MINISTRY OF EDUCATION AND TRAINING

DA NANG UNIVERSITY

----- ☙🕮❧ -----

NGUYEN DINH LAU

PARALLELIZATION OF ALGORITHMS IN GRAPH NETWORK

SPECIALIZATION: COMPUTER SCIENCE

CODE: 62.48.01.01

PHD. THESIS IN BRIEF

DA NANG - 2016

This thesis has been finished at:

Danang University of Technology, Da Nang University

Supervisor:

1. Ass. Prof. DrSc. TRAN QUOC CHIEN
2. Ass. Prof. PhD. LE MANH THANH

Examiner 1: Ass. Prof. PhD. Doan Van Ban

Examiner 2: Ass. Prof. PhD. Nguyen Mau Han

Examiner 3: PhD. Huynh Huu Hung

The PhD. Thesis was defended at the Thesis Assessment Committee at Da Nang University Level at Room No:

Da Nang University

At 8.30 am January 24th 2016

The thesis is available at:

1. The National Library.
2. The Information Resources Center, University of Danang.

PREFACE

1. Significance of the study

When building sequential algorithms for problems on the graphic network, the algorithms themselves are not only very complex but the complexity of the algorithms also is very big. Thus, sequential algorithms must be parralled to share work and reduce computation time.

For above reasons, it is crucial to build parallelization of algorithms in extended graph to find the shortest path and to find maximum flow. Therefore, a study of "**Parallelization of algorithms in graph network**" is essential to deal with many real problems with huge input data in our daily life

2. Objects and Scope of the study

- *Objects of the study*

- Investigating and discovering parallel processing theory, parallel processing models.

- Investigating graphic theory, mainly focus on parallel algorithms to find the shortest and to find the maximum flow.

- *Scope of the study*

- Suggesting parallel algorithms to find the shortest path for the two vertexes in extended graph.

- Suggesting parallel algorithms finding the maximum flow by Preflow-push methods, Suggesting parallel algorithms to find the maximum flow by Postflow-pull methods, Suggesting parallel algorithms to find concurent max flow with bounded cost in extended graph.

3. New findings of the study

- Suggesting arallel algorithms to find the shortest path in extended network. So far, parallel algorithms have been built in

traditional graphs. In this thesis, the first new parallel algorithms are especially built for real transport network, prove complexity.

- Suggesting parallel algorithms to find maximum flow by Preflow-push methods. A new feature here is the sequential and parallel algorithms are carefully demonstrated, the algorithms also consider several unexpected cases to occur in the process. Experimental section is made clear. Besides, the program is built on a cluster computer system. Finally, the results are skillfully compared and fully commented.

- Suggesting parallel algorithms to find the maximum flow by combined flow push and pull methods. New feature here is that the new algorithm built has not been announced, the theorems are proven, the experimental examples were built specifically.

- Suggesting new parallel algorithms finding the concurent max flow with bounded cost in extended graph. New feature here is that the new algorithm buil thas not been announced, the theorems are proven.

4. Results of the study

-Suggesting a new parallel algorithms based on the actual requirements, proving soundness, the complexity of the algorithms. Concurently, thesis also does parallelization for existing algorithms, then indicate the advantages of the new ones over previous algorithms.

- Developing experimental programs on various parallel systems, then offering specific data to evaluate and compare the results of new parallel algorithms with sequential algorithms or with the other previous parallel algorithms.

5. Table of contents

Besides preface, conclusion, references, the thesis has four chapters:

Chapter 1. Parallel processing.

Chapter 2. Sequential and parallel algorithms in tradition graph network

Chapter 3. parallel algorithms to find the shortest path and maximum flow in extended graph network.

CHAPTER 1
PARALLEL PROCESSING

1.1. Introduction of parallel processing

1.2. Architectures of parallel computers

1.3. Paralell algorithm

1.4. Conclusion

To deal with the problems effectively in our computer, The key here is we have to build parallel algorithms. The common way we do is to convert the sequential algorithms into parallel algorithms, or convert parallel algorithms into other suitable parallel algorithms which are totally equal to the original algorithms.

To evaluate the effectiveness of parallel algorithms, we usually based on the complexity of the algorithms. The complexity of parallel algorithms depends not only on the volume of input data but also on the the architecture of the computer as well as the number of proceesors in the system.

