Multithreading & Sequence Validation Algorithm: Solving Cryptarithmetic Problems

Orlando Díaz Muñiz
Computer Engineering
Alfredo Cruz Triana, Ph.D.
Electrical and Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— Cryptarithmetic problems are mathematical equations of unknown numbers that are represented by letters. The goal is to identify the number that represents each letter. There are algorithms that provide a simple way to solve such problems which has a big search space even for quite small problems. We propose a solution to this problem with sequence validation algorithm in parallel with optimization using multithreading technique. We have develop a program to implement this algorithm using C Sharp, as programing language, and showed that the algorithm reaches a solution, applying sequence validation and multithreading techniques, faster than using single thread.*

Key Terms *— Cryptarithmetic, Sequence Validation, Verbal Arithmetic*

INTRODUCTION

Cryptarithmetic problems are puzzles consisting of a mathematical equation of unknown numbers that are represented by letters. The goal is to identify the number that represents each letter. These mathematical equations are usually arithmetic operations. This type of problem was popularized during the 1930s is the Sphinx, a Belgian journal of recreational mathematics [1]. One of the well known Cryptarithmetic problems which published in the July 1924 issue of Strand Magazine by Henry Dudeney [2] is show in Figure 1. The solution to this problem is S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, and Y = 2.

```
   S E N D
 + M O R E
 ---------
 M O N E Y
```

Figure 1
Cryptarithmetic Problem Example

Genetic Algorithms (GAs) are search algorithms inspired by genetics and natural selection. Parallel Genetic Algorithms (PGAs) are parallel implementations of GAs which can provide considerable gains in terms of performance and scalability [3]. The most important advantage of PGAs is that in many cases they provide better performance than single population based algorithms, even when the parallelism is simulated on conventional machines [4]. Existing GA and PGA implementations were compared with the proposed algorithm results. Constraints of cryptarithmetic problems are as follow:

- Same number cannot be assigned to different letters.
- The first letter of each string cannot be assigned to zero.
- Number assigned to each letter must satisfy the arithmetic operation.

Solving cryptarithmetic problem by hand generally involves a combination of deductions and extensive tests of possibilities. Solving cryptarithmetic problems programmatically involve a lot of iterations and a big search space. The proposed algorithm provides a solution to this problem by using sequence validation method in parallel with optimization using multithreading technique.

FORMULATION OF THE ALGORITHM

The proposed algorithm provides the following elements:

- Solve problems from 5 to 10 distinct letters in an acceptable execution time.
- Distribute the work load in 1 to 9 threads.

- Find all possible solutions to the given problem.

The following are brief descriptions related to formulation of the proposed algorithm.

Calculating the Sequence of the Given Problem

To find the sequence is necessary to assign a number to each different letter of the given problem. This will generate a sequence of numbers from 0 to n-1, where n is the total letters of the given problem.

Determining All Possible Solutions

The total generators required are given by the total different letters in the problem. Each generator assigns a single number at a time. Therefore the generator $G_n(x)$ must assign a number (x) from 0 to 9 or 1 to 9; in order, before the generator $G_{n+1}(x)$ where n is a number between zero and the total distinct letters of the given problem.

Applying Sequence Validation Method

Each number generated is substituted into the equation and then the sequence is calculated. The calculated sequence is compared, from the index 0 to the n index, with the sequence of the given problem. If it is different, then $G_{n+1}(x)$ assignment is cancelled, and proceed with the next assignment in $G_n(x)$. Otherwise if the sequences are equivalent then $G_{n+1}(x)$ assignment proceed and the process is repeated.

Verification Method

If all numbers generators assign a number and the current calculated sequence match the sequence of the given problem then the numbers represented by the valid sequence are substituted in the given equation. If it satisfies the equation then a solution has been found for the given problem.

Applying Multithreading Techniques

This procedure can be separated into 2, 3, 4, 5, 6, 7, 8 or 9 simultaneous tasks to reduce the elapse or solution time. If it is divided into 2 tasks for example, then the first task will find all possible solutions to the problem starting with number 1, 2, and 3. The second task will find all possible solutions starting with number 4, 5 or 6. Finally, the third task will find all possible solutions starting with number 7, 8 or 9.

