Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 44 (2012) 7482-7485

ISSN: 2229-712X

Design of a frequent pattern mining based on systolic trees

R.P.S.Manikandan and V.Shanmugam
Sasurie College of Engineering, Vijayamangalam.

ARTICLE INFO

Article history:
Received: 9 September 2011;
Received in revised form:
14 November 2011;
Accepted: 23 November 2011;

Keywords
Systolic Tree,
Pattern Mining,
Database projection.

ABSTRACT

Frequent pattern mining algorithms are designed to find commonly occurring sets in databases. This class of algorithms is typically very memory intensive, leading to prohibitive runtimes on large databases. A class of reconfigurable architectures has been recently developed that have shown promise in accelerating some data mining applications. In this paper, I propose a new architecture for frequent pattern mining based on a systolic tree structure. The goal of this architecture is to mimic the internal memory layout of the original pattern mining software algorithm while achieving a higher throughput. We provide a detailed analysis of the area and performance requirements of our systolic tree-based architecture, and show that our reconfigurable platform is faster than the original software algorithm for mining long frequent patterns.

Introduction

The goal of frequent pattern (or frequent item set) mining is to determine which items in a transactional database commonly appear together. Given the size of typical modern databases, an exhaustive search is usually not feasible, making this a challenging computational problem. The FP-growth algorithm stores all transactions in the database as a tree using two scans. The FP-growth algorithm was originally designed from a software developer's perspective and uses recursion to traverse the tree and mine patterns. It is cumber some (and sometimes impossible) to implement recursive processing directly in hardware, as dynamic memory allocation typically requires some software management. For this reason, the dynamic data structures (such as linked lists and trees) which are widely used in software implementations are very rarely used in a direct hardware implementation. Consequently, it would be very difficult to directly translate this FP-growth algorithm into a hardware implementation.

I have previously introduced a reconfigurable systolic tree architecture for frequent pattern mining, and describe a prototype using a Field Programmable Gate Array (FPGA) platform.

The systolic tree is configured to store the support counts of the candidate patterns in a pipelined fashion as the database is scanned in, and is controlled by a simple software module that reads the support counts and makes pruning decisions. In this paper, we focus on improving the original scheme introduced in [2] by eliminating the counting nodes, and provide a new COUNT mode algorithm.

A new database projection approach which is suitable for mining a systolic tree is proposed and implemented. Based on the reconfigurable platform presented, the area requirement of FPGAs and software/hardware mining time are studied. We performed experiments over several benchmarks. The experimental results show that our FPGA-based approach can be several times faster than a software implementation of the FP-growth algorithm.

Tele:
E-mail addresses: rpsmanikandan@gmail.com , vsamsri@gmail.com

ID	Items	ID	Items
1	B,C,D	5	A,B,C
2	B,C	6	A,B,C
3	A,C,D	7	A,B,D
4	A,C,D		

Fig. 1. A simple transactional database

In VLSI terminology, a systolic tree is an arrangement of pipelined processing elements (PEs) in a multidimensional tree pattern. The goal of our architecture is to mimic the internal memory layout of the FP-growth algorithm while achieving a much higher throughput. The role of the systolic tree as mapped in FPGA hardware is then similar to the FP-tree as used in software. Given a transactional database, the relative positions of the elements in the systolic tree should be the same as in the FP-tree. To achieve this objective, the systolic tree in this work can be designed based on the following observations:

A control PE which corresponds to the root node in the FP-tree acts as the input and output interfaces to the systolic tree. The systolic tree construction algorithm starts from the root node upon receiving a new transaction. For two nodes which store any two items in a transaction, one node must be an ancestor of the other. If two transactions share a common prefix, the shared parts are merged using one prefix path. The degree of a node is equal to the number of transactions which share the common prefix. In the worst case, the degree of a node is equal to the number of frequent items N. Barring the use of dynamic partial reconfiguration, the PEs in hardware cannot be created and deleted dynamically as in software. If each PE is connected to N child PEs, the hardware structure and operation in each PE will be very complex. To avoid this hindrance, each PE in the systolic tree connects to its leftmost child. The other children connect to their leftmost siblings. To reach the rightmost child, the signal from the parent PE must travel through all the children on the left. In the FP-growth algorithm, the first item in each transaction is stored in a child of the root while the other items are stored in its descendants.

