**HO CHI MINH UNIVERSITY OF TECHNOLOGY**

---------------o0o---------------

****

**Member: Nguyễn Gia Phúc – 1450116**

**Ho Chi Minh city, 27/12/2018**

**Parallel Computing**

**Write a program to solve Association Rules algorithm use multi-thread on Xeon Phi with Offload and OpenMP mechanism**

Table of Contents

[I. Theory: 2](#_Toc533672319)

[1. Association Rules algorithm: 2](#_Toc533672320)

[1.1 What is Association Rules algorithm? 2](#_Toc533672321)

[1.2 Examples: 3](#_Toc533672322)

[2. OpenMP mechanism: 5](#_Toc533672323)

[2.1 What is OpenMP mechanism? 5](#_Toc533672324)

[2.2 Compiling and running OpenMP code 6](#_Toc533672325)

[2.3 Example (C program): 6](#_Toc533672326)

[2.4 Coding on Xeon Phi 7](#_Toc533672327)

[3. Offload mechanism: 7](#_Toc533672328)

[II. Implementation: 8](#_Toc533672329)

[1. Write program (code): 8](#_Toc533672330)

[2. Speedup: 9](#_Toc533672331)

[3. Testing speed up: 9](#_Toc533672332)

[III. Summary: 15](#_Toc533672333)

[IV. References 16](#_Toc533672334)

# Theory:

## Association Rules algorithm:

### 1.1 What is Association Rules algorithm?

Association rule learning was proposed by Agrawal and Srikant in 1994. Association Rules is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation or IP addresses). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing). Each transaction is seen as a set of items (an itemset). Given a threshold {\displaystyle C} C, the Association Rules algorithm identifies the item sets which are subsets of at least C transactions in the database.

Association Rules uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length k-1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

### 1.2 Examples:

**Example 1:**

Consider the following database, where each row is a transaction and each cell is an individual item of the transaction:

|  |  |  |
| --- | --- | --- |
| alpha | beta | epsilon |
| alpha | beta | theta |
| alpha | beta | epsilon |
| alpha | beta | theta |

The association rules that can be determined from this database are the following:

1. 100% of sets with alpha also contain beta
2. 50% of sets with alpha, beta also have epsilon
3. 50% of sets with alpha, beta also have theta

we can also illustrate this through a variety of examples.

**Example 2:**

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item: each item, such as "butter" or "bread", is identified by a numerical SKU. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together.

Let the database of transactions consist of following itemsets:

|  |
| --- |
| **Itemsets** |
| {1,2,3,4} |
| {1,2,4} |
| {1,2} |
| {2,3,4} |
| {2,3} |
| {3,4} |
| {2,4} |

We will use Apriori to determine the frequent item sets of this database. To do this, we will say that an item set is frequent if it appears in at least 3 transactions of the database: the value 3 is the support threshold.

The first step of Apriori is to count up the number of occurrences, called the support, of each member item separately. By scanning the database for the first time, we obtain the following result

|  |  |
| --- | --- |
| **Item** | **Support** |
| {1} | 3 |
| {2} | 6 |
| {3} | 4 |
| {4} | 5 |

All the itemsets of size 1 have a support of at least 3, so they are all frequent.

The next step is to generate a list of all pairs of the frequent items.

For example, regarding the pair {1,2}: the first table of Example 2 shows items 1 and 2 appearing together in three of the itemsets; therefore, we say item {1,2} has support of three.

|  |  |
| --- | --- |
| **Item** | **Support** |
| {1,2} | 3 |
| {1,3} | 1 |
| {1,4} | 2 |
| {2,3} | 3 |
| {2,4} | 4 |
| {3,4} | 3 |

The pairs {1,2}, {2,3}, {2,4}, and {3,4} all meet or exceed the minimum support of 3, so they are frequent. The pairs {1,3} and {1,4} are not. Now, because {1,3} and {1,4} are not frequent, any larger set which contains {1,3} or {1,4} cannot be frequent. In this way, we can prune sets: we will now look for frequent triples in the database, but we can already exclude all the triples that contain one of these two pairs:

|  |  |
| --- | --- |
| **Item** | **Support** |
| {2,3,4} | 2 |

in the example, there are no frequent triplets. {2,3,4} is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold.