ALGORITHM EXECUTION EXAMPLE

The algorithm starts by creating a Default Sequence (DS) based on the given problem. Let say that we have the following cryptarithmetic problem: SEND + MORE = MONEY. Then the DS is calculated by assigning a number from left to right to each letter starting at 0. Therefore the DS for this problem is {0,1,2,3,4,5,6,7,8,9,10,11,12}. The relation between each letter and the sequence number is set as shown in Figure 2.

S	E	N	D	+	M	O	R	E	=	M	O	N	E	Y
0	1	2	3		4	5	6	7		8	9	10	11	12

Figure 2
Relation Between Each Letter and Sequence Number

The next step is to assign the same sequence number for repeated letters. For this step only the first occurrence of each letter is considered. Therefore the Default Sequence First Occurrence (DSFO) will be set as shown in Figure 3.

S	E	N	D	M	O	R	Y
0	1	2	3	4	5	6	12

Figure 3
Relation Between Each Letter and Default Sequence First Occurrence

The DSFO is used to determine the Calculated Sequence (CS) by substituting the DSFO numbers into the cryptarithmetic problem. Figure 4 illustrates the relation between the DSFO and DS with the resultant CS.

The DSFO sequence is used only to obtain the CS. The DS is used to obtain the CS but is also used to populate the index matrix discussed below. In the iteration process the CS is used to determine the validity of the possible solution calculated. The use of CS in the iteration process reduce the total number of iterations required to find a valid solution to the given problem.

	DSFO	S	E	N	D	M	O	R	Y	CS
		0	1	2	3	4	5	6	12	
Problem	S	0								0
	E		1							1
	N			2						2
	D				3					3
	M					4				4
	O						5			5
	R							6		6
	E		1							1
	M					4				4
	O						5			5
	N			2						2
	E		1							1
	Y								12	12

Figure 4
Relation Between Default Sequence First Occurrence and Calculated Sequence

The CS is determined by substituting all values found from DSFO into the cryptarithmetic problem as showed in Figure 4. Therefore, the CS for this problem is giving by {0, 1, 2, 3 ,4 ,5 ,6 ,1 ,4 ,5 ,2 ,1 ,12 }as shown in Figure 5.

S E N D M O R E M O N E Y
0 1 2 3 + 4 5 6 1 = 4 5 2 1 12

Figure 5
Relation Between Each Letter and Calculated Sequence

Obtaining Index Matrix

The total number of occurrences per Variable is necessary to create and index matrix. An index matrix is required to store the index or position of each letter in the equation. The index matrix dimension is defined by the total distinct letters and the maximum occurrence of the letters. Table 1 shows the occurrence per each distinct letters.

Table 1
Index Matrix

Distinct Variables	Times Repeated
S	No repeated
E	Repeated 3 times
N	Repeated 2 times
D	No repeated
M	Repeated 2 times
O	Repeated 2 times
R	No repeated
Y	No repeated

The letter E has the maximum number of occurrences because is repeated more times than the other letters. The letter E is repeated 3 times, therefore the dimension of the index matrix is defined as 8 x 3 where 8 is the total distinct letters and 3 is maximum occurrence per letter. The occurrence per variable is defined as shown in Figure 6.

S E N D M O R Y
1 3 2 1 2 2 1 1

Figure 6
Relation Between Each Letter and Occurrence per Variable

The DS is used to populate the index matrix. Figure 7 is a representation of the populated index matrix. The letters appears in the same order of position index. Each index represents a unique position in the problem.

S	0	-	-
E	1	7	11
N	2	10	-
D	3	-	-
M	4	8	-
O	5	9	-
R	6	-	-
Y	12	-	-

Figure 7
Position Index per Variable

Iteration Process

After determine the index matrix the iteration process starts by assigning a number to each letter from 0 to 9 or 1 to 9. Table 2 and 3 illustrates the first ten iterations followed by the algorithm to solve the problem using both methods; applying the sequence validation method and without applying the sequence validation method. Both methods were executed separately.

Iterating Without Sequence Validation Method

A single number assignment is performed per iteration as show in Table 2. When all letters has a number assigned then those numbers are substituted into the equation, using the index matrix to obtain a possible solution. The sequence of the possible solution is calculated and compared with the sequence of the given problem previously calculated. If both sequences are similar then the mathematical operation is executed. A solution is found if the given numbers satisfy the equation. The process ends after all possible solutions are verified.

