An Improved Dijkstra's algorithm application to multi-core processors

Qiong Wu [1, 2], Guihe Qin[1], Hongliang Li[2]

1 College of Computer Science and Technology, Jilin University, Changchun, 130122,China
2 College of Humanities & Information of Changchun University of Technology, Changchun, 130122,China

Abstract
Dijkstra's algorithm is a classic algorithm of shortest path, widely used in the field of GIS. In order to improve the efficiency of computing, the work proposes an improved of parallelize Dijkstra's algorithm applied to multi-core environment. Through the program parallel computing on multi-core at the same time, the method greatly improve the computational efficiency. Through experimental tests, it shows that efficiency of improved of parallelize Dijkstra's algorithm can increase about 50% when it compared with traditional Dijkstra's algorithm. While multi-core environment has become the mainstream of the embedded platform development, Improved of parallelize Dijkstra's algorithm has great value.
Keywords: DIJKSTRA'S ALGORITHM; PARALLELIZATION; MULTI-CORE; THE SHORTEST PATH

1. Introduction

The shortest path analysis of the shortest path planning algorithm is the key of space network analysis in the GIS (Geographic Information System) [1]. In recent years, with the development of the vehicle mounted and road construction, the navigation system become one of the popular application of path planning algorithm. Current path planning algorithm is mainly derived from three theoretical branches [3]: the first is based on graph theory, the second is based on artificial intelligence theory, and the third is based on the theory of intelligent control.

Dijkstra's algorithm is a classic representative of graph theory and proposed by the Dutch scientist Dijkstra in 1959. It is a classical algorithm of the shortest path. The algorithm abstracts the complex geographic information for the network graph. The path for the network graph with weights of edges, vertices in the graph represents the location, usually used to calculate the shortest path between two vertices in the directed graph. Dijkstra's algorithm can get the optimal solution of the shortest path, but because of the need to traverse a large number of vertices, so the efficiency is low, the time complexity is O (n2), where n represents the number of vertices. Because the scale of road network is often large, massive computing become the bottleneck of the algorithm [2].In recent years some improved Dijkstra's algorithms were put forward in the literature [6-11], in the calculation method and optimized storage, etc. But in the onboard systems and handheld devices such as embedded system, the hardware condition of low computing power become the main obstacle of the development of the algorithm.

In recent years, the development and popularization of multi-core technology has brought changes in two aspects to the software industry: on the one hand, multiple CPUs greatly improve the computational efficiency; On the other hand, the change of the hardware brings new challenges to the software design idea and design method. Multi-core processor is that two or more than two core was integrated in a single

chip, and a computer can contain multiple multi-core processors. According to the position and role of the cores, it divided into homogeneous and heterogeneous. The emergence of the multi-core structure has dominated the area of parallel computing [5].Therefore, Dijkstra's algorithm limited by a large amount of calculation has a new development space.

This article is divided into the following several parts: section 2 introduces the current research quo of the Dijkstra's algorithm, section 3 proposed the improved of parallelized Dijkstra's algorithm in multi-core environment, the fourth quarter comparing the improved algorithm and classical algorithm through the experimental data, section 5 to summarize work and put forward the direction of next step.

2. Related Work

Classical Dijkstra's algorithm has proposed different improvements for different work requirements by many scholars in practice. In [6], ASAP algorithms have been proposed based on hierarchical processing. While APSP algorithm is based on matrix multiplication to achieve a fast optimization in [7].A Double-Sources shortest path algorithm for urban road networks is proposed in [8]. This algorithm starts at searching for the shortest path from the source station and destination station respectively and simultaneously. It can reduce the time-complexity based on Dijkstra's algorithm and the characteristics of the typical urban road network. MIFDA Algorithm was proposed in [9] for solving Intuitionistic Fuzzy Shortest Path Problem using the Intuitionistic Fuzzy Hybrid Geometric operator. It achieves tractability through tolerance for imprecision, uncertainty and partial truth. In [10], authors presented an efficient time-dependent route planning algorithm. It is able to efficiently compute exact shortest paths in time-dependent continental-sized transportation networks. Dijkstra's algorithm was discussed from the viewpoint of discrete convex analysis in [11].

