Research on Homogeneous Cluster-based Hierarchical Nested String Matching Parallel Algorithm

Lei Li, Yuyuan Gu, Yanyan Guo, Kemeng He, Yan Chen and Yuqiang Sun

Changzhou University International Institute of Ubiquitous Computing,
Jiangsu, 213164, Changzhou, China
Jiangsu University College of Electronic and Information Engineering,
JiangSu, 212013, Zhenjiang, China

Abstract: Research a master-slave duplex layer structure exact string matching parallel algorithm on the homogeneous PC cluster platform. Making use of the KMP main string mismatching irrelevant feature to construct secondary KR substring and then calculate the value of next function quickly. Using PC Cluster Message Passing Interface (MPI) parallel platforms and for the balanced load of each node processes which involved in computing, an anti-missing methods was proposed to do overlapped block with the target dictionary. The paralleled implementation process of hierarchical nested exact string matching algorithm was discussed in detail. The experiment demonstration is conducted on the homogeneous PC cluster platform, analyzing the parallel processing efficiency of data traffic on different levels under the condition of the intervention of multi nodes. Experimental results show that the algorithm has a high parallel efficiency and linear acceleration was obtained and has good scalability.

Key words: Homogeneous cluster, MPI, KMP algorithm, KR algorithm

INTRODUCTION

String matching is one of the basic and important research issues in computer science. What is called string matching (string matching, pattern matching), is that given a specific group of string set P, for any string T, to find out all position strings in P appears in string T (Liu et al., 2010). With the development of technology and the explosive growth of the amount of information, the matching efficiency requirements are getting higher increasingly, especially in the domain such as information retrieval and data mining. It is impractical to solve such problems by using traditional string matching algorithm. We must use efficient parallel computing to solve such problems. PC cluster has the advantage of cost-effective, strong scalability, short development cycle and so on. As of November 2007, among the Top500 supercomputers in the world, approximately 87.6% of the processors used multi-core chip and approximately 81.2% of the supercomputer adopted the cluster structure and is the mainstream computer platform of MPI-driven parallel applications (Tu et al., 2008).

To find a parallel hierarchical nested exact string matching algorithm by using the cluster system on the MPI parallel platform. According to the feature of homogenous PC cluster and taking full account of the load-balance of each processing node, we analyzed the effect that the data traffic and processing node with different magnitude have on parallel efficiency.

RELATED WORK

Homogeneous PC cluster: PC clusters (Clusters) is a loosely-coupled parallel computer systems formed by linking up a group of loosely integrated computer software and/or hardware together, which usually connecting by LAN or other means and accomplish computational work by closely collaborating multiple node computer.

According to the architecture, they can be divided into homogeneous and heterogeneous, the structure and performance of all computational nodes of homogeneous PC cluster system (Fig. 1) are identical and they run the same operating system which can effectively avoid different storage capacity, processing efficiency differences and other issues between different computational nodes and control the low communication consumption between the node and the computational nodes.
Control node

Computational node

Unified Messaging Interface

Fast Ethernet or switch

Control node

Fig. 1: homogeneous PC cluster with a control node

![Diagram of a homogeneous PC cluster with a control node.](image)

Fig. 2: Overlapped blocking text string

**Load-balancing of overlapping block:** The load-balancing in cluster system can also be understood as the problem of energy-efficient scheduling, which used to allocating processors and other resources for each parallel task in the system, assigning the starting and ending time these resources occupied and under the condition of satisfying for the dependency, tasks can be completed earlier and the energy consumption can be as little as possible (Li et al., 2012). In general, the strategy load scheduling can be divided into 3 categories: priority-based scheduling, packet-based (Du et al., 2006) scheduling and replication-based scheduling (Zong et al., 2011). Rantonen et al studied a kind of fair load sharing approach with vague minimum computational overhead (Rantonen et al., 2010). Zong et al proposed 2 non-preemptive offline scheduling algorithm energy-EAD (Energy-Aware Duplication) and PEBD (Performance-Energy Balance Duplication) scheduling algorithm on the basis of the TDS (Zong et al., 2011).