Table 2
First 10 Iterations Without Sequence Validation

Iteration #	Execution without applying the sequence validation								
	Number Generators								Possible Solution
	S	E	N	D	M	O	R	Y	S,E,N,D,M,O,R,E,M,O,N,E,Y
1	1								1 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
2	1	0							2 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
3	1	0	0						3 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
4	1	0	0	0					4 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
5	1	0	0	0	0				5 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
6	1	0	0	0	0	0			6 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
7	1	0	0	0	0	0	0		7 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
8	1	0	0	0	0	0	0	0	8 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
9	1	0	0	0	0	0	0	1	8 - 1,0,0,0,0,0,0,0,0,0,0,0,1,
10	1	0	0	0	0	0	0	2	8 - 1,0,0,0,0,0,0,0,0,0,0,0,2,

Iterating With Sequence Validation Method

This process is very similar to the method mentioned above. The only difference is when each letters has a number assigned then those numbers are substituted into the equation using the index matrix to obtain an incomplete or complete possible solution.

The sequence of the possible solution is calculated and compared from the beginning until the index of the first occurrence related to the current letter assigned. If the sequence related to the original problem does not match the current calculated sequence until the index specified then; the next number generator iteration is cancelled as show in Table 3. The next number is subsequently considered in the same number generator and the process repeated until an acceptable sequence is found or after all possible solutions are verified. If an acceptable sequence exists then it is subsequently validated by performing the mathematical operation of the equation. A solution is found if the given numbers satisfy the equation. The process ends after all possible solutions are verified.

The process without sequence validation involves a lot of iterations, which were reduced by the sequence validation method. Tables 2 and 3 clearly show the reduction in the search space from iteration 4 and above. Without sequence validation the number generator associated with letters D, M, O, R, Y must complete all iterations and its respective assignments from 1 to 9 before assign a number to letter N. On the other hand, the sequence validation method eliminates all unnecessary calculations and is capable to assign number to letter N in just the fourth iteration as show in Table 3.

Table 3
First 10 Iterations With Sequence Validation

Iteration #	Execution applying the sequence validation								
	Number Generators								Possible Solution
	S	E	N	D	M	O	R	Y	S,E,N,D,M,O,R,E,M,O,N,E,Y
1	1								1 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
2	1	0							2 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
3	1	0	0						3 - 1,0,0,0,0,0,0,0,0,0,0,0,0,
4	1	0	1						3 - 1,0,1,0,0,0,0,0,0,0,1,0,0,
5	1	0	2						3 - 1,0,2,0,0,0,0,0,0,0,2,0,0,
6	1	0	2	0					4 - 1,0,2,0,0,0,0,0,0,0,2,0,0,
7	1	0	2	1					4 - 1,0,2,1,0,0,0,0,0,0,2,0,0,
8	1	0	2	2					4 - 1,0,2,2,0,0,0,0,0,0,2,0,0,
9	1	0	2	3					4 - 1,0,2,3,0,0,0,0,0,0,2,0,0,
10	1	0	2	3	0				5 - 1,0,2,3,0,0,0,0,0,0,2,0,0,

Applying Multithreading Techniques

In simple words a thread is the smallest unit of processing that can be managed by an operating system. Therefore, is called multithreading, when dealing with more than one thread simultaneously.

Generally multithreading is use in methods that perform intensive calculations. Such methods can execute faster on a multiprocessor or multicore computer if the workload is shared among two or more threads. Multithreading will not always speed up your application, it can even slow it down if used excessively or inappropriately [5].

In order to apply multithreading techniques, the iteration process is separated into tasks. Each thread must take charge of a task. The user determines the number of threads used to solve the problem. If for example, 3 threads are used to solve the problem then thread 1 iterates to find possible solutions where the first digit begins with a number within 1, 2, and 3. The thread 2 iterates to find possible solutions where the first digit begins with a number within 4, 5, and 6. The thread 3 iterates to find possible solutions where the first digit begins with a number within 7, 8, and 9.

Assuming no sequence validation is applied then the first and last iteration for thread 1, 2, and 3 will be from {1,_,_,_,_,_,_,_} to {3,9,9,9,9,9,9,9}, {4,_,_,_,_,_,_,_} to {6,9,9,9,9,9,9,9}, and {7,_,_,_,_,_,_,_} to {9,9,9,9,9,9,9,9} respectively.