We have thus determined the frequent sets of items in the database, and illustrated how some items were not counted because one of their subsets was already known to be below the threshold.

## OpenMP mechanism:

### What is OpenMP mechanism?

OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer.

An application built with the hybrid model of parallel programming can run on a computer cluster using both OpenMP and Message Passing Interface (MPI), such that OpenMP is used for parallelism within a (multi-core) node while MPI is used for parallelism between nodes. There have also been efforts to run OpenMP on software distributed shared memory systems, to translate OpenMP into MPI and to extend OpenMP for non-shared memory systems.

OpenMP supports C, C++ and Fortran.

The OpenMP parts in the code are specified using #pragmas

### Compiling and running OpenMP code

**Complie**: gcc -fopenmp <filename.c> -o <object filename to excute>

G++ -fopenmp <filename.cpp> -o <object filename to excute>

**Run**: ./ <object filename to excute> <input.txt>(if any) <output.txt>(if any)

### Example (C program):

Display "Hello, world." using multiple threads.

#include < stdio.h >

int main(void)

{

#pragma omp parallel

{

printf("Hello, world.\n");

}

return 0;

}

Use flag -fopenmp to compile using gcc:

$ gcc -fopenmp hello.c -o hello

Output on a computer with two cores, and thus two threads:

Hello, world.

Hello, world.

On hpcc server, we have a CPU with 48 cores, so we get 48 “Hello, world.”.

### Coding on Xeon Phi

Put the parallel code inside the format below to run parallel on Xeon Phi

#pragma offload target(mic: i) //i:0,1,2,… is the id of Xeon Phi card.

{

//to do your code here

}

## Offload mechanism:

Application runs on the host, but some parts of code and date are moved (“offloaded”) the coprocessor.

* No additional arguments if compiled with an Intel compiler.
* Run application on host as a regular application.
* Code inside of #pragma offload is offloaded automatically.
* Console output on Intel Xeon Phi coprocessor is buffered and mirrored to the host console.
* If coprocessor is not installed, code inside #pragma offload runs on the host system

**Example:**

#include <stdio.h>

int main(int argc, char \* argv[] ) {

printf("Hello World from host!\n");

#pragma offload target(mic)

{

printf("Hello World from coprocessor!\n"); fflush(0);

}

printf("Bye\n");

}

# Implementation:

## Write program (code):

All code about the Association Rules algorithm and examples in this report, we have already pushed on HPCC server. Go to HPCC server with: “student@10.1.6.122” to get full code of our report.

By the other way, the easy way to get our code to follow below github link:

<https://github.com/giaphuc427/ParrallelAssignment>

## Speedup:

Speedup is a number that measures the relative performance of two systems processing the same problem. More technically, it is the improvement in speed of execution of a task execute.

The notion of speedup was established by Amdahl's law, which was particularly focused on parallel processing. However, speedup can be used more generally to show the effect on performance after any resource enhancement. d on two similar architectures with different resources.

## Testing speed up:

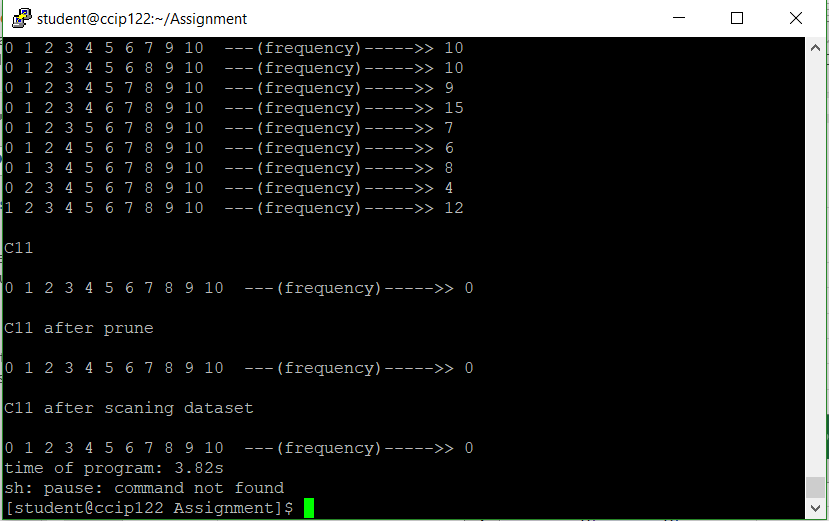
For testing, we write the C++ code to generate automatically 5 files:

