ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 2, February 2014

Use of Changing Wave-Front Angles Approach for Tiled Iteration Space Scheduling of Three-Level Nested Loops

AliReza Hajieskandar[1], Shahriar Lotfi[2], Simin Ghahramanian[3]

Department of Electrical and Computer Engineering, Islamic Azad University, Bonab, Iran

Department of Computer Science, University of Tabriz, Tabriz, Iran

Sama Technical and Vocational Training College, Islamic Azad University, Bonab, Iran

Abstract: The shortage of run time is a determinant factor in executing programs. One of the popular methods in this literature is the parallel execution of programs. The need for high computational speed and power in a majority of scientific applications fuels the incentives for gaining the computational power of several processors to raise the execution speed of programs. Furthermore, the presence of sequential programs, once very costly generated, provokes the engagement of tools known as "super-compilers" for automatic conversion of sequential codes into parallel codes. In most of computational programs nested loops for which a great amount of time is needed are used. The computations inside loops which have no interdependence can be partly executed in parallel by the engagement of several processors. One of the conversion stages of sequential nested loops into parallel ones is to schedule the tiled iteration space. Regarding the fact that, so far the block and cyclic approaches have been introduced, in this paper the wave-front approach and wave-angle changes have been incorporated in the block and cyclic approaches in order to reduce the execution time of three-level nested loops.

Keywords: Nested Loops, Iteration Tiled Space, Scheduling, Wave Fronts and Wave-Angle Change.

I. INTRODUCTION

Time is one of important factors in programs. Decreasing the execution time is one of crucial goals in program execution. To decrease the execution time many different approaches have been suggested. One important method is the parallel execution of programs. In most programs repeated loops may be observed. These loops run some parts of the program's internal instructions frequently and as a consequence the spent time for running them is noticeable. This issue becomes more important especially when the repeat loops are nested. If we could run some parts of the computations inside loops which have no interdependence; i.e. the execution of each part is independent of the others; in parallel by using several processors then the execution speed will be improved. Therefore need for high computational speed and power in a majority of scientific applications fuels the incentives for gaining the computational power of several processors to raise the execution speed of programs. Furthermore, the presence of sequential programs, once very costly generated, provokes the engagement of tools known as "super-compilers" for automatic conversion of sequential codes into parallel codes. Super compilers can detect the hidden parallelism in programs and next convert a sequential program into a parallel one. So the parallelization of nested loops is a key challenge in shortening the computer program run-time [15]. The conversion of nested loops into parallel ones is accomplished through the following orderly stages: In stage I, as two instructions can run in parallel only when there is no data dependency between them, the data dependencies in nested loops will be analysed and extracted accordingly. Data dependencies originate from the arrays used inside loops .Therefore if the instruction S executes before the instruction T and generates some data that the instruction T will use it certainly, T is called a data dependent or shortly a dependent on S. In stage II, in order to achieve better parallelization, to decrease inter-processor connections and hence to optimally distribute the dependent iterations of nested loops for being executed in processors, iteration space is tiled. A set of loop iterations running in one processor is called a tile. In stage III, based on the shape and size of produced tiles a desired parallel code is generated for the iteration space of stage II. Finally in the stage IV the tiled space of the former stage is scheduled by using the wave-front approach. In fact, the tiles are assigned to the processors in a way that the time needed for executing all of them is minimized accordingly. In other words termination time for the last tile of final wave is shortened as much as possible.

In this paper, the tiled iteration space for **three-level nested loops** is scheduled over a multi-processor system by shifting the wave-angle as in the block and cyclic scheduling approaches.

The remainder of this paper is organized as follows: In section two we address to the tiled space scheduling. In section three we review the basic concepts of the problem in question. Section four explains former related works in the context, and section five describes our proposed approach. Six and seven sections focus on the assessment of experiments and conclusions as well as a guideline for future works respectively.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 2, February 2014

II. THE PROBLEM

The main purpose behind this is to allocate tiles to the existing processors of a multi-processor system so that the overall execution time of all tiles is minimized.

To schedule loops the processors are interconnected according to a given topology. Data exchange could be performed only between two connected processors in message format and respecting the algorithm‘s governing rules. The execution time of instructions is computed by adding the execution time of each processor to the dispatch time of each message. In three dimensional spaces the interconnection topology of processors is cubic so each processor is connected to the other processors in the three axes of vertical, horizontal and depth. The commands are divided among processors; the way, start time and end time of executing them over each processor, the transposition of each instruction relative to others and the method of data exchange among processors are specified and the processors will execute the assigned commands to them. In the rest, in order to clarify the problem two examples from nested loops are shown as follows:

Example1:

Suppose the number of processors is 3 and the loop dimensions in the three axes of vertical, horizontal and depth is 2. In the following figure two solutions obtained after the execution of the scheduling algorithm are shown.