However, all PEs in the systolic tree operate in parallel and there is no "pointer" concept in the hardware implementation. Due to the intrinsic characteristics of the tree, the operation of a transaction usually starts from the root. Thus, the control PE acts as the input/output interface of the systolic tree. Each clock cycle, an item is transferred to the control PE which may forward the item to its children.

Definition 1 (Systolic Tree). A systolic tree is a tree structure which consists of the following PEs:

Control PE: The root PE of the systolic tree does not contain any item. Any input/output data of the systolic tree must go through it first. One of its interfaces connects to its leftmost child.

General PE: All other PEs are general PEs. Each general PE has one bidirectional interface connecting to its parent. The general PE which has children has one interface connecting to its leftmost child. Those general PEs which have siblings may have an interface connecting to its leftmost sibling. The general PE may contain an item and the support count of the stored item.

Each PE has a level associated with it. The control PE is at level 0. The level of a general PE is equal to its distance to the control PE. The children of a PE has the same level.

Systolic Tree Creation

The design principle of the WRITE mode algorithm is that the built-up systolic tree should have a similar layout with the FP-tree given the same transactional database. The ith item in a transaction is mapped to the ith level in the systolic tree. For any two PEs in the path of the same transaction, the PE in the ith level is the ancestor of the PE in the i þ 1th or higher level. A PE is never in the same path of a transaction with its siblings. Suppose the transactional database has N frequent items. The number of PEs in the first level is at most N. The depth of the systolic tree is at most N. Suppose all items in Fig. 1 are frequent. The first items in all transactions only include items A and B. Therefore, we only need two PEs in the first level of the systolic tree. However, counting the number of items in each level may impair the overall pattern mining performance. Thus, the values of K and W are usually set to be equal to N. The last item of a transaction may not be put into the leaf of the systolic tree if the number of items in the transaction is less than N. If two transactions share the same prefix, they will share a path in the systolic tree. In summary, the design intuition behind the WRITE mode algorithm is that the path an item travels through the FP-tree is the same as the path it travels in the systolic tree.

The WRITE mode algorithm is presented in Algorithm 1. The same algorithm runs in each PE of the systolic tree in every clock cycle. That is, the inner hardware structure of each PE is the same. Initially all PEs are empty. An item is loaded into the control PE each clock cycle which in turn transfers each item into the general PEs. After all items in a transaction are sent to the systolic tree, a control signal that states the termination of an old transaction and the start of a new one is sent to the control PE. The signal will be broadcast to all PEs which reinitialize themselves for the next transaction.

The initialization includes resetting match and Inpath flags in the first line of Algorithm 1. The input of the algorithm is an item it. The match flag is set when the item in the PE matches it. The Inpath flag is not set when the PE does not contain any item from the current transaction.

Upon receiving a new item it, the three if statements in Algorithm 1 are evaluated sequentially. The two switches match and InPath are used to make sure that the latter items in a transaction will follow the path of the former items in the same transaction. The first if statement allocates a PE for the incoming item it if it appears in the current path for the first time. In this case, the transaction represented by the items contained in the PEs from the root to the current one is a new transaction which has never been put into the systolic tree before. The second if statement is executed if the current item matches the item in the current PE. In this case, the transaction represented by the items contained in the PEs from the root to the current one is a transaction which has been put into the systolic tree before.

Pattern Mining Using Systolic trees

To mine frequent patterns in the systolic tree, a collaborating hardware/software platform is required. The software sends a candidate pattern to the systolic tree. After some clock cycles, the systolic tree sends the support count of the candidate pattern back to the software. The software compares the support count with the support threshold and decides whether the candidate pattern is frequent or not. After all candidate patterns are checked with the support threshold in software, the pattern mining is done. The approach to get the support count of a candidate pattern is called candidate item set (pattern) matching. The platform architecture of the software and hardware co-design will be introduced.

Candidate Item Set Matching

The main principle of matching is that any path containing the queried candidate item set will be reported to the control PE. Note that such a path may contain more items than the queried item set. Before introducing item set matching, we examine some useful properties of the systolic tree which will facilitate frequent pattern mining. As mentioned in previous sections, each frequent item is assigned a sequence number. The items in each transaction enter the systolic tree in an increasing order in both the WRITE mode algorithm and the SCAN mode algorithm.