CHAPTER 2. SEQUENTIAL AND PARALLEL ALGORITHMS IN TRADITION GRAPH NETWORK

2.1. Network and flow

2.2. The problem of maximum flow

2.3. Pre-fow push algorithm to find maximum flow

2.3.1. Sequential algorithm

2.3.1.1. Introduction

2.3.1.2. Some basic concept

- *Residual network G_f:*

For flow f on $G = (V, E, c)$, where a is source vertex, z is sink vertex, c is capacity. Residual network, denoted G_f is defined as the network with a set of vertices V and a set of edge E_f with the capacity as follows:

- for any edge $(u, v) \in E$, if $f(u, v) > 0$, then $(v, u) \in E_f$ with capacity is $c_f(v, u) = f(u, v)$

- for any edge $(u, v) \in E$, if $c(u, v) - f(u, v) > 0$, then $(u, v) \in E_f$ with capacity is $c_f(u,v) = c(u, v) - f(u,v)$

- *Pre-flow:*

For network $G = (V, E, c)$. Preflow is a set of

flows on the edges $f = \{f_{i,j} \mid (i, j) \in E\}$ So that

(i) $0 \le f_{i,j} \le c_{i,j} \forall (i, j) \in E$

(ii) for any vertex k is not a source or sink, inflow is not smaller than outflow, that is

$$\sum_{(i,k)\in E} f_{i,k} \ge \sum_{(k,j)\in E} f_{k,j}$$

- *Height function:*

Height function of the pre-flow in the network $G = (V, E, c)$ is a set of non-negative vertex weights $h(0)$, ..., $h(|V| -1)$ such that $h(z) = 0$ for z and $h(u) \leq h(v) + 1$ for every edge (u, v) in the residual network for the flow. An eligible edge is an edge (u, v) in the residual network with $h(u) = h(v) + 1$.

2.3.1.3. Preflow-push algorithm

- *Input: G = (V, E, c), a* is source vertex, *z* is sink , capacity

$$c = \{c_{i,j} \mid (i,j) \in E\}$$

- *Output*: Maximum Flow:

$$f = \{f_{i,j} \mid (i,j) \in E\}$$

Step 1:

Initialized: building the initial pre-flow in the edges leaving the source to the sink vertices, which is filled to capacity, the other flows are 0. Choose height function $h(v)$ which is the length of the shortest path from v to vertex z.

Push all unbalanced vertices into the queue Q.

Step 2:

Condition to terminate: If $Q=\varnothing$, then preflow f becomes max flow, end. If $Q\neq\varnothing$, go to Step 3.

Step 3:

How to perform unbalanced vertex: get unbalanced vertex u from the queue.

Browsing the priority edges $(u, v) \in E_f$. Pull along the edge (u, v) a flow with value min{delta, $c_f(u, v)$}, where deltais the excess of the vertex u.

- If vertex v is the new unbalanced vertex, then push this vertex v into queue Q.
- If vertex u is still unbalanced, then increase the height of the vertex u as follows:

$h(u)$: =1 +min$\{h(v) | (u, v) \in E_f\}$

and push u into the queue Q.

Back to step2.

2.3.1.4. For example

2.3.2. Parallel algorithm

2.3.2.1. Introduction

2.3.2.2. The idea of the parallel algorithm

2.3.2.3. Building the parallel algorithm

- *Input*: Graph $G(V, E, c)$ with source a, sink z, the capacity:

$$c = \{c_{i,j} \mid (i, j) \in E\}$$

m processors (P_0, P_1,…, P_{m-1}), where P_0 is the main processor

- *Output*: Maximum flow:

$$f = \{f_{i,j} \mid (i, j) \in E\}$$

Step 1: The main processor P_0 performs

(1.1). initialize: e, h, f, c_f, Q: set of unbalanced vertices (excluding the vertices a and z) are the vertices with positive excess.

(1.2). divide set of vertices V into sub-processors:

Let P_i be the i^{th} sub-processor (i = 1, 2, ..., m-1).