Graphic User Interface

A graphic user interface (GUI) was developed to provide the inputs required and display results. Figure 8 is a screenshot of the application with results related to SEND + MORE = MONEY using 2 threads. Below figure you can find the description of each item identified with a number from 1 to 18.

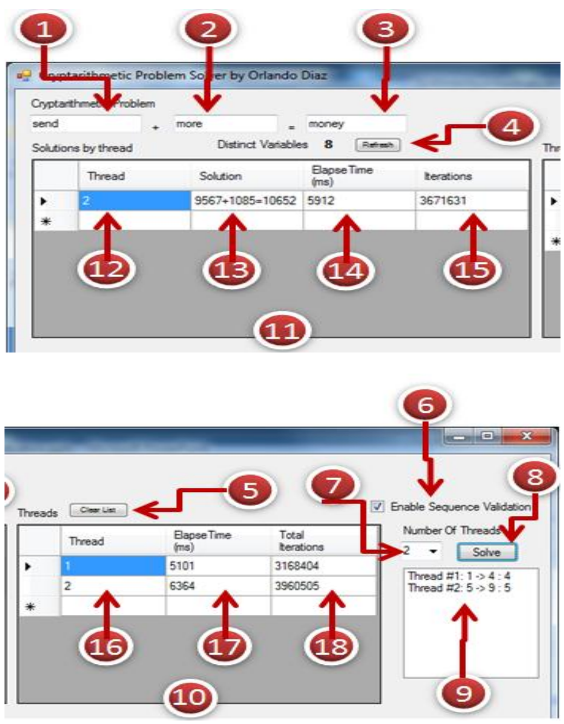

Figure 8
Graphic User Interface

- 1: First set of letters "**SEND**" representing the first number in the equation.
- 2: Second set of letters "**MORE**" representing the second number in the equation.
- 3: Third set of letters "**MONEY**" representing the result of the equation.
- 4: Calculates the total distinct letters in 1, 2, and 3. The results are displayed at the left side of the button. It has a number "**8**", indicating that the given problem has 8 distinct letters.
- 5: Clear list 10 and 11.
- 6: A checkbox to Enable or Disable the sequence validation method. It is "**checked**" therefore, the results showed in list 10 and 11 were found using sequence validation method.
- 7: A dropdown list to choose the number of threads, from 1 to 9, to solve the problem. In this case "**2**" threads were selected from the dropdown list.
- 8: A button to initiate the calculation process.
- 9: Display the work load distributed by thread. It depends in the number of threads selected at 7. In this example "**2**" threads were used therefore, it shows the work load for each thread as show below:
- Thread #1: 1→ 4: 4
- Thread #2: 5→ 9: 9
- 10: Display elapse time and total iterations related to each thread as show below:

Thread	Elapse Time	Total Iterations
1	5101	3168404
2	6364	3960505

- 11: Display the thread number, solution, solution time, and total iterations for all solutions found as show below:

Thread	Soln.	Time	Iterations
2	9567+1085=10652	5912	3671631

- 12: Indicates the thread that found the solution. In this example is the thread number "**2**".
- 13: Indicates the solution found. In this example the unique solution found is "**9567 + 1085 = 10652**".
- 14: Indicates the elapse time (milliseconds) to find the solution. In this example the solution was found after "**5912**" milliseconds.
- 15: Indicates the total iterations required to find the solution. In this example the solution was found after "**3671631**" iterations.
- 16: Indicates the thread ID. In this example two threads were used therefore the IDs are 1 and 2.

- 17: Indicates the elapse time required to verify all possible solutions in the range associated with the thread. In this example thread number 1 and 2 finished after "**5101**" and "**6364**" milliseconds respectively.
- 18: Indicates the total iterations required to verify all possible solutions in the range associated with the thread. In this example thread number 1 and 2 finished after "**3168404**" and "**3960505**" iterations milliseconds respectively.