Candidate Item Set Count Computation

According to Property 1 of the systolic tree, there is only one path tracing back from any PE to the control PE since each PE has a unique parent. Once all items in a candidate item set are sent to the systolic tree, a control signal signifying the COUNT mode is broadcast to the whole systolic tree. It shows that the first child's input interface is always connected to its parent while others accept input from the sibling in WRITE and SCAN mode. After a candidate frequent item set is delivered into the systolic tree, the PEs report the support count of the candidate item set to its unique parent. The PE which is not directly connected to its parent sends its count to the left sibling. .The parent PE collects the support counts reported by the children PEs and sends them to its own parent direction. The COUNT mode algorithm in each PE is given in Algorithm 3, where the support counts are transferred to each PE's parent in a pipelined fashion. The inputs of the algorithm are two count values sent by a PE's sibling and child, respectively. The NMyself variable is the number of the locally stored item. The Sent flag is set when the local number has been reported to the parent. Only the PEs with the IsLeaf flag set can report its local value, while other PEs deliver the count values sent by the child and sibling to their parents. The control PE adds up all count values and sends it as an output signal.

Tree Scaling by Database Projection

It is unreasonable to assume that a tree-based representation can fit in the available hardware resources (either memory or logic) for any arbitrary database. In the case that memory or

logic is not large enough to hold the whole tree, the database must be divided into multiple smaller databases with fewer frequent items. Without the technique of database projection, the brute-force candidate item set matching will take an intolerable amount of time when the number of frequent items is large.

Introduction to Database Projection

To use the systolic tree to mine frequent item sets, the original database is projected into subdatabases. Each of the projected database has no more than N ¼ minðK; WÞ frequent items and is guaranteed to fit into the FPGAs. Let's illustrate the database projection with an example. Suppose the FPGA logic can at most hold a systolic tree with two frequent items. The database in Fig. 1 has four frequent items and should be projected into subdatabases each of which has at most two frequent items. The frequent items are usually sorted in frequency-decreasing order, which introduces a dense tree structure. For illustrative purpose, we arrange the frequent items in an alphabetic order, i.e., A; B; C; D. Starting at frequent item A, the set of transactions that contain A are collected as an A-projected database. Since there are three frequent items B; C; D in the A-projected database, it should be further projected into two subdatabases. One is an AB-projected database. The transactions in the AB-projected database are obtained by removing B from those transactions containing B in the A-projected database. The other is an A_3-projected database which is composed of those transactions in the A-projected database after removing B. Since the transactions in both of them only contain C and D, the projection process terminates. The projected databases which will be put onto an FPGA are shown as dotted leaves.

Notice the difference between the A-projected database and the A_3-projected database. The former data-base contains all the transactions starting with A while the latter contains only those transactions starting with A but not including item B.

Experimental Evaluation

Area Requirements

In our VHDL implementation, different sizes of systolic trees are generated automatically by specifying the depth and degree parameters. We placed and routed various configurations of this implementation using Xilinx ISE 9.1.03i. The targeted device is a Xilinx Virtex-5 XC5VLX330 FPGA with package ff1760 and -2 speed grade. From the place and route report, each node requires 50 LUTs. Approximately 25 percent of the available slices are used when both K and W are four. When both K and W are five, the slice usage is more than 100 percent.

Performance Comparison

For performing a fair comparison, we selected as our target environment the XtremeData XD2000i, which is a complete platform to explore the benefits of FPGA coprocessing with traditional microprocessor computing systems. The XD2000i development system comprises of a Dual Xeon motherboard with one Intel Xeon processor and two Stratix III EP3SE260 FPGAs in the other socket, running the CentOS Linux operating system.

In our simulated environment, the systolic tree of size four was integrated with other hardware IP components provided by XtremeData in one of the FPGAs. The Intel

CPU executes Algorithm 5, which was implemented in C++. For each frequent item set ₃, a projected tree is generated as a set of pointers pointing to the paths in the original FP-tree. The algorithm reads the transactions pointed by those pointers and sends them to the systolic tree in FPGA. The sending operation returns immediately. The FPGA hardware receives the transactions and creates the systolic tree using Algorithm 1. Thereafter, the candidate item sets are generated and matched by hard-ware without software intervention in a pipelined style. Algorithms 2 and 3 are executed in each node of the systolic tree. After all frequent item sets are collected, they are sent back to the software. After receiving the frequent item sets from the hardware, the software combines them with ₃.

