Deadlock Detection Based on the Parallel Graph Theory Algorithm

Xiaorui Wang, Qingxian Wang, Yudong Guo
China National Digital *Switching* System Engineering and Technological Research Center
Zhengzhou, China
henanwxr@sina.com

Jianping Lu
Chongqing Communication Institute
Chongqing, China
lu8311@yahoo.cn

***Abstract*—Deadlock detection and release is an important means to maintain the concurrency of task in operating system, the relationship between system process and resources is usually described by means of resource allocation graph, the core of the deadlock detection is to determine whether there is a loop in the resource allocation graph and simplify the resource allocation graph. If using the traditional serial method to process the resource allocation graph, it will cause delay in the system even impossible to unlock in condition of a large number of processes exist in the system. This paper introduces the parallel processing of graph for deadlock detection, including two aspects: using the transitive closure algorithm on SIMD-CC model for loop detection in resource allocation graph, using the parallel P-BFS algorithm for Simplification of resource allocation graph. By parallel processing on the serial detection process, efficiency of the system in dealing with deadlock has been improved.**

Keywords-deadlock detection; parallel computing; resource allocation graph; loop detection; SIMD-CC

I. INTRODUCTION

The concurrent execution of tasks in modern operating systems can improve the utilization rate of resources, but it may also lead to the occurrence of deadlock. If the deadlock occurs it will waste a lot of system resources and even cause the system to crash, therefore the detection and handling mechanism of deadlock is the important guarantee to maintain the collaboration between the multitasking[1-5].

In graph theory, graph is constituted by a number of the given vertices and the edges that connecting the vertices, this graphic is usually used to describe certain relations between things, vertices represent things, edges represent that it has a relationship between the corresponding two things. Although many graph theory problems are easy to express, but they are difficult to be solved. It has proven that there are quite a lot of graph theory problems are NP-Complete problems. For example: the Hamiltonian path problem, the graph isomorphism problem, the graph coloring problem, and so on[6-10].With the development of VLSI(Very Large Scale Integration) technology and the emergence of parallel computers, it provides a new way for the fast solution of the problem of graph theory[10]. Using multi-processor systems to solving the problem in graph theory has played an important role in the application of scientific and engineering [11-15].

Deadlock detection usually use the concept graph theory, such as digraph, undigraph, connected sub-graph, adjacency matrix, etc.. Graph theory algorithms have been extensively studied in the multi-processor system, for example: parallel searching of the graph, transitive closure of the graph, connected components of graph, the shortest path of the graph, minimum spanning tree of the graph. If the parallel searching can be applied into the deadlock detection, the efficiency of the deadlock detection will be greatly improved. In this paper, the relationship between the system processes and the resources are described by RAG(Resource Allocation Graph). In order to improve the efficiency of the deadlock detection, serial processing for loop detection of RAG and RAG simplifying is done by parallel processing during the course of deadlock detection.

II. DESCRIPTION OF THE DEADLOCK

A. Deadlock definition

The so-called deadlock refers to the concurrent processes waiting for the resources owned by others, and these concurrent processes don't release their own resources before they getting resources from others, therefore resulting the state that each of the processes wants to get the resources but they can't, so that the concurrent processes can't continue to move forward.

Generally, the deadlock can be described as: there are concurrent processes $P_1, P_2, \ldots, P_n$, they share the resources $R_1, R_2, \ldots, R_m$ $(n > 0, m > 0, n \geq m)$. Wherein each $P_i (1 \leq i \leq n)$ has the resources R_j $(1 \leq j \leq m)$ until there is no remaining resources. Meanwhile, each P_i requests to get R_k $(k \neq j, 1 \leq k \leq m)$ in the premise of not release R_j, resulting the mutual occupation and waiting for each other of the resources. This group of concurrent processes will stop to push ahead and fall into a permanent state of waiting in case there is no external drive.

The system state are as follows when the deadlock taking place: *1)* at least two processes involved in the deadlock. *2)* at least two of the processes involved in the deadlock already hold the resources. *3)* all the processes involved in the deadlock are waiting for resources. *4)* processes involved in the deadlock is a subset of all the processes in the current system.