RESULTS

The algorithm was implemented with C Sharp language and has been applied on commonly used cryptarithmetic problems. Each problem was executed five times. The elapse time, solution time, total iterations, speed-up, and efficiency metrics were calculated per each execution to analyze results. This metrics are defined as follow:

- **Total Iterations**: Total number of iterations required to obtain the results or complete the process.
- **Solution Time**: Total time it takes to find a solution. Solution time is defined as shown in (1).

$$SolutionTime = \frac{ST}{p} = SE_p - S_p \qquad (1)$$

Where the variables are defined as follow:

- p is the number of processors
- S_p is the start time
- SE_p is the time where the solution was found

- **Elapse Time**: Total time it takes to complete the whole process. Elapse time is defined as shown in (2).

$$ElapseTime = ET = E_p - S_p \qquad (2)$$

Where the variables are defined as follow:

- p is the number of processors
- S_p is the start time
- E_p is the end time

The difference within solution and elapse time is the moment in where the end time is obtained. The start time is the same for both. The solution time is the time it takes to find a solution and the elapse time is when it finishes the whole process as shown in Figure 9.

```
class Solver
   start= empty
   solver(start time,...)
     start  =  start time
   return
   solve()
       .
    if solution found then
     call addResutlsItem() //Solution Time
    endif
       .
    calladdTerminationItem()//Elapse Time
   return
   addResutlsItem()
     get system time
     end = system time
   return end - start
   addTerminationItem()
     get system time
     end = system time
   return end - start
End Class
```

Figure 9
Solution and Elapse Time Pseudo Code

Two important measures of the quality of parallel algorithms are speedup and efficiency [6].

- **Speed-up**: Indicates how much a multithreading algorithm is faster than a corresponding single thread algorithm. Speed-up is defined as shown in (3).

$$Speedup = S_p = \frac{T_1}{T_p} \qquad (3)$$

Where the variables are defined as follow:

- p is the number of processors
- T_1 is the execution time of the sequential algorithm
- T_p is the execution time of the parallel algorithm with p processors

Linear speedup or ideal speedup is obtained when Sp = p. When running an algorithm with linear speedup, doubling the number of processors doubles the speed. As this is ideal, it is considered very good scalability.

- **Efficiency**: Estimates how well-utilized the multithreads are in solving the problem, compared to how much effort is wasted in communication and synchronization. Efficiency is a performance metric defined as shown in (4).

$$Efficiency = E_p = \frac{S_p}{p} = \frac{T_1}{p \bullet T_p} \qquad (4)$$

Where the variables are defined as follow:

- p is the number of processors
- T_1 is the execution time of the sequential algorithm
- T_p is the execution time of the parallel algorithm with p processors
- S_p is the speed-up

Result for 8 Variable Cryptarithmetic Problem SEND+MORE=MONEY

The solution time could be reduced from an average of 11,717 to 1,781 milliseconds when using 5 threads and the elapse time could be reduced from 18,713 to 3,416 milliseconds when using 8 threads as show in Table 6.

Table 4 and 5 shows the work load distribution using 1 to 5 and 6 to 9 threads respectively and Table 5 shows the results using 1 to 9 threads.

Table 4
Work Load Distribution by Range for 8 Distinct Letter Problems and One to Five Threads.

Thread Load by Range for 8 distinct letter problems (threads 1-5)									
1 Thread		2 Threads		3 Threads		4 Threads		5 Threads	
Range	Max Iterations	Range	Max Iterations	Range	Max Iterations	Range	Max Iterations	Range	Max Iterations
1-9	99,999,999	1-4	44,444,444	1-3	33,333,333	1-2	22,222,222	1-2	22,222,222
		5-9	55,555,555	4-6	33,333,333	3-4	22,222,222	3-4	22,222,222
				7-9	33,333,333	5-6	22,222,222	5-6	22,222,222
						7-9	33,333,333	7-8	22,222,222
								9-9	11,111,111
	99,999,999		99,999,999		99,999,999		99,999,999		99,999,999

The thread load in some cases is not properly balanced. This is because the iterations are distributed among the threads and the maximum number of iterations for problems containing 8 distinct letters is 99,999,999. Therefore, the only way to have a balanced work load is using 3 or 9 threads in which the work load can be distributed in ranges with a maximum of 33,333,333 or 11,111,111 iterations respectively.

Table 5
Work Load Distribution by Range for 8 Distinct Letter Problems and Six to Nine Threads.