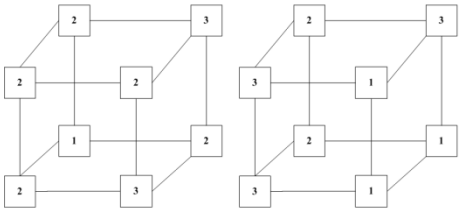

Fig. 1. Two typical scheduling for example 1

Example 2:

Consider the below nested loop:

```
For i:= 0 To 9 Do
  For j:= 0 To 5 Do
    For k:= 0 To 9 Do
      A[i, j, k]:= A[i-1, j, k] + A[i, j-1, k]+A[i, j, k-1];
    EndFor
  EndFor
EndFor
```

In the above example reference to the array causes internal data dependency in loop iterations. Therefore, each iteration [i, j, k] depends on its three neighbours namely [i-1, j, k], [i, j-1, k] and [i, j, k-1].

In order to decrease communication among processors the iteration space can be tiled. The purpose of tiling is to distribute optimally the dependent iterations of nested loops for being executed only over the processors that have message transactions. The main challenge in this literature is the plentiful number of dependencies. For performing this task the different iterations of loops are categorized in a tile format. In fact a tile is the set of points in the iteration space which have highest data dependency to each other and should be executed over the same processor. The determination of the tile size and form is an NP-hard problem [13, 24]. If the iteration space is not tiled correctly, the inter-processor commutations will be high.

In this problem we want to assign a certain number of processors to the existing tiles in the tiled space, so the inputs of our problem are the tiled iteration space and a certain number of specified processors. In this tiled space, each tile has two prerequisite tiles as, and it means that in the related tiled space there is an edge from three prerequisite tiles toward the corresponding tile. So while the execution of prerequisite tiles has not been finished completely, cannot start execution.

For optimum scheduling of tiled space two constraints are postulated: (1) balanced loads among processors (2) The least possible cost for inter-processor connections when scheduling. The nearly identical figures of tiles (tiles are created incompletely in borders) and the execution of each tile in one processor satisfies the first constraint. In order to reduce inter-processor connecting cost, the tiles with higher communication cost should be assigned to the same processors so that with the zero connection time, the time needed for executing tiles minimizes [11].

We denote the cost of communication and message sending between the current tile and the prerequisite ones with Vcomm. Since each tile connects to two tiles of different dimensions, so we use two parameters as Vcomm(1) ,Vcomm(2) and Vcomm(3). Vcomm(1) stands for the communication volume of with the neighbouring tile in respect for the first dimension while Vcomm(2) stands for the communication volume of with the neighbouring tile for the second dimension and Vcomm(3) stands for the communication volume of with the neighbouring tile for the third dimension. As long as and its neighbouring tiles are executed over different processors we have to tolerate the existing costs as the problem input but for the case of identical processors , the communication cost will be zero. Regarding the fact that processors are fully-connected in style; i.e. each processor is directly connected to other processors, the connection cost among all processors is identical.

The execution cost of each tile in an individual processor, Vcomp, for all tiles is identical and is as input of problem. So the termination time of tile is computed through the following equation:

$$completiontime(J^S) = Max\begin{pmatrix} completiontime(J'^S) + V_{comm}(1), \\ completiontime(J''^S) + V_{comm}(2), \\ completiontime(J'''^S) + V_{comm}(3) \end{pmatrix} \quad (1)$$

The total time duration for executing all current scheduled tiles, known as "makespan", this parameter that is an output of problem is given by the following equation:

$$makespan = \max(completiontime(J^S)) \quad , J^S \in \text{Tiled Iteration Space} \quad (2)$$

III. DATA DEPENDENCY AND THE NESTED LOOPS ITERATION SPACE TILING, AN OVERVIEW

The first stage in parallelization of nested loops is to analyse the data dependency problems; two instructions can be executed simultaneously only if there is no data dependency between them [11]. Generally speaking the T instruction is called S - data dependent or shortly S-

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 2, February 2014

dependent [11] if both of instructions refer to the same memory location, S instruction is executed before T instruction, if there is an execution path between the two instructions and in the time interval between S and T instructions nothing is recorded in the commonly referred location.

In order to achieve better parallelization and cache locality of reference in individual processors and extraction of great parallelization in multiprocessors the iteration space of nested loops is tiled. A set of loop iterations which must run in an individual processor is called a tile. The purpose from tiling is to reduce communication volume among processors and consequently to optimally distribute the dependent nested loops for being executed in the processors which exchange messages with each other. A sample tiling method is depicted in the below diagram [4, 11].

