并行计算实验二

PB19030888 张舒恒

问题描述

　输入一个带非负边权的图，一个源节点，求该节点到图中所有节点的最短路径长度。参考现有的并行单源点算法最短路径算法实现。选取OpenMP，MPI，CUDA中的一种实现。

实验文件结构

```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/Desktop/并行计算/并行计算Lab2$ tree
.
├── bin
|   ├── a.out              //并行算法可执行文件
|   ├── b.out              //串行算法可执行文件
|   ├── check.exe          //结果验证可执行文件
|   ├── main_a.cpp         //并行算法程序
|   └── main_b.cpp         //串行算法程序
├── data
|   ├── USA-road-d.CTR.gr  //图文件(文件太大未上传)
|   ├── USA-road-d.CTR.ss  //源点文件
|   ├── USA-road-d.NE.gr
|   ├── USA-road-d.NE.ss
|   ├── USA-road-d.NY.gr
|   ├── USA-road-d.NY.ss
|   ├── USA-road-t.CTR.gr
|   ├── USA-road-t.CTR.ss
|   ├── USA-road-t.NE.gr
|   ├── USA-road-t.NE.ss
|   ├── USA-road-t.NY.gr
|   ├── USA-road-t.NY.ss
|   ├── genSources.pl        //生成源点文件的附属脚本
|   └── genUSA-road-d.ss.pl  //生成源点文件的脚本
├── result                 //实验输出
|   ├── USA-road-d.CTR.res
|   ├── USA-road-d.NE.res
|   ├── USA-road-d.NY.res
|   ├── USA-road-t.CTR.res
|   ├── USA-road-t.NE.res
|   └── USA-road-t.NY.res
└── 并行计算实验二.pdf        //实验报告
```

算法设计

串行算法——Dijkstra

　Dijkstra算法[1]是由荷兰计算机科学家Dijkstra于1959 年提出的，解决的是有权图中最短路径问题。Dijkstra算法主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。

　Dijkstra的原始算法的时间复杂度为$O(|V|^2)$，基于Fibonacci堆实现优先队列的优化算法可以达到$O(|E|+|V|log|V|)$。对于具有无界非负权重的任意有向图，这是趋近于最快的已知的单源最短路径算法。

算法伪代码如下:

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

并行算法——Δ-Stepping

Δ-Stepping算法[2]由U. Meyer和P. Sanders在2003年的论文里提出。算法结合了Dijkstra和Bellman-Ford算法的思想，引入了"桶"和Δ的概念，把边与Δ相比分为heavy和light两类，把所有顶点按到源点距离大小放入不同的桶里。该算法从第一个非空桶中移除所有节点，并从这些节点中松弛所有light边，这可能导致进入当前桶的新节点在下一阶段被删除。此外，如果先前阶段已经改善了它们的距离，则可以重新插入从该桶中删除的节点。此时不需要松弛heavy边，因为它们不会将节点插入当前桶中。一旦当前桶在一个阶段后最终保持为空，则已为其内的所有节点分配了它们的最终距离值。

算法过程主要是按照当前距源点距离将所有点分入桶中，每次迭代从第一个非空桶开始，对桶中每个节点的边权重小于桶长的邻节点做松弛，新松弛出的节点若仍有落在该桶中的则继续对该桶进行上述操作，期间记录所有松弛过的节点，否则对刚才松弛过的节点的边权重大于桶长的邻节点做松弛，之后进入下一次迭代，所有桶均空算法结束。

可并行部分是对所有点分入桶中的操作以及对桶中每个节点的边权重小于桶长的邻节点做松弛的操作。

这种算法的精妙之处在于边权重大于桶长的节点松弛过后一定不会落在当前桶，所以具有良好的**可并行性。**

Δ-Stepping算法能达到$O(n + m + d \cdot L)$的时间复杂度(n、m是顶点数和边数，d是最大度数，L是源点到其他点最短路径中的最大值)，十分高效，同时又允许实现有效的并行化。

算法伪代码如下:

