case	节点个数	运行组数	串行运行时间 (s)	并行运行时间 (s)	加速比
7	74277	512	0.80	0.55	1.45
7	74277	1024	1.67	0.94	1.78
7	74277	2048	3.82	1.74	2.20
8	181851	512	2.91	1.18	2.46
8	181851	1024	5.65	2.02	2.80
8	181851	2048	11.06	2.39	4.63

Table 2: 较大的 test case 运行时间

Test Case	节点个数	运行组数	串行运行时间 (s)	并行运行时间 (s)	加速比
9	568449	64	1.12	1.64	0.68
9	568449	128	2.08	1.88	1.11
9	568449	256	4.18	2.37	1.76
9	568449	512	7.88	3.25	2.42
10	921601	64	1.94	2.63	0.74
10	921601	128	5.65	3.00	1.88
10	921601	256	7.39	3.77	1.96
11	2752001	32	2.94	7.41	0.40
11	2752001	64	5.56	7.78	0.71
12	5728200	32	6.09	15.39	0.40

case 允许较大的线程数, 因而带来了很好的加速比. 从这个结果中, 我们可以知道加速比和 test case 的节点个数以及运行的组数均正相关.

对于更大的 test case, 受到 GPU 的内存限制, 我们只能使用相对少量的线程, 从而导致加速比受到限制. 数据列在 Table 2. 我们看到只有在线程数量不低于 128 的情况下, GPU 才有大于 1 的加速比.

线程数量对加速比的限制是可以预见的. 比如总线程数为 32 的情况下, GPU 实际上只有一个 SM 在工作. 而在节点数量很大的情况下, host 和 device 之间的内存拷贝严重影响性能. (相比之下, 串行版本的内存拷贝只发生在本地.)

为了更好地比较测试结果, 我将所有测试情形的加速比呈现在了 Figure 1 中.

3.2 精度

经过对比, 串行和并行的计算结果有很小的差别. 误差在 10^{-5} 以下.

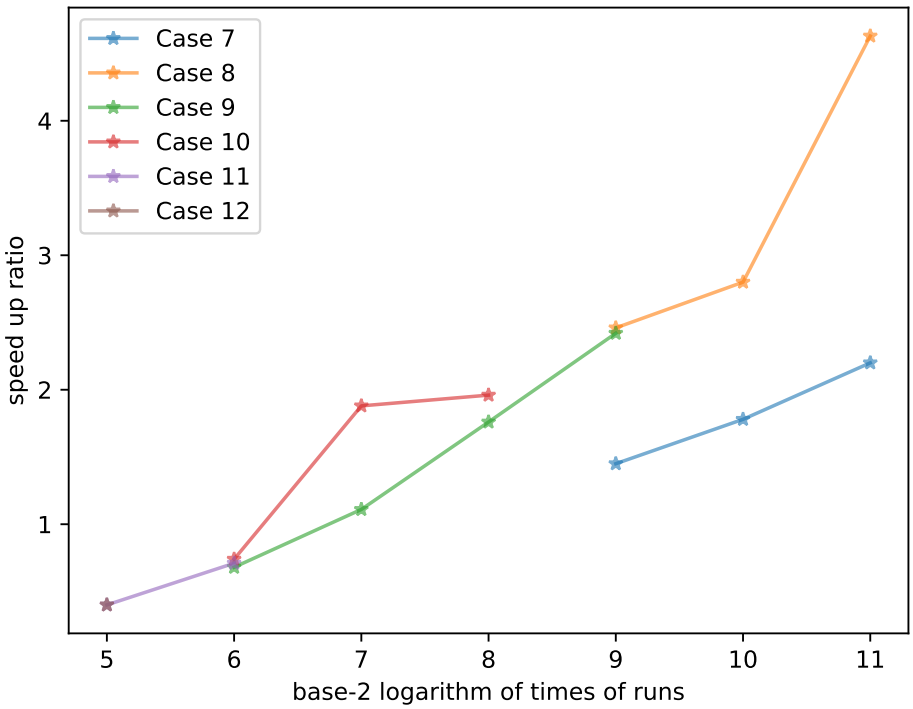

Figure 1: 不同 test case 和运行组数的加速比

4 可取的改进方式

4.1 内存布局

在我的实现中, 每个线程访问的数据地址是连续的. 这种实现的优点是单个线程访问内存的时候不用频繁地跳跃, 但缺点是不能利用聚合度低, 即相邻线程访问的内存地址不连续. 一种可行的改善方法是使用 “Block-Interleaved” 的方法.[1] 这种方法的思路是把线程分成 “块”, 在每一个 “块” 中内存交替地分配给其中的线程.

4.2 多 GPU 并行

对于大 test case 而言, 限制运行性能的一个重要因素是有限的 GPU 内存限制. 将程序在多 GPU 上并行可以改善这个限制.

References

[1] P. Valero-Lara, I. Martínez-Perez, A. J. Peña, X. Martorell, R. Sirvent, and J. Labarta, “cuHinesBatch: Solving Multiple Hines systems on GPUs Human

Brain Project". Procedia Computer Science, vol. 108, pp. 566–575, 2017, doi: https://doi.org/10.1016/j.procs.2017.05.145.

[2] N. Matloff, "Programming on Parallel Machines".

A 测试平台

本报告使用数据是 Tesla K80 上测试得到的. 并行部分采用的编译器版本为 nvcc, release 10.1, V10.1.243.