* File 1: 1000 transactions (lines), each transaction has randomly 1->10 number and each number has the range 0->10.
* File 2: 3000 transactions.
* File 3: 5000 transactions.
* File 4: 7000 transactions.
* File 5: 9000 transactions.

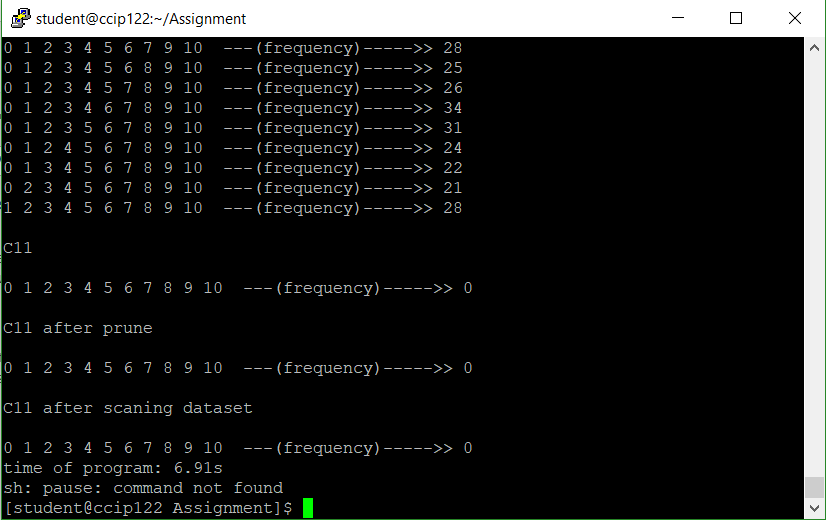
**Sequence code:**

When we run the code sequentially, the time goes up based on the number of transactions. It is the linear of time.

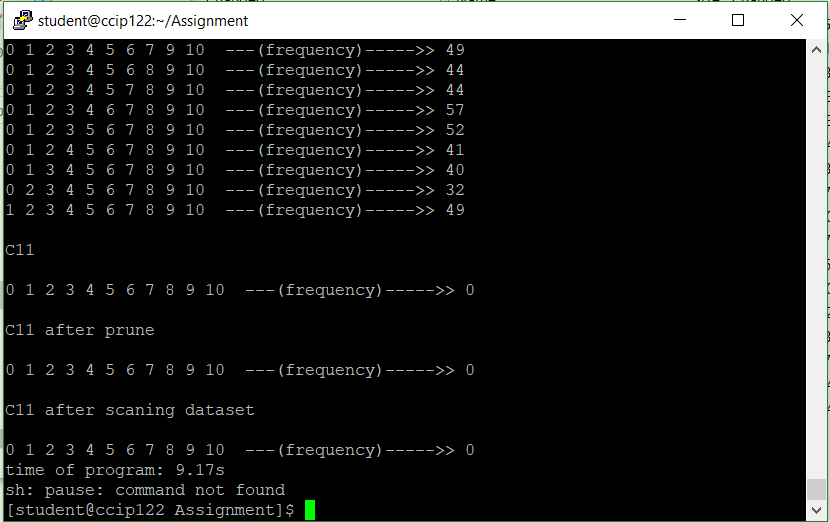
* File 1: 1000 transactions.