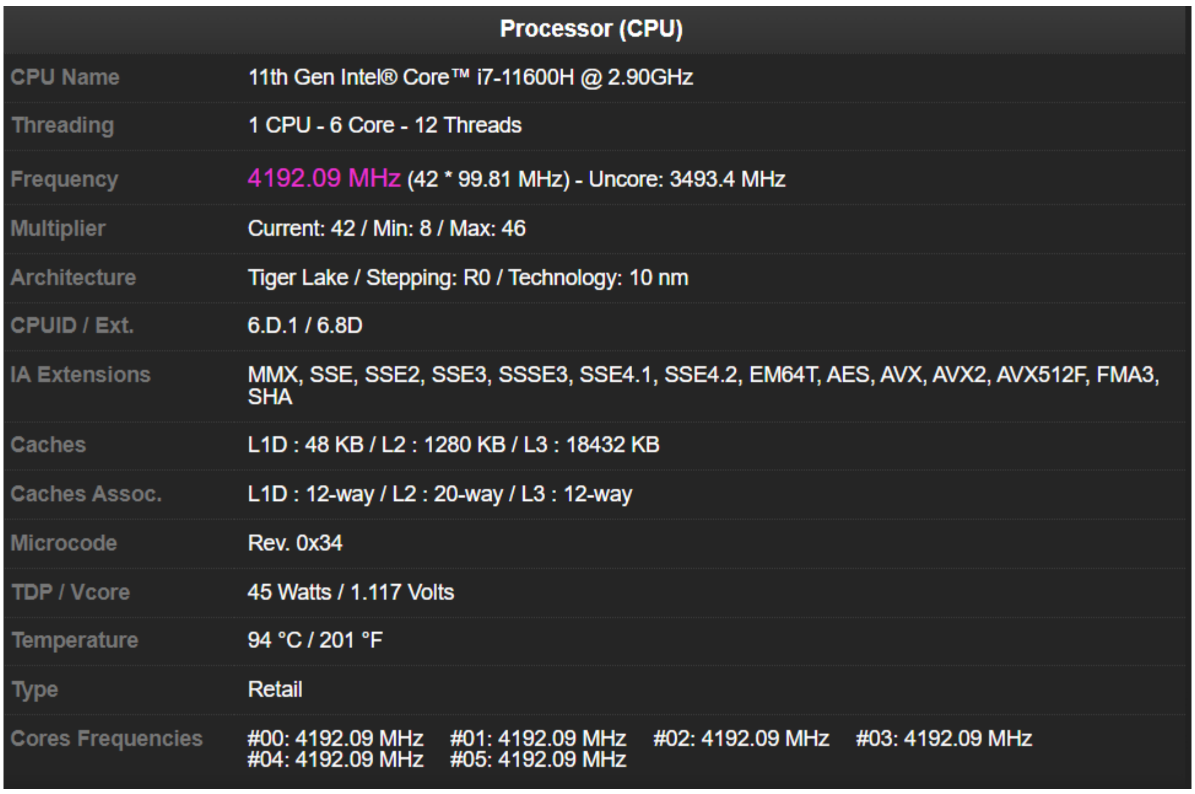
```
foreach $v \in V$ do tent($v$) := $\infty$
relax($s$, 0);                                          (* Insert source node with distance 0 *)
while $\neg$isEmpty(B) do                               (* A phase: Some queued nodes left (a) *)
    $i := \min\{j \geqslant 0\colon\ B[j] \neq \emptyset\}$          (* Smallest nonempty bucket (b) *)
    $R := \emptyset$                                    (* No nodes deleted for bucket $B[i]$ yet *)
    while $B[i] \neq \emptyset$ do                      (* New phase (c) *)
        Req := findRequests($B[i]$, light)              (* Create requests for light edges (d) *)
        $R := R \cup B[i]$                              (* Remember deleted nodes (e) *)
        $B[i] := \emptyset$                             (* Current bucket empty *)
        relaxRequests(Req)                              (* Do relaxations, nodes may (re)enter $B[i]$ (f) *)
    Req := findRequests($R$, heavy)                     (* Create requests for heavy edges (g) *)
    relaxRequests(Req)                                  (* Relaxations will not refill $B[i]$ (h) *)

Function findRequests($V'$, kind : {light, heavy}) : set of Request
    return $\{(w, \text{tent}(v) + c(v, w))\colon\ v \in V' \wedge (v, w) \in E_{\text{kind}}\}$

Procedure relaxRequests(Req)
    foreach $(w, x) \in$ Req do relax($w$, $x$)

Procedure relax($w$, $x$)                               (* Insert or move $w$ in $B$ if $x <$ tent($w$) *)
    if $x <$ tent($w$) then
        $B[\lfloor \text{tent}(w)/\Delta \rfloor] := B[\lfloor \text{tent}(w)/\Delta \rfloor] \setminus \{w\}$     (* If in, remove from old bucket *)
        $B[\lfloor x/\Delta \rfloor] := B[\lfloor x/\Delta \rfloor] \cup \{w\}$                  (* Insert into new bucket *)
        tent($w$) := $x$
```

