单源最短路径并行算法的研究

任旭彤

摘要

本文实现了基于优先队列优化的 Dijkstra 单源最短路径算法和 Δ-Stepping单源最短路径算法。并对其效率、并行性进行实验比较，以进一步掌握并行程序设计原理。

1. 算法介绍

1.1. Dijkstra 算法

Dijkstra 算法由 E.W.Dijkstra 于 1959 年提出，算法主要解决有向图中单个源点到其他顶点最短路径的问题，适用于非负边权图。Dijkstra 算法应用了贪心算法模式，其主要特点是每次迭代时选择的下一个顶点是标记点之外距离源点最近的顶点。给定图的总顶点数 $\mathbf{n}$ 和 总边数 $\mathbf{m}$，朴素 Dijkstra 算法的时间复杂度是 $\mathbf{O(n^2)}$，如果用 Fibonacci-heap 可以优化到 $\mathbf{O(m + n\ log\ n)}$。

1.2. Δ-Stepping 算法

Δ-Stepping 算法由 U. Meyer 于 2003 年提出，算法结合了 Dijkstra 和 Bellman-Ford 算法的思想，利用桶间排序保证算法效率，并利用桶内数据的独立性提高并行性。算法流程如图 **1** 所示。Δ-Stepping 算法可以看作是 Dijkstra 和 Bellman-Ford 算法在效率和并行性上的平衡。当 $\Delta = 1$ 时，算法等效于 Dijkstra 算法，当$\Delta \to \infty$ 时，算法等效于 Bellman-Ford 算法。

2. 串行算法实现

2.1. 数据的组织

- 数据的输入和输出：数据的输入与输出由 parser_*.cc 实现，采取与示例代码相同的方式处理输入文件 *.gr (图输入文件)和 *.ss (源点输入文件)，并打印输出概述信息。

Initialization
Set $d(\texttt{rt}) \leftarrow 0$; for all $v \neq \texttt{rt}$, set $d(v) \leftarrow \infty$.
Set $B_0 \leftarrow \{\texttt{rt}\}$ and $B_\infty \leftarrow V - \{\texttt{rt}\}$.
For $k = 1, 2, \ldots,$ set $B_k \leftarrow \emptyset$.

Δ-Stepping Algorithm
$k \leftarrow 0$.
Loop // Epochs
ProcessBucket(k)
Next bucket index : $k \leftarrow \min\{i > k \ : \ B_i \neq \emptyset\}$.
Terminate the loop, if $k = \infty$.

ProcessBucket(k)
$A \leftarrow B_k$. //active vertices
While $A \neq \emptyset$ //phases
For each $u \in A$ and for each edge $e = \langle u, v\rangle$
Do `Relax`(u, v)
$A' \leftarrow \{x \ : \ d(x)$ changed in the previous step$\}$
$A \leftarrow B_k \cap A'$

`Relax`(u, v)**:**
Old bucket: $i \leftarrow \lfloor \frac{d(v)}{\Delta} \rfloor$.
$d(v) \leftarrow \min\{d(v), d(u) + w(\langle u, v\rangle)\}$.
New bucket : $j \leftarrow \lfloor \frac{d(v)}{\Delta} \rfloor$.
If $j < i$, move v from B_i to B_j.

图 1. Δ-Stepping 算法原理

- 图结构：由于图顶点总数极大，所以图无法通过邻接矩阵存储，因而采用临接表的形式。
- 边结构：每条有向边由目标定点和权重两个元素组成，并按照起始顶点编号有小到大进行组织。

2.2. Dijkstra 算法

Dijkstra 算法采用优先队列优化的形式，即每次寻找当前路径最短的顶点的时间复杂度为 $\mathbf{O(1)}$，每次更新加入的顶点的时间不超过 $\mathbf{O(log\ n)}$。

2.3. Δ-Stepping 算法

Δ-Stepping 算法根据其流程可以分为三个主要部分：ProcessBucket，FindRequest 和 Relax。其中，FindRequest 负责根据当