P_i will receive the set of vertices V_i so that:

$$\begin{cases} (V_i \cap V_j) = \emptyset, i \neq j \\ V_1 \cup V_2 \cup \ldots \cup V_{m-1} = V \end{cases}$$

Step 2: sub-processors P_i receive V_i *(i=1,…,m-1)*

Step 3: The main processor checks if $Q=\varnothing$ then end, f becomes maximum flow, else to step 4

Step 4: Main processor send e, h, f, c, c_f to the sub-processors.

Step 5: m-1 sub-processors (P_1, P_2, …,P_{m-1}) implement

(5.1) Receive e, h, f, c, c_f and the set of vertices from the main processor
(5.2) Handling unbalanced vertice v (push and replace label) as in Step 3 of the sequential algorithm. Get unbalanced vertexs v ($e(v)>0$) from Q and $v \in V_i$ (i= 1, 2, ..., m-1). Browsing the priority edges $(u, v) \in G_f$.
Push along edge (u, v) a flow with valuel min{delta, $c_f(u, v)$}, where delta is the excess of vertex u.

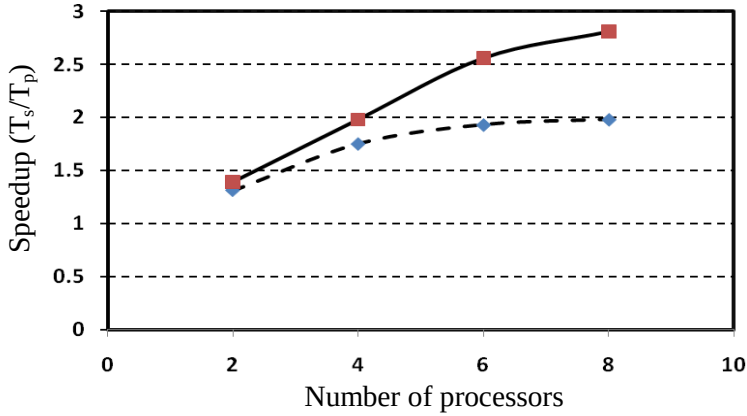
If it does not exist the priority edge fromv, then increased the hight of the vertex u as follows:

$$h(u)= 1 + \min\{h(v) \mid (u, v) \in E_f\}$$

(5.3) Send e, h, f, c, c_f, to the main processor
Step 6: The main processor implements
(6.1) Receive e, h, f, c, c_f from step 5.3
(6.2) This step is distinctive from the sequential algorithms to synchronize our data, after receiving the data in (6.1), the main processor checks if all the edges $(u,v) \in E$ that have $h(u)> h(v)+1$, the main processor will relabel for vertices u, v as follows:

$f(u, v)= f(u, v)+\min\{\text{delta}, c_f(u, v)\}$;
$e(u)= e(u)–c_f(u, v)$;
$e(v)= e(v)+c_f(u, v)$

Put the new unbalanced vertex into set Q.
(6.3) If $\forall u \in V$ $e(u)=0$, eliminate u from active set Q.
Back to step 3.
2.3.2.4. For example
2.3.2.5. Analyse complexity
2.3.2.6. Execute results



Fugire 2.12. Chartper forms the speedup of graph having 7000 nodes and 5000 nodes

2.3.2.7. Conclusion

2.4. Combined flow push and pull algorithm

2.4.1. Postflow-pull algorithm

2.4.1.1. Introduction

2.4.1.2. Some basic concept

◊ Postflow

For network $G = (V, E, c)$. Postflow is a set of flows on the edges

$$f = \{f_{i,j} \mid (i, j) \in E\}$$

So that

(i) $0 \le f_{i,j} \le c_{i,j} \forall (i, j) \in E$

(ii) for any vertex k is not a source or sink, outflow is not smaller than inflow, that is:

$$\sum_{(i,k)\in E} f_{i,k} \le \sum_{(k,j)\in E} f_{k,j}$$

◊ Depth function of the flow in the network $G = (V, E, c)$ is a set of non-negative vertex weights $d(0), ..., d(|V| -1)$ such that $d(a) = 0$ for a and $d(u) + 1 \geq d(v)$ for every edge (u, v) in the residual network for the flow. An eligible edge is an edge (u, v) in the residual network with $d(u)+1=d(v)$.