实验评测

实验配置

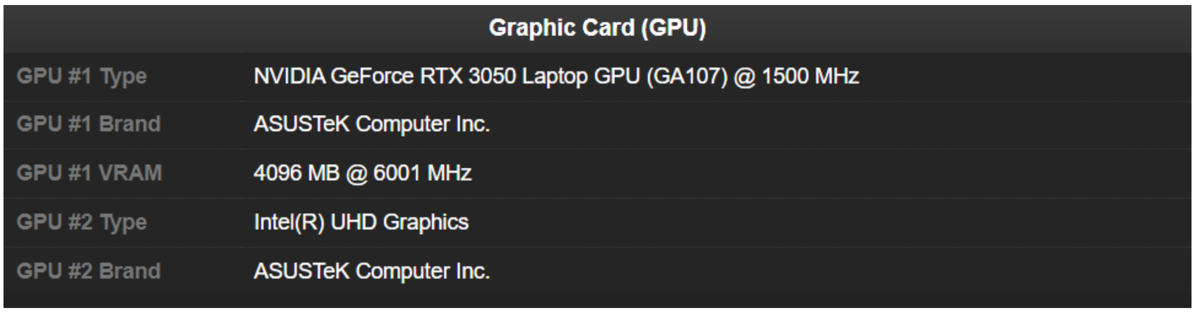
软硬件配置

CPU参数：



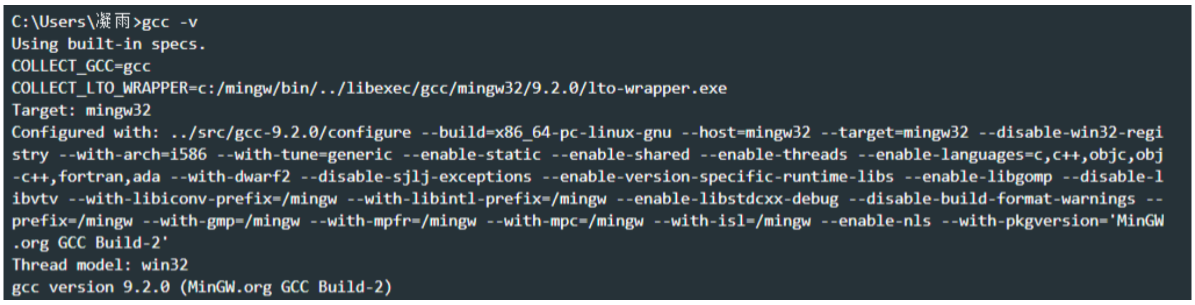
Processor (CPU)	
CPU Name	11th Gen Intel® Core™ i7-11600H @ 2.90GHz
Threading	1 CPU - 6 Core - 12 Threads
Frequency	4192.09 MHz (42 * 99.81 MHz) - Uncore: 3493.4 MHz
Multiplier	Current: 42 / Min: 8 / Max: 46
Architecture	Tiger Lake / Stepping: R0 / Technology: 10 nm
CPUID / Ext.	6.D.1 / 6.8D
IA Extensions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, AVX512F, FMA3, SHA
Caches	L1D : 48 KB / L2 : 1280 KB / L3 : 18432 KB
Caches Assoc.	L1D : 12-way / L2 : 20-way / L3 : 12-way
Microcode	Rev. 0x34
TDP / Vcore	45 Watts / 1.117 Volts
Temperature	94 °C / 201 °F
Type	Retail
Cores Frequencies	#00: 4192.09 MHz #01: 4192.09 MHz #02: 4192.09 MHz #03: 4192.09 MHz #04: 4192.09 MHz #05: 4192.09 MHz