For example, there is a printer R_1 and a card reader R_2, the process P_1 occupies printer R_1, the process P_2 occupies

Published by Atlantis Press, Paris, France.

card reader R_2 . If P_1 requests to get card reader R_2 , P_2 requires to get the printer R_1 , then it will be fall into the state of deadlocked, as shown in Figure 1.

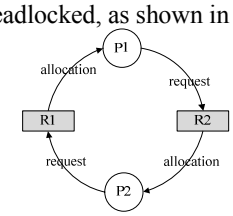

Figure 1. schematic diagram of deadlock

B. The cause of deadlock

Essentially, there are two reasons for the generation of deadlock: *1)* competition for resources, multiple processes competing for resources but these resources can't meet their needs at the same time. *2)* The promoting order of processes are improper, that is the order that these processes apply and release resources are improper. Further analysis, there are four necessary conditions to produce the deadlock:

1) Mutually exclusive using of resources, one resource can only be used by one process each time.

2) Resources can't be deprived, resources owned by the occupants can't be forcibly seized from the applicants, and the resources can only be released voluntarily by the occupants.

3) Requesting and holding, the process which applying for a new resource hold the possession of the original resource.

4) Loop waiting, there exists a waiting queue $\{P_1, P_2, \ldots, P_n\}$ for the processes, wherein P_1 waits the resource occupied by P_2 , P_2 waits the resource occupied by P_3 ,…, P_n waits the resource occupied by P_1 ,so it forms a waiting loop of processes.

Generally, the solution of the deadlock can be divided into three categories: prevention, avoidance, detection and recovery. Deadlock prevention is a static strategy, it determines the resource allocation algorithms in period of system design, it ensures no deadlock occurring by limiting requests for resources of the concurrent processes. The specific approach is to destroy one of the four necessary conditions of deadlock, making the necessary conditions of deadlock are not satisfied at any time during the execution of the system. Deadlock avoidance refers to make predictions in advance according to the usage of resources when these resources are allocated by system, thereby avoiding the occurrence of deadlock. Deadlock avoidance is a dynamic strategy, the most representative algorithm for deadlock avoidance is the Banker's Algorithm. Deadlock detection and recovery refers to allow the generation of deadlock and the operating system takes remedial measures after the emergence of deadlock. The operating system has special agency and the agency is able to detect the location and cause of the deadlock when a deadlock occurs. It recovers the concurrent processes from the state of deadlock through damaging necessary condition for a deadlock with the help of external force.

It is a difficult thing that achieve the purpose of excluding deadlock through the means of prevention and avoidance. Deadlock detection and recovery will be able to find deadlock and recover from deadlock and it needs not have to spend much of execution time. Therefore, the approach of deadlock detection and recovery is used for excluding deadlock in most of the actual operating system.

III. DEADLOCK DETECTION AND HANDLING

A. Process description based on RAG

The digraph is able to descript the state of process deadlock accurately, the RAG can be defined as a two-tuples: $RAG = (V, E)$, wherein V is the set of vertices, E is the set of directed edges.

Set of vertices can be divided into two parts: (1) $P = (P_1, P_2, \ldots P_n)$ is the set composed by all processes within the system, each P_i $(P_i \in P)$ represents a process.(2) $R = (R_1, R_2, \ldots R_m)$ is the set composed by all resources within the system, each R_i $(R_i \in R)$ represents a class of resources. Each edge in the set of edge is an ordered pair $< P_i, R_i >$ or $< R_i, P_i >$. If $< P_i, R_i > \in E$, then there exists a directed edge from P_i to R_i , it represents that P_i requesting one resource in class of R_i ,and it's waiting to be assigned currently. If $< R_i, P_i > \in E$, then there exists a directed edge from R_i to P_i , it represents that the one resource in class R_i has been allocated to the process P_i .The directed edge $< P_i, R_i >$ is called a request edge, and the directed edge $< R_i, P_i >$ is called an allocation edge.

The circles represent processes and the boxes represent each type of resource in RAG. There may be multiple instances of each type of resource R_i , the instances of each resource can be expressed by the dot in the box. The request edge is a directed edge that from process to resource, it represents that the process applies for a resource, but the process is waiting for the resource.

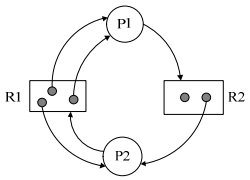

Figure 2. diagram of resource allocation

The allocation edge is a directed edge that from resource to process, it represents that one resource has been allocated to a process, as shown in figure 2.