The emergence of multi-core platform soon shows technical advantage. Bring more powerful computing performance for users; More important, because it can invoke multiple threads at the same time work together. The traditional serial task on multi-core processor speed will increase greatly, but the ability to parallel processing tasks can give full play to the advantages of multi-core. But at present, there are problems in the work of transform serial task into parallel processing tasks. An improved of parallelize Dijkstra's algorithm was proposed, experiments show that the Parallelized algorithm in terms of speed of increase even more than the number of CPU advantage.

Dijkstra's shortest path algorithm is described below.

Node a is defined as the source node, the starting point of the vertices, L (i, j) is the distance between nodes i and j, distance of source node a to node v is defined as D (v), it is sum of all the length of the link along a certain path from node a to node v (plus the weight). Algorithm is divided into two parts: initialization and computing.

A. Initialization part. C is collection of network nodes. C = {a}.For all node v not in C, the method of calculation D (v) as follows: if the node v and the source directly connected, D (v) = L (a, v), otherwise D (v) =∞.

B. Computation part. Looking for a node w not in C, D (w) is Min. Add w to C, and for node v is not in the C, update the original D (v) value, namely: D (v) = Min [D (v), D (w) + L (w, v)]. Repeat looking for the next node, until all the network nodes are in C. If computing the minimum distance of source a to node t, when t into the set C, end of the algorithm, the D (t) is the shortest distance between them. Figure 1 is a network with five nodes, the calculation process of shortest distance from source node a to node e as shown in Table 1.

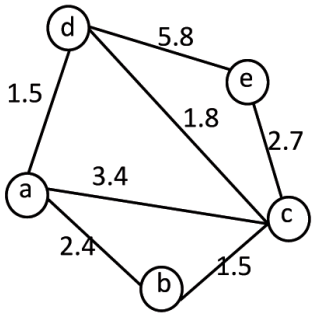

Figure 1. The Network With 5 Nodes

Talbe 1. The Calculation Process of Shortest Distance Between Node a and Node e Algorithm

Step	Path	D(b)	D(c)	D(d)	D(e)
initialization	{a}	2.4	3.4	1.5	∞
1	{a,d}	2.4	3.3	1.5	7.3
2	{a,d,b}	2.4	3.3	1.5	7.3
3	{a,d,b,c}	2.4	3.3	1.5	6.0
4	{a,d,b,c,e}	Push-back path: e,c,d,a			

3.The Proposed Improved of Parallelize Dijkstra's Algorithm

3.1. The Description of Algorithm

This section description the improved of parallelize algorithm, it is shown in figure 2. Initialization algorithm is the first operate step, the second step is to calculate AC to parallelize the Dijkstra's algorithm, after the third back to the shortest path. Here variable AC represents a criterion of the number of threads.

In the process of the Dijkstra's algorithm parallelization, set for the number of parallel threads is more important. Sets the number of split parallel threads to N, number of nodes in network for K. Due to parallelized programming involves the split of threads, synchronization, merge etc, need to take up a certain

Algorithm: improved Parallelized Dijkstra's algorithm

Assumption

Multi-cores environment

Input

M //Number of cores
N // The actual number of parallel threads
K //Number of nodes in the network
a //Source node
e //End node

Begin

Step 1: Initialization

C is collection of network nodes. C = {a}.For all node v not in C, initialization D (v) as Eq.(1)

$$D(v)=\begin{cases} L(a,v) & \text{if the node v and the source directly connected} \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

Step 2: Calculation

Calculation AC。
If AC<1 then
N=1
Else
if AC>2M then N=2M Endif // calculation AC
Else N=AC
EndIf

Looking for a node w not in C, D (w) is minimum and w∉C .

While(w≠e)do
Parallelized program, split into N threads simultaneously calculated.
update D (v), namely: D (v) = Min [D (v), D (w) + L (w, v)]
IF D(v) is changed, record prior node w
EndIF
until N threads all end.

Looking for a node w not in C, D (w) is minimum and w∉C .

Endwhile

Step 3: Push-back path

push-back prior node of each node From e. Get a shortest path.