前桶内的元素查找所有满足处理条件的 (u, v)，Relax 负责更新当前距离以及元素在桶间的移动。

3. 并行算法实现

并行算法采用 OpenMP 实现，并在服务器上进行测试，以保证其效率和准确性。

3.1. Dijkstra 算法

由于 Dijkstra 算法在每次选择顶点时有顺序依赖，所以该操作不能并行。此外，如果使用优先队列进行优化，每次 Relax 操作都可能需要对队列进行更新，所以结点加入优先队列的操作应当是互斥的。因此算法的并行性很低。

3.2. Δ-Stepping 算法

Δ-Stepping 算法的 ProcessBucket 部分是有顺序依赖的，而 FindRequest 和 Relax 操作在每轮桶操作中是独立的，结点之间没有依赖。因此，这两部分可以并行处理。

4. 实验分析

本文实现了上述两种单源最短路径算法的串并行形式，并对二者的加速比进行比较。由表 **1** 可知，由于 Δ-Stepping 算法包含了对 Dijkstra 算法的贪心策略进行松弛，故加速比更高，算法并行性更好。

针对 Dijkstra 算法，本文衡量了优先队列优化对算法效率的提升效果。表 **2** 展示了使用优先队列的 Dijkstra 算法与朴素 Dijkstra 算法的加速比。

表 1. 两类算法的加速比

数据集	NY	NE	CTR
Dijkstra	0.90	0.88	0.90
Δ-Stepping	2.18	2.02	2.11

表 2. 优先队列优化效果

数据集	NY	NE	CTR
非优化时间/优化时间	210.95	1566.33	—

对于 Δ-Stepping 算法，本文展示了不同 Δ 和并行线程数对于运行速度的影响。如图 **2** 所示，当并行数较低时，额外操作开销比重较大，算法速度较慢，虽并行程度增加，算法速率提高。图 3 展示了不同 Δ 对并行算法效率的影响，其中 Δ 的选择在图中反映为桶的数目 $(b \propto 1/\Delta)$。Δ 越大，桶数越少，算法越接近 Bellman-Ford 算法，并行性越高。

图 2. 运行时间与线程数关系

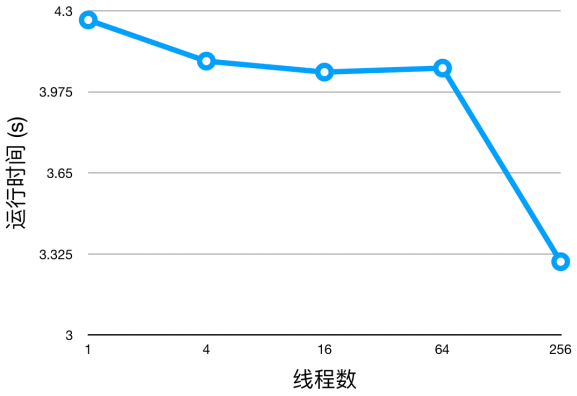
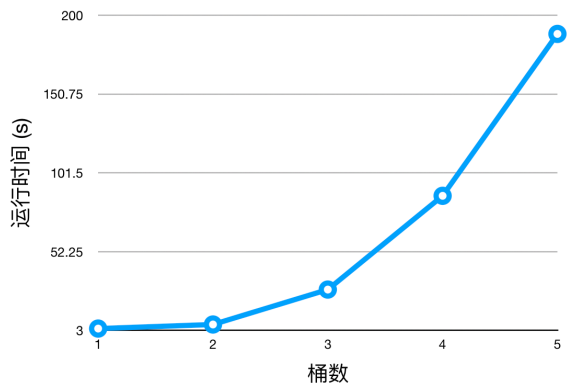


图 3. 运行时间与桶数关系



5. 结论

本文详细介绍并实现了了两种单源最短路径算法以及其并行形式，并对两种算法和串并行算法进行了析和实验验证。此外，针对 Δ-Stepping 进行了参数的比较和实验，进一步掌握了该方法和并行原理。

参考文献

[1] U. Meyer and P. Sanders, "Δ-stepping: A Parallelizable Shortest Path Algorithm," *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, October 2003.