2.4.1.3. Postflow-pull algorithm

- *Input:*

$G = (V, E, c)$, a is source vertex, z is sink , capacity:

$$c= \{c_{i,j} \mid (i, j) \in E\}$$

- *Output*:

Maximum Flow

$$f = \{f_{i,j} \mid (i, j) \in E$$

Step 1:

Initialized: building the initial postflow in the edges leaving the source to the sink vertices, which is filled to capacity, the other flows are 0. Choose depth function $d(v)$ which is the length of the shortest path from source a to vertex v.

Push all unbalanced vertices into the queue Q.

Step 2:

Condition to terminate: If $Q=\varnothing$, then postflow f becomes max flow, end. If $Q\neq\varnothing$, go to Step 3.

Step 3:

How to perform unbalanced vertex: get unbalanced vertex v from the queue.

Browsing the priority edges $(u, v) \in G_f$. Pull along the edge (u, v) a flow with value $\min\{-\text{delta}, c_f(u, v)\}$, where delta(<0) is the excess of the vertex v. If vertex u is the new unbalanced vertex, then push this vertex u into queue Q.

If vertex v is still unbalanced, then increase the depth of the vertex v as follows:

$$d(v) := 1 + \min\{d(u) \mid (u, v) \in E_f\}$$

then push v into the queue Q.

Back to Step 2.

2.4.1.4. For example

2.4.2. Combined flow push and pull algorithm

This is new algorithm to find maximum flow. Combined algorithm preflow-push and method postflow-pull to build the algorithm combined flow push and pull as follows:

2.4.2.1. Combined flow push and pull algorithm

This is a particular algorithm in *combined flow push and pull* method. Here the unbalanced positive vertices are push into the queue Q_+ and the unbalanced negative vertices are push into the queue Q_-.

With each vertex from the queue Q_+, we will push the flow in the priority edge until the vertex becomes either balanced or does not have any priority edge. If it does not exist priority edge but there are unbalanced vertices, then we increase the height and push it into the queue Q_+.

With each vertex from the queue Q_-, we will pull the flow in the priority edge until the vertex becomes either balanced or does not have any priority edge. If it does not exist priority edge but there are unbalanced vertices, then we increase the depth and push it into the queue Q_-.

2.4.2.2. For example

2.4.3. Parallel aglorithm combined flow push and pull

2.4.3.1. Itroduction

Preflow-push algorithm and postflow-pull algorithm combined the flow push and pull algorithm all of them have complexity $O(|V|^2|E|)$.

To reduce the computer time of this combined the flow push and pull algorithm, we build this algorithm on multiple processors

2.4.3.2. The idea of the parallel algorithm

Parallel algorithms using 3 processors P_0, P_1, P_3. In 3 processors, there is a main processor (P_0) to manage data, divide data into 2 sub-processors (P_1, P_2). Sub-processors P_1 push flow from Q_+.

Sub-processors P_2 pull flow from Q_-. The processors finish when all P_1, P_2 to qual null.

2.4.3.3. Building the parallel algorithm

Input: Graph G (V, E, c) with source a, sink z, the capacity

$$c = \{c_{i,j} \mid (i,j) \in E\}$$

Three processors (P_0, P_1, P_2), where P_0 is the main processor

Output: Maximum flow

$$f = \{f_{i,j} \mid (i,j) \in E$$

Step 1: The main processor P_0 performs

- initialize: h, d, e, f, c, Q_+, Q_-

Step 2: The main processor checks

- Receive datas from the sub-processors
- Checks if Q_+, Q_- is null and P_1, P_2 all of them finish, then f is maximum flow, Stop.

Else go to Step 3.

Step 3: The main processor checks

- Get node u from Q_+ and y from Q_-
- Send h, e, f, u, Q_+ to P_1. Send d, e, f, node y, Q_- to P_2
- The main processor checks: If any the priority edges$(u, v) \in E_f$ and any the priority edges$(x, y) \in E_f$ so that $u=x$ or $y=v$ then go to Step 5.

Else go to Step 4

Step 4: Sub-processors P_1 and P_2 parallel performs

- Two sub-processorsReceive datas from the main processor

- Processor P_1 performs

Preflow-push

Send $Q+$, h, e, f, c_f to the main processor

- Processor P_2 performs

postflow-pull

Send Q_-, d, e, f to the main processor.