B. *Deadlock detection theorem*

It is assumed that a set of processes using a set of resource and the state of the system is S at a certain time, RAG is the graph corresponding to state S then:

1) If it doesn't appear any loop in RAG, then S is in non-deadlocked state, or said the security state. *2)* If it appears loop in RAG, and each resource in the loop is single-unit resource(only one allocation unit), then S is in a deadlock state. In other words, the loop that constituted by single-unit resources is the necessary and sufficient condition of deadlock for state S. *3)* If it appears loop in RAG, but not all of the resources in this loop are single unit resource, then the S is not necessarily in the deadlock state. In other words, the loop that only constituted by part of single-unit resources is a necessary but not a sufficient condition of deadlock for state S.

The approach of RAG Simplification are as follows:

1) Find a non-isolated node P_i which has only allocation edge, or although this node has request edge but the request edge can be immediately converted to allocation edge(that is the request can be satisfied immediately), then eliminate all of directed edges of P_i to make it to be the isolated node(that is release all the resources owned by P_i).

2) Assume that R_k is a certain resource node which released by a process node P_i, than the request edge of another node P_j requests to R_k : $P_j \rightarrow R_k$ can be converted into allocation edge, that is the resource R_k released by P_i can be allocated to process P_j .If P_j has only allocation edges after a series of conversion, then then making P_j to be the isolated node.

3) After a series of simplification, if all the directed edges in RAG can be eliminated and all process nodes are made to the isolated nodes, then the RAG is said to be completely simplified, otherwise this RAG is not completely simplified. Obviously, the RAG that can't be simplified completely must has loops.

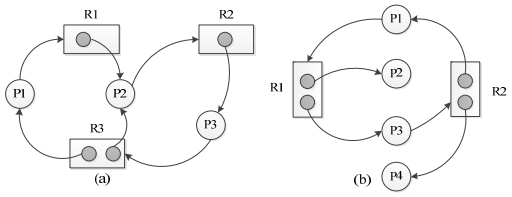

Figure 3. loop and deadlock in RAG

Deadlock theorem description: the necessary and sufficient condition that state S is in deadlock state if and only if the RAG of S can't be simplified completely. Deadlock with loop and deadlock without loop in RAG are shown as (a) and (b) in Figure 3.

IV. PARALLEL PROCESSING OF DEADLOCK DETECTION

A. *Analysis of parallel processing for RGA*

In computers, $G = (V, E)$ represents the data structures that used by RAG, there are usually adjacency matrix and adjacency list. However, it is more convenient using the adjacency list when searching edges in the graph.

1) Parallelization of loop detection in RAG

The process of deadlock detection has been mentioned in section 2.2, firstly it needs to determine whether there exists a loop in the given RAG, the problem can be solved quickly with the help of transitive closure algorithm of graph, transitive closure of directed graph expresses the reachability between each vertex. Transitive closure of graph is defined as follows:

$G = (V, E)$ represents directed graph and $A = (a_{ij})_{n\times n}$ represents adjacency matrix, the transitive closure of A is $A^+ = (b_{ij})_{n\times n}$ and b_{ij}=1 iff there exists a path between vertex i to vertex j, as follows:

$$b_{ij} = \begin{cases} 1 & i = j \text{ or existing a path from } i \text{ to } j \\ 0 & \text{else} \end{cases}$$

Figure 4 shows a directed graph, the adjacency matrix of the graph and its transitive closure.

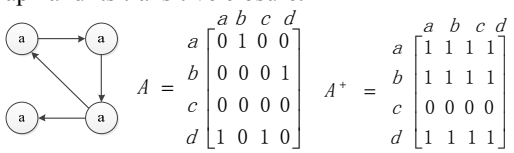

Figure 4. adjacency matrix of graph and its transitive closure

It needs to do the operation of matrix multiplication in the process of transitive closure calculation, the time complexity of serial algorithm for transitive closure is $O(n^3)$. An important advantage of parallel computing is the processing of matrix operation, typical matrix multiplication comprises[16]: Fox,Systolic,Cannon,DNS,etc.,Therefore it can take parallelization measures in the course of transitive closure calculation.