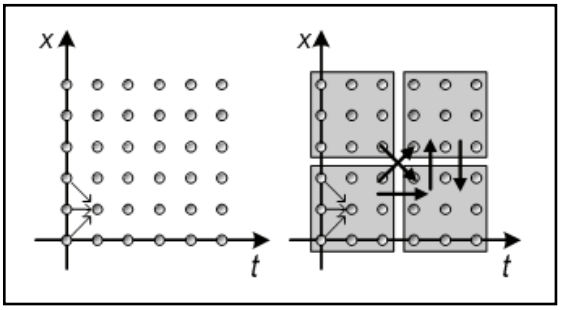

Fig. 2. An example of Iteration Space Tiling

IV. RELATED WORKS

As the scheduling of iteration space for nested loop is a determinant factor in increasing the run speed of programs, so far w many approaches have been suggested to deal with this problem. Any of these methods somehow tries to decrees the run time of repeated loops in a parallel manner [1, 6, 8, 11, 14-16, 18, 19, 21, 23, 25, 26]. But for three-level iteration space two approaches of the block and cyclic scheduling are suggested [13, 14, and 22]. In some methods, in order to minimize the overall execution time of iterations, the communication time of processors overlaps with the time of internal calculations of processors [3, 8, 23]. Also some other approaches inspired by the wave-front method, eliminate the need for communicating of many processors and assign dependent caches to the same processor [8, 9, 14, 23]. In other approaches, the critical path for the task graph resulted from the loop iterations' space is calculated and in order to obtain load balancing, other non-critical iterations are distributed among processors according to their data dependency [1].

We in the former articles [9] and [10] using the wave-front method introduced two genetic-based algorithms for scheduling two level nested loop. One of the methods with angle shift and the other without it can culminate at the proper scheduling of the problem in question.

We also achieved optimum results in [11] by shifting the wave angle in the following four approaches: the horizontal block, the vertical block, the horizontal cyclic and the vertical cyclic. They schedule the two-level iteration space. In this paper we extended this approach for the three-level space.

Some of these methods [13, 14] try to remove the great number of interactions among processors and assign dependent tiles to a single processor by using the block and cyclic approaches. In the block approaches the tiles are blocked horizontally or vertically and each block is assigned only to one processor, in a way that if there be n rows and m processors n/m blocks will be created horizontally or vertically and a single processor will be allocated to one block. Also in the cyclic approaches each row or column will be assigned to the same processor alternately and intermittently. So in these cases, inter-processor communication is decreased which in turn lowers the overall execution time.

Doritos and et. al. in [7] introduced a low-cost algorithm which using geometrical calculations generates the typical schemes consisting of iteration subsets which can be executed earlier in a certain k-time phase.

Lotfi and Parsa in [20] suggested a new algorithm over the irregular iteration space in which in order to extract efficient waves they have granted that all tiles of identical coordinates are located over one single wave. They improved the block scheduling and achieved better outcomes through using this way. In this case the tiles are taken as parallelepiped with the highest degree of parallelism, load balance and the lowest level of expense. And also by using the Genetic algorithm better results were obtained for parallelepiped tiles compared with rectangular or square shaped tiles.

V. PROPOSED STRATEGY

The proposed approach of BCAATLLS is based on the wave-front approach and wave-angle shifts and by shifting angle in the horizontal Block (BlockH), the Vertical Block (BlockV), the Horizontal Cyclic (CyclicH), the Vertical Cyclic (CyclicV) methods trying to better load balancing between processors and thereby minimizing the makespan of nested loops' execution. In fact, a wave consists of tiles without data dependency which can be executed in parallel. In two surface spaces, waves are as line, and in three surface spaces, as page. If waves are drawn over the iteration loop, they form an angle which its shift culminates at the minimization of required processors in number, the optimization of their loading balance and consequently the optimization of makespan for scheduling. The difference of proposed algorithm with the suggested algorithm in our previous paper [11] is that in this paper due to the three-level nature of loops and consequently the three-dimension iteration space each wave comes a plane .As a result the number of existing tiles over one wave increases to form 5 tile types totally which in the rest we will address to it after showing the pseudo-code of "How to compute the execution end time ".