Back to Step 2

Step 5: Sub-processors P_1 and P_2 sequential performs

- Processor P_1 performs

Receive datas from P_0 send to in Step 3

Preflow-push

- Processor P_2 performs:

Receive datas from P_0 send to in Step 3 *and r*eceive data from P_1 send to in Step 5

postflow-pull

Send Q_-, d, e, f to the main processor.

Back to Step 2.

2.4.3.4. For example

2.4.3.4. Conclusion

2.5. Chapter Conclusion

In chapter two, we have carefully presented preflow-push algorithm inherited from previous studies and have suggested *Combined flow push and pull* to find maximum flow. After that we optimize *Parallel aglorithm* preflow-push and propose *Parallel aglorithm combined flow push and pull to find* maximum flow. The parallel algorithms are thoroughly presented. The theorems, propositions and consequences related to the algorithms are clearly

demonstrated. The parallel algorithms analyse computation time. In particular, the main content of this chapter is published in three Information Technology journals and listed in [1], [3], [4] in the portfolio of the writer.

CHAPTER 3. PARALLEL ALGORITHMS TO FIND THE SHORTEST PATH AND MAXIMUM FLOW IN EXTENDED GRAPH NETWORK

3.1. Extended graph

Given extended graph $G(V, E)$ with a set of vertices V and a set of edges E, where edges can be directed or undirected. Each edge $\in E$ is weighted $w_E(e)$. With each vertex $v \in V$, E_v is the set of edges of vertex v. Each vertex $v \in V$ and each edge$(e, e') \in E_v \times E_v$, $e \neq e'$ is weighted $w_V(v, e, e')$.

A set (V, E, w_E, w_V) is called extended graph.

3.2. Extended traffic network

Given a graph network $G(V, E)$ with a set of vertices V and a set of edges E, where edges can be directed or undirected. The functions in network as follows:

For c_E edge capacity function: $E \rightarrow R^*$, then $c_E(e)$ is edge capacity $e \in E$

For c_V vertices capacity function: $V \rightarrow R^*$, then $c_V(u)$ is vertices capacity $u \in V$.

For b_E edge cost function: $E \rightarrow R^*$, then $b_E(e)$: cost must be return to transfer an unit transport on edge e

With each $v \in V$, Set E_v is a set of edge of vertice v.

For b_V vertices cost function: $V \times E_v \times E_v \rightarrow R^*$, $b_V(u,e,e')$: cost must be paid to transfer commodity from edge e to vertice u to edge e'.

A set $(V, E, c_E, c_V, b_E, b_V)$ is called extended traffic network.

3.3. Algorithm finding the shortest path in extended graph

3.3.1. Sequential Algorithm

3.3.1.1. Introduction

3.3.1.2. Building the sequential algorithm

- *Input*: Extended graph $G(V, E, w_E, w_V)$, vertex s, $t \in V$

-*Output*: $l(v)$ is the length of the shortest path from s to t and the shortest path (if $l(v) < +\infty$)

- *Steps.*

The algorithm uses the following symbols:

S is a set of vertices that found the shortest path starting from s.

$T=V-S$; $l(v)$ is the length of the shortest path from s to v;

$VE=\{(v, e)|v \in V\backslash\{s\} \& e \in E_V\} \cup \{(s, \phi)\}$ is the set of vertices and edges;

SE is a set of vertex-edge excluded from VE;

$TE=VE-SE$;

$L(v, e)$ is the vertex-edge pair label $(v, e) \in VE$

$P(v, e)$ is the vertex- front edge $(v, e) \in VE$

Step 1. (Initialized) Let $S = \phi$; $T=V$;

Let $VE=\{(v, e)|v \in V\backslash\{s\} \& e \in E_V\} \cup \{(s, \phi)\}$ $SE= \phi$; $TE=VE$;

Set $L(v, e)=\infty$, $\forall (v, e) \in VE$, $L(s, \phi):=0$.

Set $P(v, e)= \phi$ $\forall (v, e) \in VE$

Step 2. Calculate $m = \min\{L((v, e))| (v, e) \in TE\}$.

If $m=+\infty$. Not path. The end.

Else, if $m<+\infty$, choose $(v_{\min}, e_{\min}) \in TE$ so that $L(v_{\min}, e_{\min})=m$,

Set $TE=TE-\{(v_{\min}, e_{\min})\}$, $SE=SE \cup \{(v_{\min}, e_{\min})\}$, go to step 3.