End

Figure 2. A Improved of Parallelized Dijkstra's Algorithm

amount of memory and time, time consumption of realize the parallelization more than time saved by parallel computing when the number of node is smaller, so parallelized algorithm slower than the traditional serial algorithm. Figure 3 described the comparison of the traditional algorithm and Parallelize Dijkstra's algorithm in the Windows environment with 2cores 4 threads.

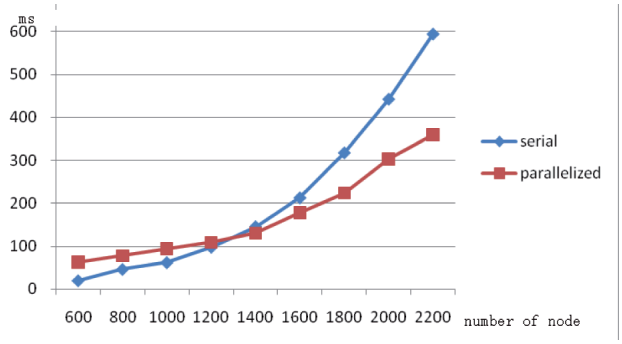

Figure 3. Compared of Runtime

Where N = 4, 200≤K≤2600. It can be seen from the figure 2, when K > 1300, the superiority of Parallelized algorithms to show up. However different operating system platforms, different CPU and different network model will have different influence on the calculate speed of algorithm, therefore, here introduce a criterion of the number of threads variable AC. When AC>1, Parallelized Dijkstra's algorithm is called; AC≤1, serial algorithm is called. So it is a self-adapt algorithm in figure 2.

3.2. AC Parameters Discussed

The split of thread requires a certain resource consumption, including time and space. AC associates with three factors by research: K (Total number node), M(number of cores) and frequency of CPU. When K is small, using serial algorithm, and when K is big, it is more rational for using of the parallel algorithm because of a large amount of calculation. At the same time, the faster cpu is, the stronger computing power is, it can process more calculations in criteria time. In turn, it only can process less calculations.

$$AC = \left(K / 1000\right)^2 / \left(M * \left(1 - \alpha\right)\right) \qquad (2)$$

It shows the relationship between the AC , the nodes number K, and cores number M in Eq.2. M is inversely proportional to AC, and K is proportional,α as a factor (α∈[0,1]), α is smaller when the cpu faster, while α is greater when cpu speed is slower. The data of runtime show in figure 4 and Table 2. The experiments show it is more reasonable when α= 0.8. At this point, the critical value of conversion algorithm is close to serial algorithm's best number of node.

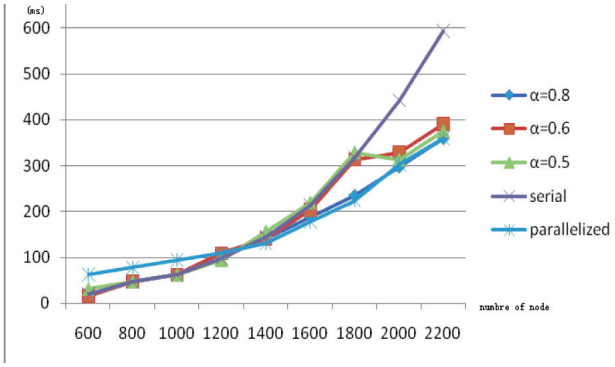

Figure 4. Runtime of Diffrent Parameter α

It indicating the number of nodes fewer or cores more, using the serial algorithm for calculation is more reasonable When AC≤1. When the maximum number of threads AC>1, the number of threads allocated according to calculations.But the number of threads supplied with the number of system thread when number of calculation over the system hardware.

Table 2. Runtime of Improved of Paralized Algorithm with Different α

Run Time Node Number	α=0.8	α=0.6	α=0.5	serial	parallelized
600	16	16	31	21	63
800	47	47	47	47	78
1000	62	62	62	63	94
1200	110	109	94	98	109
1400	140	141	156	146	130
1600	188	203	219	213	177
1800	234	313	328	318	224
2000	297	328	313	443	302
2200	359	390	375	594	359

4. Experimental Study

This section discusses the methodology used in experimental design and observes the experimental results.