GPU参数：



Graphic Card (GPU)	
GPU #1 Type	NVIDIA GeForce RTX 3050 Laptop GPU (GA107) @ 1500 MHz
GPU #1 Brand	ASUSTeK Computer Inc.
GPU #1 VRAM	4096 MB @ 6001 MHz
GPU #2 Type	Intel(R) UHD Graphics
GPU #2 Brand	ASUSTeK Computer Inc.

编译器参数：



```
C:\Users\凝雨>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/9.2.0/lto-wrapper.exe
Target: mingw32
Configured with: ../src/gcc-9.2.0/configure --build=x86_64-pc-linux-gnu --host=mingw32 --target=mingw32 --disable-win32-regi
stry --with-arch=i586 --with-tune=generic --enable-static --enable-shared --enable-threads --enable-languages=c,c++,objc,obj
-c++,fortran,ada --with-dwarf2 --disable-sjlj-exceptions --enable-version-specific-runtime-libs --enable-libgomp --disable-l
ibvtv --with-libiconv-prefix=/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --disable-build-format-warnings --
prefix=/mingw --with-gmp=/mingw --with-mpfr=/mingw --with-mpc=/mingw --with-isl=/mingw --enable-nls --with-pkgversion='MinGW
.org GCC Build-2'
Thread model: win32
gcc version 9.2.0 (MinGW.org GCC Build-2)
```

数据集配置

实验数据来自罗格斯大学(Rutgers University)离散数学和理论计算机科学中心(The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS))于2006年发起的一项单源最短路径问题编程竞赛数据集，http://users.diag.uniroma1.it/challenge9/format.shtml，它以美国各个地区在真实地图中的位置，距离或时间作为数据源，这也是国际公认较为权威的数据源。取其中的六个数据集：**CTR-d**，**CTR-t**，**NE-d**，**NE-t**，**NY-d**，**NY-t**，数据规模如下:

Name	Description	Nodes	Arcs	Longitude	Latitude	Graph file size
NY-d	New York City,Distance	264,346	733,846	[40.3; 41.3]	[73.5; 74.5]	3.5MB
NY-t	New York City,Travel time	264,346	733,846	[40.3; 41.3]	[73.5; 74.5]	2.6MB
NE-d	Northeast USA,Distance	1,524,453	3,897,636	[39.5, 43.0]	[-infty; 76.0]	21MB
NE-t	Northeast USA,Travel time	1,524,453	3,897,636	[39.5, 43.0]	[-infty; 76.0]	21MB
CTR-d	Central USA,Distance	14,081,816	34,292,496	[25.0; 50.0]	[79.0; 100.0]	195MB
CTR-t	Central USA,Travel time	14,081,816	34,292,496	[25.0; 50.0]	[79.0; 100.0]	198MB