Step 3. If $v_{\min} \notin S$, then set $le(v_{\min})=e_{\min}$, $S=S \cup \{v_{\min}\}$, $l(v_{\min})=L(v_{\min}, e_{\min})$, $T=T-\{v_{\min}\}$.

-if $t \in v_{min}$ then go to step 5, else go to step 4.

Step 4. For any $(v, e) \in TE$ adjacent (post-adjacent) $(v_{\min}, e_{\min})$,

Set $L'(v, e)=L(v_{\min}, e_{\min})+w_E(v_{\min}, v)+ w_V(v_{\min}, e_{\min}, e)$, if $v_{\min} \neq s$ and $L'(v, e)=L(s, \phi)+ w_E(v_{\min}, v)$ if $v_{\min} = s$.

IF $L(v, e)>L'(v, e)$, then set $L(v, e)=L'(v, e)$ and $P(v, e)= (v_{\min}, e_{\min})$ Back to step 2.

Step 5. (Finding the shortest path)

Set $l(t)=L(t, le(t))$ is the length of the shortest path from s to t. From t tracing back the front vertex-edge, we receive the shortest path as following: let $(v_1, e_1)=P(t, le(t))$, $(v_2, e_2)=P(v_1, e_1)$, …, $(v_k, e_k) =P(v_{k-1}, e_{k-1})$, $(s, \phi)= P(v_k, e_k)$.

The shortest path is:

$$s \to v_k \to v_{k-1} \to ... \to v_1 \to t .$$

The end.

Theorem 3.1. The algorithm finding the shortest path from s vertex to t vertices in the extended graph is true.

Theorem3.2. Let G be the extended graph having n vertices. The algorithm has complexity $O(n^3)$.

3.3.2. Parallel algorithm

3.3.2.1. Introduction

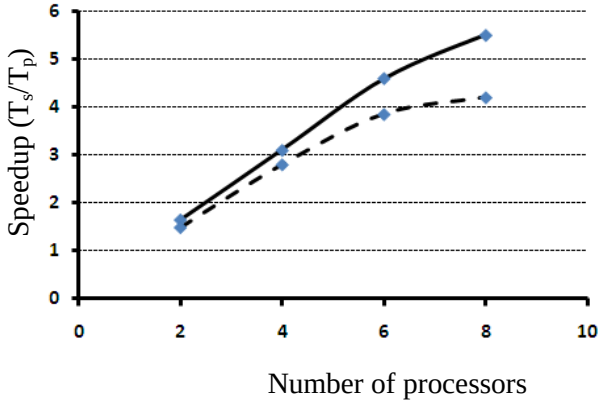
3.3.2.2. The idea of the parallel algorithm

We build parallel algorithms on k processors (P_0, P_1,…,P_{k-1}). In k processors, there is a main processor (P_0) to manage data, divide data into k-1 sub-processors (P_1,…,P_{k-1}) Sub-processors receive the data from the main processor and find minximun L(v, e) on their vertices and send it to main processor. The main processor will find $L(v_{min}, e_{min})$ =min($L_i(v, e)$), i=0,…,k-1) which sub-processors sent.

Next, The main processor will send (v_{min}, e_{min}) to sub-processors to calculate.

3.3.2.3. Building the parallel algorithm

3.3.2.4. Execute results



Fugire 3.4. The speedup of graph having 7000 nodes and 5000 nodes

3.3.2.5. Conclusion

3.4. Algorithm finding concurent max flow with bounded cost

3.4.1. Sequential algorithm

3.4.1.1. Introduction

3.4.1.2. Problem finding concurent max flow with bounded cost

Given is extended traffic network $G = (V, E, c_E, c_V, b_E, b_V)$.$G$ have k pairs of source and destination (s_j, t_j), any pair assign a j, j=1, …, k transporttation.

Any a j commodity with $d(j)$ unit commodity moves from node source s_j to node sink t_j, $\forall j = 1, ..., k$. Given limit cost B. Task problem is finding λ number maximum so that there is a multicommodity flow to transfer λ. $d(j)$ unit commodityj pass flow, $\forall j = 1, ..., k$ concurent, sum cost of flow is not bigger than B.