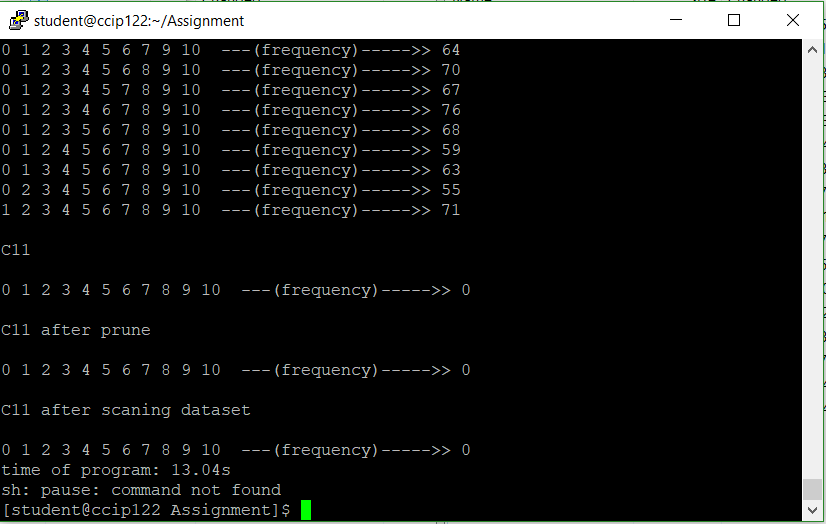
* File 2: 3000 transactions.



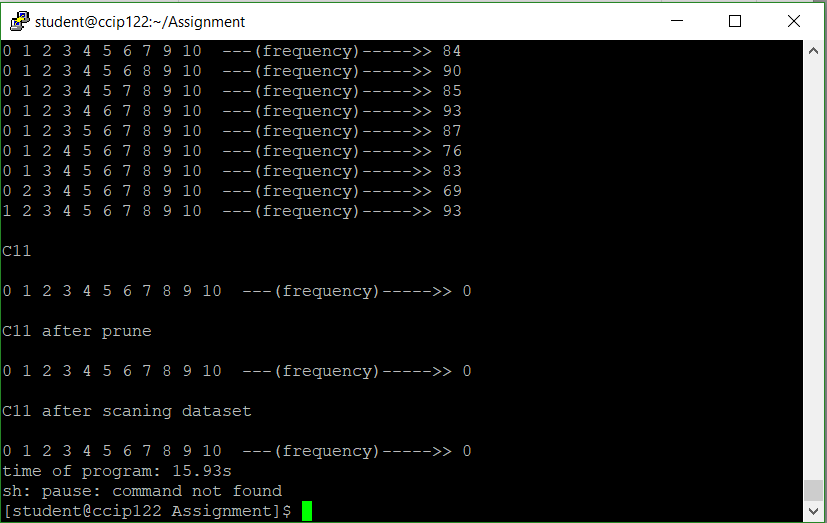
* File 3: 5000 transactions.



* File 4: 7000 transactions.



* File 5: 9000 transactions.



**Parallel code with 1 Xeon Phi card:**

When using the Offload and OpenMP mechanism with 1 Xeon Phi card, we realize that the time decreased a part. This is also our target before running.

**Parallel code with 2 Xeon Phi card:**

When we config code, to try to speed up the algorithm with 2 Xeon Phi card, the result that is not expectable. The figure is fluctuated the same as when we use 1 Xeon Phi card.

# Summary:

Through the assignment of Write a program to solve Association Rules algorithm use multi-thread on Xeon Phi with Offload and OpenMP mechanism, we conclude that the speed up of the program is not totally based on the hardware. It looks like when we run the program gets the peak of speed up, we only improve the speed up by the code (algorithm), cannot improve by the hardware (core, processor).

# References

*https://en.wikipedia.org/wiki/Association\_rule\_learning*. (n.d.).

*https://en.wikipedia.org/wiki/OpenMP*. (n.d.).

*https://en.wikipedia.org/wiki/Speedup*. (n.d.).

*https://www.saedsayad.com/association\_rules.htm*. (n.d.).

*https://www.youtube.com/watch?v=AgAI5lDCyuA*. (n.d.).