In this study, a parallel two-chain nesting exact string matching algorithm was studied and we take full consideration on the context independence of the target dictionary string collection and overlapped-block the target string, then after detecting the length of the pattern string, we work out the length of the overlapped string and the lack of distribution can be effectively avoided (Fig. 2).

\[ t(n) = \frac{t + p - \ln n}{n} \]  

(1)

The length of the \( \sigma \) can be deduced by the formula:

\[ t = n \left( \frac{t + p - 1}{n} \right) \]  

(2)

\[ \sigma = \frac{n \left( \frac{t + p - 1}{n} \right) - t}{n - 1} \]  

(3)

\( n \) is the nodes number that the cluster could process, in a real world application, the number of the processor should usually be limited within \( F(n) - n(1-\varepsilon) \), \( 0 < \varepsilon < 1 \).

**MPI and parallel implementation:** MPI (Message Passing Interface) is one of the standard of message passing interface in parallel, it provides an efficient, scalable and unified parallel programming environment. MPI is a complex system, which includes over 200 functions and has 6 basic functions that constitute the subset of MPI: MPI_INIT, MPI_FINANLIZE, MPI_COMM_SIZE, MPI_COMM_RANK, MPI_SEND and MPI_RECV.

During the implementation of MPI, at the initialized time the program will generate a process in each processing node and not point out how to activate and generate the process, but turn over to the system area. MPI assumes that all processes remains active during program execution, each process initialize the MPI environment must by calling MPI_INIT before calling MPI interface. Since then to get the default size and number of process group by calling MPI_COMM_SIZE and MPI_COMM_RANK. The communication mode of each processing node could be P2P or broadcast and they send and receive messages by calling MPI_SEND and MPI_RECV function. When MPI ends, each process must terminate the MPI environment by calling MPI_FINANLIZE.

**BASIC IDEA**

The hierarchical nesting exact string matching algorithm (Fig. 4) is designed on the premise that the
This improved algorithm combines the advantages of the KMP algorithm and KR algorithm, makes some improvements to the KMP algorithm for calculating the next function on the condition of "large text - big Mode" string matching. When a mismatching occurs, the j-1 mode substring before matching will be saved and its hash value will be acquired in order to construct secondary KR text-pattern string. The matching efficiency of improved algorithm will be much higher than the KMP algorithm.

Parallelization of the hierarchical nested exact string matching algorithm: In the phase of pre-processing, according to the number of processing nodes, using the division method with overlapped anti-mismatched segmentation for specifying the data size and text dictionary randomly generated, hard the segmented even sub-string over to every processor in the cluster system. Each processor uses the improved matching algorithm in parallel to process the data (Fig. 5).

The improved parallel algorithm use the master-slave structure and the program is composed of two parts: the main process is used to input data from the problem domain and assigning tasks to n sub-processes by using anti-mismatched algorithm, also conduct the data communication and information exchange over the way site broadcast passing message. Finally incorporate the data results that processed by sub-processes. Slave process is used to access data, match data by making the use of the improved algorithm and output the matched initial position then return the result.
Implementation of algorithm: The improvement of the parallel algorithm uses the platform of MPI parallel interface. The pseudo-code of the algorithm is given below.

Algorithm 2: Exact string matching with n processors
Input: Amount of the processor: n, the data capacity of text dictionary, pattern string P[0, m]
Output: Matching the initial position i, 0 = i = n-m
Main function:
Begin
First the specification of the program parameters was needed including MPI header files, initialize the MPI environment using MPI_Init
Call MPI_COMM_RANK, establish the communication system MPI_COMM_WORLD
Identifies the processes Rank from 0 to n-1
When Rank = 0, the main process Master will be called and put process 0 as the main process.
Else, calling from the slave process Slave, put other processes as slave process
Call MPI_FINALIZE- MPI ending-
End