3.4.1.3. Algorithm finding concurent max flow with bounded cost

3.4.1.4. Present algorithm by C

◊ Input:

1) Extended traffic network$G = (V, E, c_E, c_V, b_E, b_V)$.

2) Demand (s_j, t_j, d_j), j=1, …, k.

3) Limit cost B. approximate coefficient $\omega > 0$.

◊ Output:

1) Coefficient λ maximun: λ_{max}

2) Practical flow $\{fe_j(a), fv_j(u,e,e')| a \in E, (e,u,e') \in$ Table b_v, j=1,...,k}.

3) Practical cost $B_f \leq B$.

Put $\varepsilon = 1 - \sqrt[3]{\frac{1}{1+\omega}}$; $\delta = \left(\frac{m+n+1}{1-\varepsilon}\right)^{-\frac{1}{\varepsilon}}$;

$le(e) = \delta c_E(e), \forall e \in$ E; $lv(v) = \delta c_V(v), \forall v \in V$; $\varphi = \delta / B$;

$D = (m+n+1)\delta$; $fe_j(a) = 0$; $\forall a \in E$,

$fv_j(u,e,e') = 0$; $\forall u \in V, \forall (e,u,e') \in$ Talbe b_v, j=1,...,k

t= 1;

$B_{ex} = 0$;

while ($D < 1$)

{

for j = 1 to k do

{

$d' = d_j$

while $d' > 0$ do

{

$$\text{length(p)} = \sum_{i=1}^{h+1} le(e_i) + \sum_{i=1}^{h} lv(u_i)$$

$$+ \text{b(p)}.\varphi = \sum_{i=1}^{h+1} \left[\varphi.b_E(e_i) + le(e_i)\right]$$

$$+ \sum_{i=1}^{h} \left[\varphi.b_V(u_i, e_i, e_{i+1}) + lv(u_i)\right]$$

$f'=\min\{d', c_E(e), c_V(v) | e \in p, v \in p\}$;

$B' = b(p)*f'$;

if $B' > B$ $\{f' = f'*B/B'; B' = B\}$;

$fe_j(a) = fe_j(a) + f'; \forall a \in p$

$fv_j(u,e,e') = fv_j(u,e,e') + f'; \forall (e,u,e') \in p$

$d' = d' - f'; \varphi = \varphi*(1+\varepsilon*B'/B)$;

$le(e) = le(e)*(1+\varepsilon*f'/c_E(e)); \forall e \in p$

$lv(v) = lv(v)*(1+\varepsilon*f'/c_V(v)); \forall v \in p$

$D = D + \varepsilon*f'*length(p)$;

$B_{ex} = B_{ex} + B'$;

} //End while $d' > 0$

} //End for

$t = t + 1$;

} //End $D < 1$

$$c' = max\{\frac{le(e)}{\delta / c_E(e)}, \frac{lv(v)}{\delta / c_V(v)}, \frac{\varphi}{\delta / B} | e \in E, v \in V\};$$

$c_{ex} = log_{1+\varepsilon} c'$;

$fe_j(a) = fe_j(a)/c_{ex}; \forall a \in E, j=1,...,k$

$fv_j(u,e,e') = fv_j(u,e,e')/c_{ex}; \forall u \in V, \forall (e,u,e') \in B$ång $b_v, j=1,...,k$

$$B_f = B_{ex}/c_{ex}; \lambda_{max} = \frac{t}{c_{ex}};$$

3.4.2. Parallel algorithm finding concurent max flow with bounded cost

3.4.2.1. Introduction

3.4.2.2. The idea of the parallel algorithm

We build parallel algorithms on m processors P_1, $P_2,\ldots,P_m$. In m processors, there is a main processor P_1 to manage data, divide data into

m-1 sub-processors $P_2,\ldots,P_m$. Sub-processors receive the data from the main processor

Main processor P_1 divide k demand (s_j, t_j, d_j), $j=1,\ldots,k$ into *m* processors.

m-1 sub-processors receives the demand that main processor sent to and multiplication perform and m times d_j then perform and independent calculations on the set demand. The result on m-1 sub-processors will be sent to the main processor, the main processor will sum the results and divide into main.

$$\lambda_{\max} = \min\{\lambda_1, \lambda_2, ..., \lambda_m\}$$

3.4.2.3. Steps performing parallel algorithms

3.4.2.4. For example

3.4.2.5. An analysis of complexity

3.4.2.6. Conclusion

Parallel algorithms reduce greatly computation time than sequential algorithms do. The Parallel algorithms are built systematically and are clearly proved.