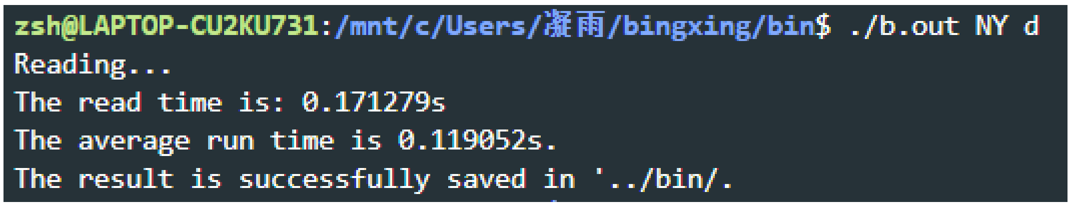
实验结果

正确性验证

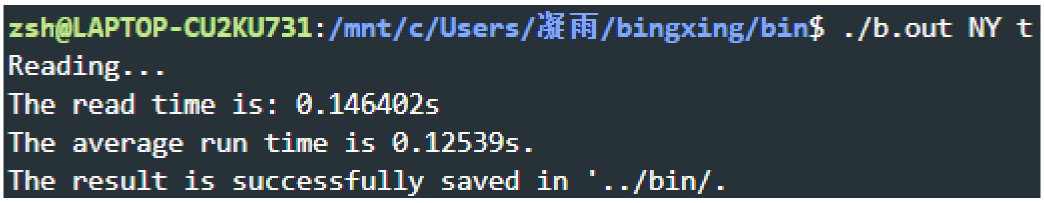
利用该竞赛所给的Correctness checking files (.ss.chk files)所生成的check.exe(见bin文件夹下)即可验证程序计算结果的正确性

加速比分析

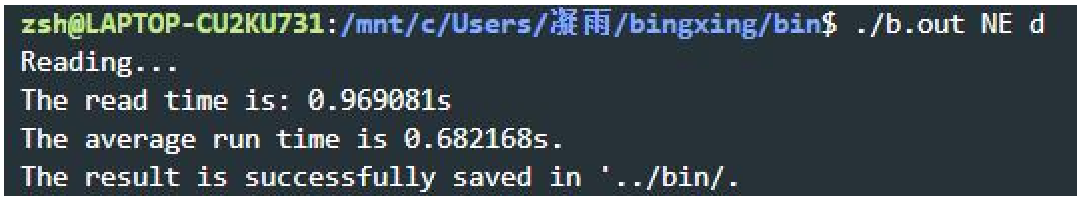
串行算法运行结果：



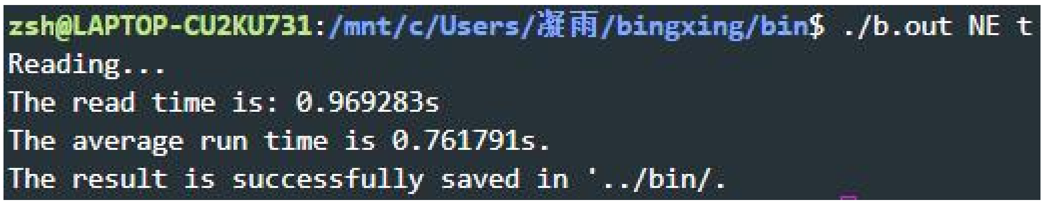
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./b.out NY d
Reading...
The read time is: 0.171279s
The average run time is 0.119052s.
The result is successfully saved in '../bin/.
```



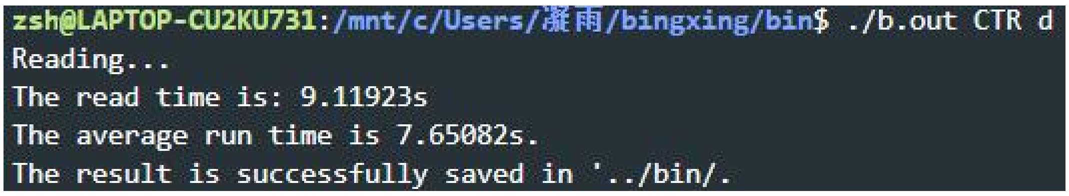
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./b.out NY t
Reading...
The read time is: 0.146402s
The average run time is 0.12539s.
The result is successfully saved in '../bin/.
```