2) Parallelization of simplification in RAG

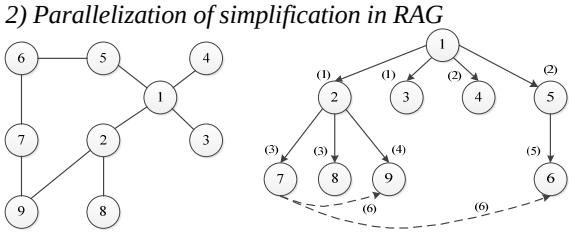

Figure 5. P-BFS with two processors

In the process of RAG simplification, it needs to search the graph when eliminating the edges between the processed and resources. A process is usually associated with multiple resources (already assigned or been applying), all of the edges associated with this process can be detected at the same time taking advantage of the parallel ability of multi-processor. Therefore, parallel searching can be employed in the process of RAG simplification, and P-BFS(parallel

breadth first search) is appropriate. Figure 5 shows the process of P-BFS with two processors.

B. Parallel algorithms of transitive closure for RAG

Conventions used in this paper: n represents the number of vertices of graph in $G=(V,E)$, m represents the number of edges, d_i represents the degree of vertex n_i, p represents the number of processors.

1) Algorithm principle based on the Boolean matrix multiplication

Let B=A+I, I represents the unit matrix, $B=(b^{(1)}_{ij})_{n\times n}$,then if $b^{(1)}_{ij}=1$, i=j or there exists directed edge from i to j. That is the length of directed path is 0 or 1 from i to j.

For $B^2=(A+I)^2=(b^{(2)}_{ij})_{n\times n}$, $b^{(2)}_{ij}=\vee_{k=1\sim n}b^{(1)}_{ik}b^{(1)}_{kj}$, here $\vee$ is the logical symbols or, then $b^{(2)}_{ij}=1$, that is the length of directed path is L≤2 from i to j.

Analogously, $B^k=(b^{(k)}_{ij})_{n\times n,}$ $b^{(k)}_{ij}=1$, the length of directed path is L≤k from i to j. Therefor $A^+=B^r$ (r≥n-1). So it has total $\lceil \log(n-1) \rceil$ times of multiplications in :

$B\rightarrow B^2\rightarrow B^4\rightarrow\cdots\rightarrow B^{2^{\lceil \log(n-1) \rceil}}=A^+$.

2）Transitive closure algorithm on SIMD-CC model

n^3 processors arranged in a three-dimensional array of n×n×n, the coordinate of P_r is (i,j,k), there are there registers A(i,j,k), B(i,j,k) and C(i,j,k), therein $r=i*n^2+j*n+k$, (0≤i,j,k≤n-1). The initial: A(0,j,k)←ajk 0≤j,k≤n-1; The end: C(0,j,k)is the element of (j,k) in A^+. Input: $A_{n\times n}$, Output: $C_{n\times n}$. Algorithm descripted as in table I.

TABLE I. TRANSITIVE CLOSURE ALGORITHM ON SIMD-CC MODEL

```
Begin
 (1) Add to unit matrix: Par-do A(0,j,j)←1, 0≤j≤n-1
 (2)A is copied to B: Par-do B(0,j,k)← A(0,j,k), 0≤j,k≤n-1
 (3)for i=1 to ⌈log(n − 1)⌉ do
    (3.1)C←A×B // Call DNS multiplication algorithm, O(logn)
    (3.2)par-do: A(0,j,k)← C(0,j,k), B(0,j,k)← C(0,j,k), 0≤j,k≤n-1
    endfor
End
```

3）DNS algorithm

Let $r^{(m)}$represents the negating of m-th bit of r； {p, r_m=d}represents the set of r(0≤r≤p-1), d is the m-th bit of r in its binary form. input: $A_{n\times n}$, $B_{n\times n}$; output: $C_{n\times n}$. Algorithm descripted as in table II.

C. Parallel algorithm of P-BFS for RAG

It usually has a primary table and sub-table in the process of parallel computing. The general process of parallel search is as follows:(1) initially let the primary table empty, then obtains a certain vertex as the initial search point from the graph and adds it to the primary table. (2) In any one of the search process, each processor first sets the sub-table empty, then selects one vertex to be searched from the primary table, one or several edges which associated with the vertex are checked as follow: if the edge connected to this vertex has not been searched, it will be placed in the sub-table of corresponding processor. Each sub-table stores some vertices that will be merged into the primary table, all the sub-tables will be linked together and added to the primary table at certain time.