Thread Load by Range for 8 distinct letter problems (threads 6-9)							
6 Threads		7 Threads		8 Threads		9 Threads	
Range	Max Iterations	Range	Max Iterations	Range	Max Iterations	Range	Max Iterations
1-1	11,111,111	1-1	11,111,111	1-1	11,111,111	1-1	11,111,111
2-3	22,222,222	2-3	22,222,222	2-2	11,111,111	2-2	11,111,111
4-4	11,111,111	4-4	11,111,111	3-3	11,111,111	3-3	11,111,111
5-6	22,222,222	5-6	22,222,222	4-5	22,222,222	4-4	11,111,111
7-7	11,111,111	7-7	11,111,111	6-6	11,111,111	5-5	11,111,111
8-9	22,222,222	8-8	11,111,111	7-7	11,111,111	6-6	11,111,111
		9-9	11,111,111	8-8	11,111,111	7-7	11,111,111
				9-9	11,111,111	8-8	11,111,111
						9-9	11,111,111
	99,999,999		99,999,999		99,999,999		99,999,999

The reduction of iterations is due to multithreading and sequence validation methods. The best results reached for this problem has a speed-up of 6.6 and an efficiency of 132% using 5 threads as show in Table 6 for SEND+MORE=MONEY problem.

Table 6
Execution Results for Eight Variables Cryptarithmetic Problem SEND+MORE=MONEY

SEND+MORE=MONEY									
Count				Solution Time (ms)				Metrics	
# Thread/s Used	Solution Thread/s	Total Solutions Found	Iterations	Max Elapse Time (ms)	Min	Max	Average	Speed-up	Efficiency %
1	1	1	6,840,035	18,713	10,140	17,933	11,717	1.0	100
2	2	1	3,671,631	6,645	5,975	6,224	6,075	1.9	96
3	3	1	2,087,429	5,398	4,664	4,786	4,726	2.5	83
4	4	1	2,087,429	6,220	5,406	5,434	5,424	2.2	54
5	5	1	503,227	5,529	1,669	1,887	1,781	6.6	132
6	6	1	1,295,328	5,054	4,212	4,383	4,288	2.7	46
7	7	1	503,227	3,610	1,965	2,199	2,046	5.7	82
8	8	1	503,227	3,416	2,039	2,366	2,145	5.5	68
9	9	1	503,227	3,744	2,355	2,456	2,403	4.9	54

The thread that has a range containing {9,_,_,_,_,_,_,_} is always the thread that find the solution to the problem. This is because the first letter "S" of the given problem is equal to 9. Figure 5 shows the relation between total iterations and solution time. Figure 10 shows the change in efficiency and speedup. The solution time decrease as increases the number of threads from 1 to 3 threads. This is because the search space decreases as increase the number of threads as shown in Table 4 when using from 1 to 3 threads. The execution with 4 threads showed an increment in the solution time due to the overhead. The overhead in this case is because the search range has the same numbers of iterations compared with the execution of 3 threads but one more thread was used. The solution time decrease when using 5 threads because the search space is smaller when compared with 1, 2, 3, and 4 threads ranges as show in Table 4.The solution time increase when using 6 threads because the search space is bigger and also more threads were used, when compared with 1, 2, 3, 4, and 5 threads ranges as show in Table 4. The solution time increase when using 7, 8, and 9 threads due to overhead, because the search space is the same and the number of threads increases. Therefore, speed-up and efficiency decreases due to overhead as shown in Figure 11.