3.5. Chapter Conclusion

In this chapter, we have proposed two algorithms: Algorithm finding the shortest path in extended graph and Algorithm finding concurent max flow with bounded cost. The findings are comprehensively and logically demonstrated. In particular, the main content of this chapter is published in three Information Technology journals and listed in [2], [5], [6] in the portfolio of the author related to the thesis.

CONCLUSION

A study on *"parallelization of algorithms in graph network"* focus on building five following parallel algorithms:

1. Parallel algorithm pre-fow push to find maximum flow.

2. Parallel algorithm combined flow push and pull algorithm to find maximum flow.

3. Parallel algorithm finding the shortest path in extended graph.

4. Parallel algorithm finding concurent max flow with bounded cost.

Here are some main findings of the study.

Firstly, Investigating and discovering parallel processing theory, investigating graphic theiory, mainly focus on parallel algorithms finding the shortest and finding the maximum flow in normal graph and extended graph.

Secondly, Suggesting new algorithms finding the maximum flow, we underpin other existing algorithms to analyse, evaluat and prove our algorithms. Form that we parallel the sequential algorithms.

Thirdly, Suggesting a new parallel algorithm for below algorithms detailedly, proving soundness, the complexity of the algorithm with experimental programs on various parallel systems.

Fourthly, we excute the algorithms on a number of processors. After that, we evaluate, compare the computation time between parallel algorithms and sequential algorithms.

PUBLICATIONS OF THE AUTHOR

[1] Tran Quoc Chien, Nguyen Dinh Lau, Nguyen Thi Tu Trinh, *Sequential and Parallel Algorithm by Postflow-Pull Methods to Find Maximum Flow*, Proceedings 2013 13th International Conference on Computational Science and Its Applications, ISBN:978-0-7695-5045-9/13 $26.00 © 2013 IEEE, DOI 10.1109/ICCSA.2013.36, published by CPS.

[2] Nguyen Dinh Lau, Tran Quoc Chien, Le Manh Thanh, *Improved Computing Performance for Algorithm Finding the Shortest Path in Extended Graph*, proceedings of the 2014 international conference on foundations of computer science (FCS'14), July 21-24, 2014 Las Vegas Nevada, USA, Copyright © 2014 CSREA Press, ISBN: 1-60132-270-4, Printed in the United States of America, pp 14-20.

[3] Nguyen Dinh Lau, Le Manh Thanh, Tran Quoc Chien, *Sequential and Parallel Algorithm by preflow-Push Methods to Find Maximum Flow*, Journal of special Issue for Electronics, Communications and Information Technology of the National Academy of Science and Technology Vietnam & Telecommunication Institute, No 51(4A)2013 ISSN: 0866 708X, 109- 125.

[4] Tran Quoc Chien, Le Manh Thanh, Nguyen Dinh Lau, *Sequential and parallel algorithm by combined the push and pull methods to find maximum flow***,** Proceeding of national Conference on Fundamental and Applied Infromation Technology Research (FAIR), *Hue, Vietnam*, 20-21/6/2013.ISBN: 978-604-913-165-3, 538-549.

[5] Nguyen Dinh Lau, Tran Quoc Chien, Le Manh Thanh, *Parallel algorithm to divide optimal linear flow on extended traffic network,* Research, Development and Application on Information & Communication Technology, Ministry of Information & Communication of Vietnam, No 3, V-1, 2014, pp 15-28.

[6] Tran Quoc Chien, Le Manh Thanh, Nguyen Dinh Lau, *Parallel algorithm to find maximum flowcostlimits on extended traffic network,* Proceedingnational Conference XVI "Some selected issues of Information Technology and Communications" Danang 14-15/11/2013, ISBN: 978-604-67-0251-1, 314-321.

[7] Nguyen Dinh Lau, Tran Ngoc Viet, *Parallelizing algorithm finding the shortest paths of all vertices on computer cluster system*, Proceedings national Conference XVthSome selected issues of Information Technology and Communications, Ha Noi, 03-04-2012, 403-409