During this process, the original database is projected sequentially which is similar to partition projection men-tioned in fig. The experimental results presented below are based on this sequential mode. Reconfigurable develop-ment systems with multiple FPGAs integrated in the same board would not benefit from this model since the running time of FPGAs in this case is smaller than the running time to generate a projected database. Parallel projection, which scans each transaction in the original database and projects it into multiple projected databases in parallel can fully utilize the multi-FPGA system. Designing an efficient parallel projection software algorithm and applying it in a parallel FPGA implementation is a planned avenue of future work. We can expect that the parallel FPGA implementation will be much faster than the sequential model used in this paper.

The experiments are performed on real data sets described in and from the KDD Cup 2000 data. Table 1 characterizes the data sets in terms of the number of distinct items and transactions, the maximum and mini-mum size of the transactions. The performance measure was the execution time of the systems on the benchmarks with different support thresholds. The execution time only includes the time for disk reading and memory I/O, but excludes the disk writing time. The original software FP-growth algorithm which is implemented in C++ is from [13]. In order to have a fair comparison, the software FP-growth algorithm also runs on our target Xtreme Data XD2000i platform without using the FPGA hardware.

The systolic tree algorithm is almost two times faster than FP-growth for the chess and BMS-WebView-2 data sets. For the kosarak data set, the mining time of the systolic tree is larger than FP-growth. The mining time for FP-growth grows larger than the systolic tree algorithm with the decrease of support threshold in BMS-POS.

The advantage of the systolic tree over FP-growth becomes dramatic with long frequent patterns, which is challenging to the algorithms that mine the complete set of frequent patterns. In retrospect, the systolic tree algorithm removes the most frequent items and mines them in the projected databases.

Compared with the original FP-growth algo-rithm, the systolic tree algorithm reduces the runtime by mining the dense part of the FP-tree in hardware and the sparse part in software simultaneously. However, it introduces the overhead of database projection. If the overhead is not amortized by the runtime reduction, the systolic tree algorithm is slower than the original FP-growth algorithm. This is illustrated in the kosarak benchmark.

One can expect that the mining time will decrease with more frequent patterns mined in hardware when the size of the systolic tree is fixed. , the systolic tree algorithm removes the most frequent items and mines them in the projected databases. Hence more patterns will be mined in the systolic tree. It will be more efficient than the FP algorithm that can be used for the mining. Also the systolic tree algorithm reduces the runtime.

Conclusion

A new systolic tree-based approach to mine frequent item sets is proposed and evaluated. A transactional database must be projected into smaller ones each of which can be mined in hardware efficiently. A high performance projection algorithm which fully utilizes the advantage of FP-growth is proposed and implemented. It reduces the mining time by partitioning the tree into dense and sparse parts and sending the dense tree to the hardware. The algorithm was implemented on an XtremeData XD2000i high performance reconfigurable platform. Based on the experimental results on several benchmarks, the mining speed of the systolic tree was several times faster than the FP-tree for long frequent patterns. Improving the mining efficiency on sparse patterns will be included in our future work.

References

[1]J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," Data Mining and Knowledge Discovery, vol. 8, no. 1, 53-87, Jan. 2004.

[2]S. Sun and J. Zambreno, "Mining Association Rules with Systolic Trees," Proc. Int'l Conf. Field-Programmable Logic and Applications (FPL '08), Sept. 2008.

[3] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An FPGA Implementation of Decision Tree Classifi-cation," Proc. Conf. Design, Automation, and Test in Europe (DATE), 189-194, Apr. 2007.

[4]Z. Baker and V. Prasanna, "Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs," Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM), pp. 3-12, Apr. 2005.

[5]Y.-H. Wen, J.-W. Huang, and M.-S. Chen, "Hardware-Enhanced Association Rule Mining with Hashing and Pipelining," IEEE Trans. Knowledge and Data Eng., vol. 20, no. 6, pp. 784-795, June 2008.

[6]A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y.-K. Chen, and P. Dubey, "Cache-Conscious Frequent Pattern Mining on a Modern Processor," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 577-588, 2005.

[7]R. Bayardo, B. Goethals, and M. Zaki, eds., Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI), Nov. 2004.

[8]T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient Mining Algorithms for Frequent Closed Maximal Itemsets," Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI), 2004.

Table 1
Benchmark Information

Benchmark	Items	Num. Trans.	Max Trans.	Min Trans.	Total Items
chess	75	3196	37	37	118252
BMS-WebView-2	3340	77512	161	1	358278
connect	129	67557	43	43	2904951
BMS-POS	1657	515597	164	1	3367020
pumsb	7117	49064	74	74	3629404
kosarak	41270	990002	2498	1	8019015