4.1. Experiment Platform

The algorithm was experimented on Windows and Linux operating system.In the experiments, the number of K is from 200 to 9800, the node of edge between the initial value generated by the random function in the [-100,100], with a K * K matrix storage, all zero and negative on behalf of the ∞.The number of positive value of M line of the matrix on behalf of node M has number of edges. Records of prior nodes using of a one-dimensional array (length of K) to store. Figure 5 is shown the prior node of the every node in figure 1. Nodes in the array respectively represent the prior node of the node on the array.

Node	a	b	c	d	e
Array	#	a	d	a	c

Figure 5. Storing of Prior Node

4.2. OpenMP Tools

OpenMP is a Shared memory parallel system of multithreaded programming technology [4].OpenMP standard was formed in 1997, used to write portable multithreaded applications. This model provides a set of platform-independent compiler directive commands and environment variables, function calls, instruction, to direct the compiler explicitly when using the application of parallelism and how to do it. OpenMP provides a high-level abstract description of parallel algorithm, an OpenMP program started from a single thread, need parallel program is derived at some point in the program (Fork) of some thread group,the parallel code executed in parallel area, and connected together (Join)in different threads at the end of the parallel region when all threads in the thread group is ends. OpenMP is Fork - Join model work. In the above experiments, parallel work performed by OpenMP tools.

4.3. The result of the experiment

In order to better explain the experimental results, we compare algorithm in two aspects: on the one hand, under the same conditions, compared the total completion time of serial algorithm and improved of parallelize algorithm comparison; On the other hand, in the case of different cpus, compared the total completion time of two algorithms. By the experiments, the total completed time on the basis of the comparison research, the experimental data is shown in figure 6. İmproved of parallelize algorithm in speed has the obvious improvement, speed increased about by 50%.

Figure 7 shows the comparison of the completion time of improved of parallelize algorithm and serial

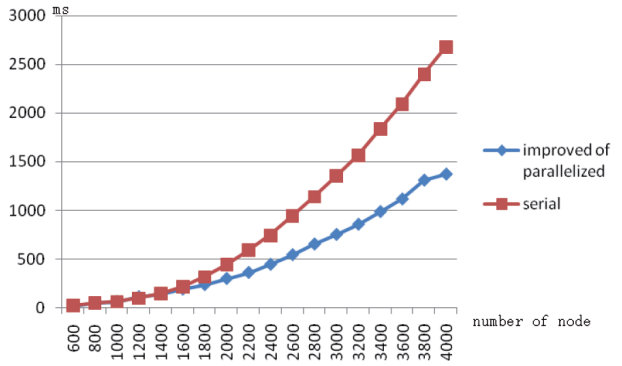

Figure 6. Compaired of Runtime of Two algorithm

algorithm under the condition of 2 cores 4 threads and 4 cores 8 threads. It can see from the figure 7, a serial algorithm has the most complete time under the condition of 2 cores 4 threads, and the improved algorithm has the least finish time under the condition of 4 cores 8 threads, it is in accordance with the above rule.

Then, comparison of serial algorithm under 4 cores 8 threads and improved algorithm under 2 cores 4 threads will found that serial algorithm has less completed time when the number of node K <7000. Because of the advantages of hardware, serial algorithm faster than improved of parallelize algorithm. But when the node number K continues to increase, the advantage of improved algorithms become obvious, its speed can even faster than 4 cores serial algorithm.

Conclusions

Traditional algorithm optimization is under the premise of performance stability of processor, enhance operation efficiency as much as possible by changing the method to calculate or storage means. But that changed with the emergence of multi-core processors. Due to difficult of hardware performance improved, the number of software algorithm optimization should be transferred to the number of CPU. At present, because of the influence of traditional