Main process master:
• Call MPI_COMM_RANK and establish communication system MPI_COMM_WORLD to get the total number of processes
• Specified size of text dictionary, generated text dictionary randomly
• Traverse each process
• Segment the text dictionary by calling the overlapped anti-mismatched algorithm missing text dictionary, text sent divided to each substring from the process, then send each divided text sub-string to each sub-process
• Establish message buffer to accept the results from any slave process and display the output
• When matching task ended it will inform all processes to exits

Slave process slave:
• Call MPI_STATUS and accept the randomly marked task from the main process, call MPI_RECV
• Check the identification of received message, if it is the end tag, then exit the task
• Check the identification of received message, if it is the new task, then call Algorithm 1, execute the corresponding matching calculation
• Call MPI_SEND, then return matching results

The communication overhead of this algorithm occurs only before the starting of calculation and after the end of the calculation of the data.

Preprocess is to overlapped block the target dictionary, to avoid the communication overhead bring from the re-allocation of the text sub-string, greatly reducing the execution time the multiple parallel match needed.

PERFORMANCE ANALYSIS

The improved parallel algorithm needs n+1 times operation to allocate the n pieces of text substring to n processing nodes. The worst match between the pattern string and text sub-string need:

\[ 2 \left( \frac{1}{n} + \frac{1}{p} - 1 \right) = 2 \left( \frac{1}{n} + 2p - 1 \right) \]

time's calculation, when the amount of pattern string is large, the mantissa can be rounded, then:

\[ \frac{n}{2} + 2p \]

times calculation is needed. The calculation of next [j] is related to the length of pattern string p, the algorithm complexity is \( O(p) \) and the required communication time to parallel transmission of data is \( O(n) \).

The data communication of this parallel algorithm occurs only before the start of the matching calculation (transfer target substring and pattern string to the child process node) and after the end of matching calculation (returns the matched results to the control node).
Fig. 6: Time's changes after the completion of the match of three kind of dictionaries scaled, processing nodes

Fig. 7: Speed-up ratio of parallel matching with the increase of three dictionaries scaled, processing node

Preprocessed link divide the text dictionary into n text sub-strings, and map the pattern string into unique couple of integer value stored in each child processing nodes, which decrease the workload and communication overhead of the child nodes.

TEST EXPERIMENT TRIAL

The test environment is PC clusters[8-9] with 100Mbps Ethernet connected. Choose DELL Pentium (R) Dual 2.20GHz, memory is 1.00GB as host processor to send the received tasks. Choose DELL Pentium(R) Dual 2.20GHz, memory is 512MB as slave node. Using MPI and C language to program. Test the completion time of the matching of the different magnitude data traffic under the condition of multi-node involvement (Fig. 6) and test the accelerated time ratio when the Processing nodes increase in turn (Fig. 7).

Given text dictionary whose scale is 500MB, 800MB, 1GB respectively, then generated the pattern string with determined size, it is observed that with the increase of the processing node the matching time is gradually decreased and gradually slowing down and communication time of the cluster system also gradually increased.

As shown in figure, the three-scale text dictionary with 500MB, 800MB, 1GB, as the increase of the processing node, the speed-up ratio gradually rise, however, with the growing amount of the processing node, the growth rate of the speed-up ratio gradually slow down, due to the increase of the communication overhead. When the text dictionary size is large, the time scale communication overhead occupied is relatively smaller and the gained speedup is better.

CONCLUSION

In the process of the research of parallel string matching, it is inevitable about how to process load-balancing problems appropriately and how to high efficiently matching and retrieval the large-scale data. This study discussed a master-slave stranded exact string matching layer structure parallel algorithm based on the MPI platform with PC cluster, which solves the problem of strong dynamic memory share and time-consuming about calculating the next value under the condition of "large text-big Mode" string matching, so that the algorithm can accomplish the matching in shorter time and proposed a kind of overlapped data parallel method for high efficiency of large-scaled data sets matching, which can effectively avoid the problem about lacking of allocation.

Certainly, the efficiency of the algorithm still has space for improvement. During the processing phase, the processed data could be placed into the global stack to further reduce the overhead of communication. Meanwhile, the Contexts-sensitive problem in the data dictionary as well as the judgment of redundant data remains further stud

REFERENCES