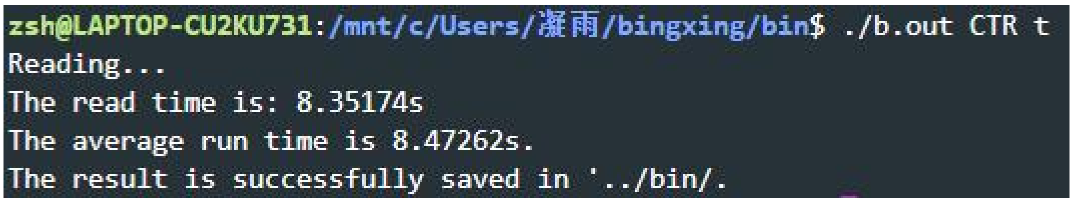
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./b.out NE d
Reading...
The read time is: 0.969081s
The average run time is 0.682168s.
The result is successfully saved in '../bin/.
```



```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./b.out NE t
Reading...
The read time is: 0.969283s
The average run time is 0.761791s.
The result is successfully saved in '../bin/.
```

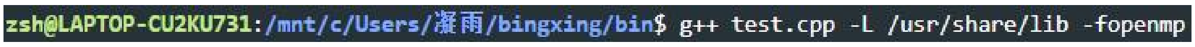


```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./b.out CTR d
Reading...
The read time is: 9.11923s
The average run time is 7.65082s.
The result is successfully saved in '../bin/.
```

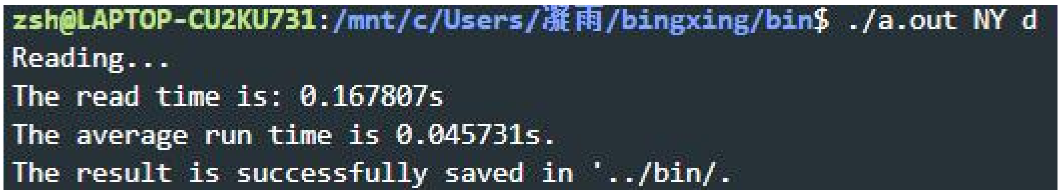


```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./b.out CTR t
Reading...
The read time is: 8.35174s
The average run time is 8.47262s.
The result is successfully saved in '../bin/.
```

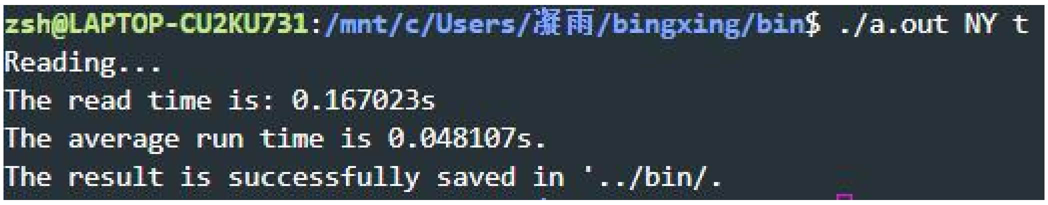
并行算法运行结果：



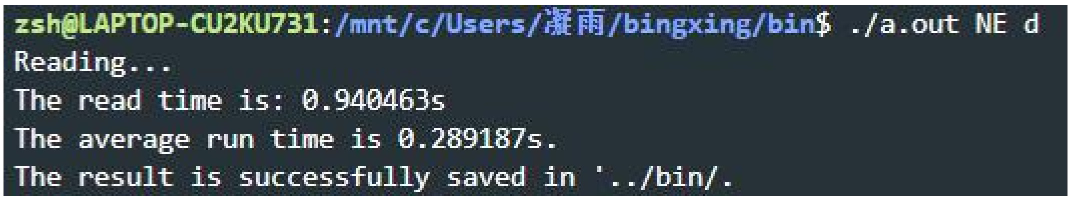
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ g++ test.cpp -L /usr/share/lib -fopenmp
```