TABLE II. DNS ALGORITHM

```
Begin
(1)for m=3q-1 to 2q do
        for all r in {p, r_m=0} par-do
          (1.1) A_r(m) ← A_r
                (1.2) B_r(m) ← B_r
        endfor
      endfor
(2)for m=q-1 to 0 do
        for all r in {p, r_m= r_2q+m} par-do
          A_r(m) ← A_r
        endfor
      endfor
(3)for m=2q-1 to q do
        for all r in {p, r_m= r_q+m} par-do
          B_r(m)← B_r
        endfor
    endfor
(4)for r=0 to p-1 par-do
        C_r=A_r×B_r
      endfor
(5)for m=2q to 3q-1 do
        for r=0 to p-1 par-do
          C_r=C_r+C_r(m)
        endfor
      endfor
   End
```

In the process of P-BFS searching, the primary table is a FIFO queue, if a vertex i which is to be searched is selected from the primary table, the vertex i needs to deleted from the primary table, then p processors check edges associated with vertex i and put these vertices that has not been searched into sub-tables. Vertices to be search are still selected from the primary table in next search, these sub-tables will be linked and merged to primary table until the main table is empty, the searching process stops if these sub-tables are still empty after they have been linked. According to the above description, the algorithm flow of P-BFS is shown in Figure 6.

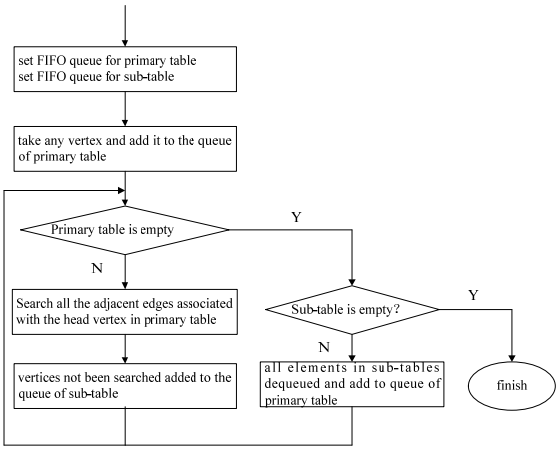

Figure 6. Algorithm flow of P-BFS

V. ANALYSIS OF ALGORITHM COMPLEXITY

Main measures of complexity of parallel algorithm including: running time t(n), the number of processors p(n), and the cost c(n), therein c(n)=t(n)×p(n).

1) Complexity of the transitive closure algorithm

It needs to do operation of matrix multiplication in the process of computing transitive closure, there is $\lceil \log(n-1) \rceil$ times multiplication from B→B2→B4→…→$B^{2^{\lceil \log(n-1) \rceil}}$ to A+, the time complexity is O(logn) for calling DNS multiplication algorithm each time. Then: the time complexity t(n)=O($n \log^2 n$), the number of processors p(n)= n^3, the algorithm cost c(n)=O($n^3 * \log^2 n$).

2) Algorithm complexity of P-BFS

Assume that the time-consuming operation including vertex selection, tables linking and binding, and also assumes that it need one of these operations when selecting a vertex. For serial algorithm, there is only one primary table, and only one vertex which has not been searched is added to the primary table when checking an associated edge each time. Assuming $d_i (i \in V)$ is degree of the vertex, then the upper bound of searching time of serial algorithm for a graph is:

$$T_s = \sum_{i=1}^{n} (d_i + 1) = 2m + n$$

The operation of linking and binding takes $\lceil \log p \rceil + 1$ time after searching each floor of the graph, checking edges that associated with every vertex takes $\lceil d_i / p \rceil + 1$ time, there are total L layers searching from the starting point in entire graph, then the searching time T_p of P-BFS is:

$$T_p = \sum_{i=1}^{n} (\lceil d_i / p \rceil + 1) + L \times (\lceil \log p \rceil + 1)$$

$$\leq \sum_{i=1}^{n} (d_i / p) + L \times \lceil \log p \rceil + 2n$$

$$\leq T_s / p + L \times \lceil \log p \rceil + 2n$$

So the running time is: $\sum_{i=1}^{n} (\lceil d_i / p \rceil + 1) + L \times (\lceil \log p \rceil + 1)$, the number of processors is p, and the cost of algorithm is $[\sum_{i=1}^{n} (\lceil d_i / p \rceil + 1) + L \times (\lceil \log p \rceil + 1)] \times p$.