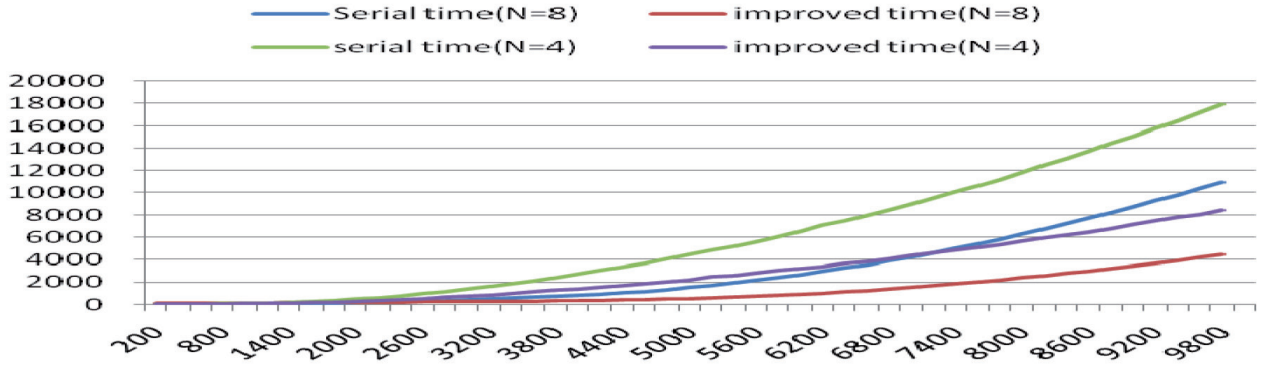

Figure 7. Improved Algorithm Compared with Serial Algorithm on Different Plant

programming ideas, the development of the parallel software far behind the development of the parallel computing architecture. In this paper, the improved of parallelize Dijkstra's algorithm on multi-cores can greatly increase calculation efficiency. it can improve computing speed by 50% when number of nodes is huge. It provides a broad application space for improved ofparallelized Dijkstra's algorithm in the field of GIS with Multi-cores processors applications on embedded system.

Acknowledgements

This work is supported by the Foundation of Jilin Province Education Department (2014645).

References

1. Tangwen wu, Shixiao dong, Zhuda kui, The Caculation of the Sortest Path Using Modified Dijkstra Algorithm in GIS, Journal of Image and Graphics, 2000.12, Vol5(A), No.12, p.p.1019-1023.
2. Liu Zhiyu, Yang Liu, Applying Improved Dijkstra Algorithm to Embedded GIS Syestem, Computer Applications and Software, 2009.12, Vol 26, No. 12. p.p.262-263.
3. Li Qing, Xie Sijiang, TongXinhai, Wang Zhiliang, A Self-adaptive genetic algorithm for the Shortest Path Planning of vehicles and its comparison with Dijkstra and A* algorithms, Journal of University of Science and Technology Beijing, Nov.2006, Vol 28, No.11, p.p.1082-1086.
4. J. M. Bull, J. Enright, X. Guo, C. Maynard and F. Reid, Performance Evaluation of Mixed-Mode OpenMP/MPI Implementations, International Journal of Parallel Programming, 2010, Vol. 38, p.p. 396-417.
5. Neetesh Kumar, Deo Prakash Vidyarthi, Improved scheduler for multi-core many-core systems, Computing, 2014, 96, p.p.1087-1110.
6. Seth Pettie, A new approach to all-pairs shortest paths on realweighted graphs[J], Theoretical Computer Science, Spacial Issue on Papers From I CAL P, 2002, 312(1), p.p.47-74.
7. zwick U, All pairs shortest paths using bridging sets and rectangular matrix multiplication[J]. ACM, 2002, 49, p.p.289-317.
8. Shiming Wang, Jianping Xing, Yong Wu, Yubing Wu, Wei Xu, Xiangzhan Meng, Liang Gao, Double-Sources Dijkstra Algorithm within Typical Urban Road Networks, Advances in Computer Science and Education Advances in Intelligent and Soft Computing, 2012, Vol 140, p.p. 155-161.
9. Sathi Mukherjee, Dijkstra's Algorithm for Solving the Shortest Path Problem on Networks Under Intuitionistic Fuzzy Environment, Journal of Mathematical Modelling and Algorithms, 2012, 12, Vol 11, Issue 4, p.p.345-359.
10. Daniel Delling, Time-Dependent SHARC-Routing, Algorithmica, 2011,5, Vol60, Issue 1, p.p. 60-94.
11. Kazuo Murota, Akiyoshi Shioura, Dijkstra's algorithm and L-concave function maximization, Mathematical Programming, 2014, 6, Vol 145, Issue 1-2, p.p.163-177.