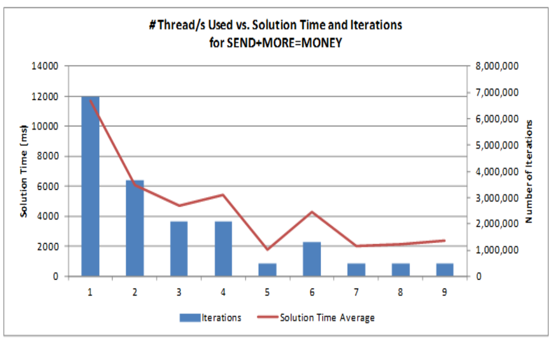

Figure 10
Number of Threads vs. Solution Time

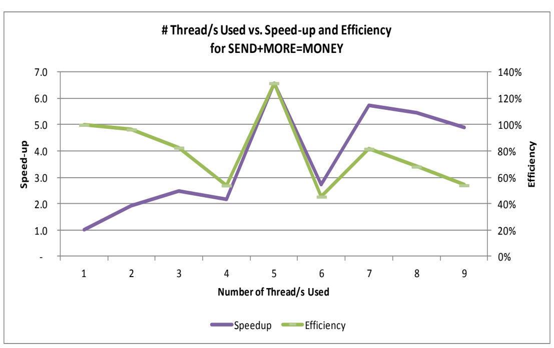

Figure 11
Number of Threads Used vs. Speed-up and Efficiency

Comparing Results

We compared the Parallel Genetic Algorithm (PGA), Efficient Parallel Algorithm (EPA), and Evolutionary Algorithm (EA) with the proposed Multithreading and Sequence Validation Algorithm (MSVA) results. Table 7 is a summary of results based on commonly used cryptarithmetic problems and illustrates the comparison between the proposed algorithm (MSVA) and all other algorithms mentioned above..

MSVA has better results than all other algorithms for a 9 distinct variable problem as show in Table 7 using BASIC+LOGIC=PASCAL problem. In this problem MSVA reaches a solution in an average time of 1.36 seconds where it takes the PGA, EA, and EPA 2.53, 10.52, and 12.58 relatively. In general terms MSVA showed good results solving 9 and 10 distinct variables problems in comparison with others algorithms.

Table 7
Execution Results for Eight Variables Cryptarithmetic Problem SEND+MORE=MONEY

Algorithm	Problem	Min Time (S)	Max Time (s)	Ave. Time (s)
PGA	BROWN+YELLOW=PURPLE	0.43	3.632	2.421
MSVA		3.104	3.603	3.252
EPA		9.67	26.089	18.94
EA		0.288	512	87.279
MSVA	BASIC+LOGIC=PASCAL	1.138	1.497	1.36
PGA		0.574	3.342	2.533
EA		0.24	379.52	10.521
EPA		8.976	16.178	12.583
PGA	SEND+MORE=MONEY	0.18	0.974	0.68
EA		0.24	9.248	1.669
MSVA		1.669	1.887	1.781
EPA		1.356	17.16	1.781

CONCLUSION

This project concentrated on designing and implementing a multithreading sequence validation algorithm to solve cryptarithmetic problems. Advantage of our approach are the algorithm is simple for implementation, iteration process and evaluation is parallelized by using multithreading method, and there is no need for any communication mechanism. The use of multithreading techniques combined with sequence validations showed that it is possible to find the result of large instances of cryptarithmetic problems within an acceptable time.

FUTURE WORK

The proposed algorithm in this paper has the number of threads as an initial parameter. While the implementation of this algorithm is simple; the iterations process must increase for certain number of threads due to the fact that some threads may perform additional iterations if non proportional work loads are encountered. Assigning the number of threads is problem oriented and depends on the problem. Some mechanism can be established to find an ideal number of threads for each specific Cryptarithmetic problem in order to get better results. A good selection in the number of threads can reduce the calculation time and the possible overhead of threads.

REFERENCES

[1] Reza, Abbasian, et al., "Solving Cryptarithmetic Problems Using Parallel Genetic Algorithm", *In Proceedings of the 2009 Second International Conference on Computer and Electrical Engineering*, Vol. 01, 2009, pp. 308-312.

[2] Hassan, Kamrul, et al., "An Evolutionary Algorithm to Solve Cryptarithmetic Problem", *In Proc. International Conference on Computational Intelligence*, 2004, pp. 494-496.

[3] Gholamali, Rahnavard, et al., "An Efficient Parallel Algorithm for Solving Cryptarithmetic Problems: PGA", *In Proceedings of the 2009 Third UKSim European Symposium on Computer Modeling and Simulation*, 2009, pp. 102-106.

[4] Naoghare, Manisha, et al., "Comparison of parallel genetic algorithm with depth first search algorithm for solving verbal arithmetic problems", *In Proceedings of the International Conference & Workshop on Emerging Trends in Technology, 2011*, pp. 324-329.

[5] Jagger, Jon, et al., "C# Annotated Standard", Morgan Kaufmann Publishers Inc., 2007, pp. 640

[6] Hurley, Stephen, "Factors That Limit Speedup", http://www.cs.cf.ac.uk/Parallel/Year2/section7.html, February 5, 2012.