VI. CONCLUSIONS

In normal course of operation system running, the operating system needs to constantly monitor the state of the processes, and detects whether the system generates deadlock through related operations to RAG, it needs to relieve the deadlock once the deadlock is detected and recovery the running of operating system with minimal cost, for example: restart, undo the process, deprived of resources, processes fallback, etc.. Take advantage of the parallel algorithm of graph can quickly complete loop detection and search for RAG, which will significantly shorten the response time and improve the continuous running ability for operation system. It also exists other similar situations that can make use of the parallel algorithm in the operating system, such as process trees search, the parent and child trees destruction, it needs to fully explore these type of serial process which can make use of the parallel computing.

REFERENCES

[1] Robles-Gomez, A. “A deadlock-free dynamic reconfiguration scheme for source routing networks using close up*/down* graphs” , Parallel and Distributed Systems, vol. 22 , no. 10 , pp.1641 – 1652, Oct. 2011.

[2] Chunfu Zhong, Zhiwu Li. “A Deadlock Prevention Approach for Flexible Manufacturing Systems with Uncontrollable Transitions in Their Petri Net Models”, Engineering with Computers, vol. 25, no. 3, pp.269-278,
Sep. 2009.

[3] Keyi Xing, Mengchu Zhou, Feng Wang, Huixia Liu, Feng Tian. “Resource-transition circuits and siphons for deadlock control of automated manufacturing systems”, Systems, Man and Cybernetics, vol. 41, no. 1, pp. 74- 84, Jan. 2011.

[4] Hesuan Hu, MengChu Zhou, Zhiwu Li. “Supervisor design to enforce production ratio and absence of deadlock in automated manufacturing systems”, Systems, Man and Cybernetics, vol. 41 , no. 2, pp.201- 212, Mar. 2011.

[5] Colin S. Gordon, Michael D. Ernst, Dan Grossman. “Static lock capabilities for deadlock freedom”, TLDI '12 Proceedings of the 8th ACM SIGPLAN, New York, 2012, pp.67-78.

[6] Skupie n, Zdzis aw, Borowiecki, M. Combinatorics and graph theory, Poland: Warsaw, 1989, pp.201-247.

[7] W. T. Tutte. Graph theory as I have known it. Oxford University Press, May. 2012, pp.25-164.

[8] Miklos Bona. A walk through combinatorics: an introduction to enumeration and graph theory, World Scientific, 2011, pp. 321-456.

[9] Martin Charles Golumbic, Irith Ben-Arroyo Hartman. Graph theory, combinatorics and algorithms: interdisciplinary applications, Springer Publishing Company, 2011, pp.112-298.

[10] CHEN Guo-Liang, LIANG Wei-Fa, SHEN-Hong. “RESEARCH ADVANCES IN PARALLEL GRAPH ALGORITHMS”, Journal of Computer Research and Development, no. 9, 1995.

[11] TANG Ce-Shan, LIANG Wei-Fa. “Parallel Graph Theory Algorithm” Press of China Science and Technology University, Oct. 1991.

[12] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, et.al. “A view of the parallel computing landscape”, Communications of the ACM, vol. 52, no. 10, pp. 56-67, Oct. 2009.

[13] Haoqiang Jina, Dennis Jespersena, Piyush Mehrotraa, Rupak Biswasa, Lei Huangb, Barbara Chapmanb. "High performance computing using MPI and OpenMP on multi-core parallel systems", Parallel Computing, vol. 37, no. 9, pp. 562–575, Sep. 2011.

[14] Rainer Keller,David Kramer,Jan-Philipp Weiss. Facing the multicore-challenge: aspects of new paradigms and technologies in parallel computing, Springer, 2010, pp.45-98.

[15] Jaliya Ekanayake, Geoffrey Fox. "High performance parallel computing with clouds and cloud technologies", Cloud Computing, vol. 34, pp.20-38, 2010.

[16] CHEN Guo-Liang. Parallel Algorithm Design and Analysis, Beijing: Higher Education Press, Nov. 2002. pp.208-325.

Published by Atlantis Press, Paris, France.