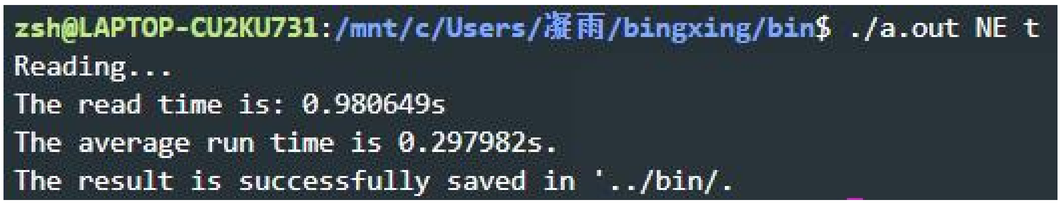
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./a.out NY d
Reading...
The read time is: 0.167807s
The average run time is 0.045731s.
The result is successfully saved in '../bin/.
```



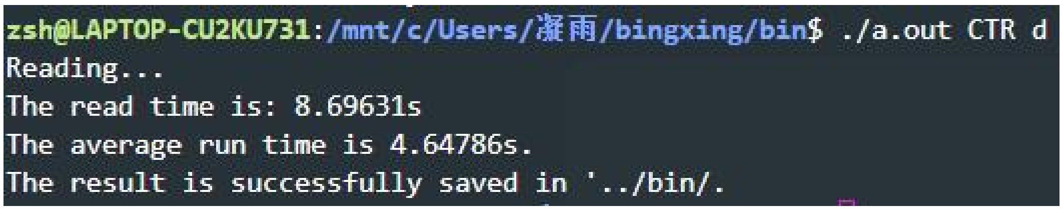
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./a.out NY t
Reading...
The read time is: 0.167023s
The average run time is 0.048107s.
The result is successfully saved in '../bin/.
```



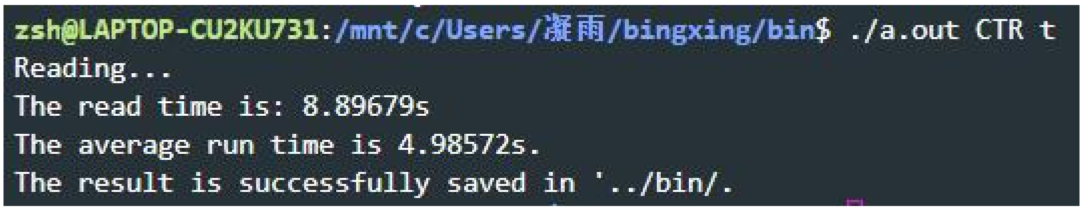
```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./a.out NE d
Reading...
The read time is: 0.940463s
The average run time is 0.289187s.
The result is successfully saved in '../bin/.
```



```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./a.out NE t
Reading...
The read time is: 0.980649s
The average run time is 0.297982s.
The result is successfully saved in '../bin/.
```



```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./a.out CTR d
Reading...
The read time is: 8.69631s
The average run time is 4.64786s.
The result is successfully saved in '../bin/.
```



```
zsh@LAPTOP-CU2KU731:/mnt/c/Users/凝雨/bingxing/bin$ ./a.out CTR t
Reading...
The read time is: 8.89679s
The average run time is 4.98572s.
The result is successfully saved in '../bin/.
```

加速比：

数据集	串行算法时间	并行算法时间	加速比
NY-d	0.119052s	0.045731s	2.603
NY-t	0.12539s	0.048107s	2.606
NE-d	0.682168s	0.289187s	2.359
NE-t	0.761791s	0.297982s	2.557
CTR-d	7.65082s	4.64786s	1.646
CTR-t	8.47262s	4.98572s	1.699

可以看出随着问题规模的增大加速比逐渐下降，这是因为问题规模的增大导致最佳的Δ难以找到从而降低了算法效率。

参考文献

[1]Dijkstra's AlgorithmDijkstra's Algorithm (programiz.com)

[2]Δ-stepping: a parallelizable shortest path algorithm.U.Meyer.P.Sanders(https://www.sciencedirect.com/science/article/pii/S0196677403000762)