A. The pseudo code for computing the overall execution time of tiles in a tiled space based on the wave-front method and inspired by the angle concept

To do this the pseudo code of Figure 3 is used. This algorithm is based on the wave-front method. The waves should be executed sequentially. So in this algorithm the

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 2, February 2014

overall execution time equates he sum of tiles' execution times over waves. Presumably, at the wave beginnings processors are synchronized and between two successive waves the transmission time of messages are overlapped possibly. Also for identifying the number of wave in which the tile $J^S = \left(J_1^S, J_2^S, J_3^S\right)$ is located, the Relation No.3 is used in which a and b stand for angle coefficients. (The wave angle is represented by two coefficients of a and b) These coefficients were taken as 1 in the former works which account for a 45 degree angle indeed.

$$Number\ of\ Wave - front(i,j,k) = a \times i + b \times \mathrm{j} + \mathrm{c} \times \mathrm{k} \qquad (3)$$

Scheduling Completion time := 0
ForEach wave front number, wn, lwn ≤ wn ≤ hwn **Do ForEach** Processor Pi, 0 ≤ i ≤ N −1 **Do** Processing time (P_i)= 0;
  ForEach tile $J^s=(j^s_1, j^s_2, j^s_3)$ on wave front wn such that $a.j^s_1 + b.j^s_2 + c.j^s_3 = wn$ **Do**
    find processor number, P_i, assigned to j^s;
    Assume $J'^s =(j^s_1-1, j^s_2, j^s_3)$, $J''^s=(j^s_1, j^s_2-1, j^s_3)$ and $J'''^s=(j^s_1, j^s_2, j^s_3-1)$;
    find processor number, $p_{i'}$, $p_{i''}$ and $p_{i'''}$ assigned to J'^s, J''^s and J'''^s, respectively;
    Processing time(p_i) += V_{comp} . T_{comp} : *communication time of the tile*
    (1) **If** J^s is not dependent on J'^s, J''^s and J'''^s **Then** Processing time(p_i)+= T_{comp}
    (2) **Else If** J^s is only dependent on J'^s **Then If** $p_i \neq p_{i'}$ **Then** Processing time(p_i)+= $V_{comm}(1)$. T_{comp}
    (3) **Else If** J^s is only dependent on J'^s **Then If** $p_i \neq p_{i''}$ **Then** Processing time += $V_{comm}(2)$. T_{comp}
    (4) **Else If** J^s is only dependent on J'''^s **Then If** $p_i \neq p_{i'''}$ **Then** Processing time += $V_{comm}(3)$. T_{comp}
    (5) **Else If** J^s is dependent J'^s, J''^s and J'^s **Then**
      If $p_{i'} \neq p_{i''} \neq p_{i'''}$ **Then If** $p_i \neq p_{i'}$, $p_{i''}$, $p_{i'''}$ **Then**
        Processing time (p_i) +=Max($V_{comm}(1)$. T_{comm} , $V_{comm}(2)$. T_{comm}, $V_{comm}(3)$. T_{comm})
      Else If $p_i \neq p_{i'}$, $p_{i''}$, $p_{i'''}$ **Then** Processing time(p_i) += $V_{comm}(1)$. T_{comm} + $V_{comm}(2)$. T_{comm}+ $V_{comm}(3)$. T_{comm}
    Processing time (wn) = max (processing time (p_i)), ∀i=0, 1,…, N-1
    Scheduling completion time += Processing time (wn)

Fig. 3. The pseudo code for computing the overall execution time of tiles in a three level tiled space

As observable, there are five kinds of tiles: 1- tiles located in the origin of coordinates 2- tiles located alongside the horizontal axis 3- tiles located alongside the vertical axis 4- tiles located alongside the depth axis and 5-internal tiles.

As noted before, two constrains should be postulated for optimum scheduling of tiled space (1) loading balance among processors and (2) low communication cost among processors in scheduling time. The results obtained from the execution of proposed algorithm indicates that shifting wave angles will lower the number of needed processors for scheduling the iteration space with implications for improving the load balance among processors. Therefore the wave angle shift covers the first constraint. The second constraint is tacked with as well by using the mentioned four approaches because these approaches are intrinsically postulated to lower the inter-processor communication during the scheduling process.

VI. ASSESSMENT AND PRACTICAL RESULTS

In order to verify the proposed algorithm BCAATLLS and to contrast its results versus the former algorithms, a software application is designed and implemented in Delphi 2010. The engaged algorithms for comparing the new proposed algorithm include the horizontal Block, the Vertical Block, the Deep Block, the Horizontal Cyclic, the Vertical Cyclic and the Deep Cyclic algorithms.

A. Proposed Algorithm assessment

As the proposed algorithm is definitive, so its yielding solutions are the same for a given problem, so the proposed algorithm is completely (100%) stable and the generated solutions are also 100% reliable.

B. The comparison of the proposed algorithm with former works

In this section, by running implemented program for proposed algorithm and conducting various tests of mixed types, the quality of generated solutions by the proposed algorithm is compared against those of former ones. Notably, because of page paucity, only some of tests are shown here. The testing results as well as the related parameters of each test are tabulated in Table 1.

TABLE I
THE OUTCOME RESULTS FROM IMPLEMENTING THE PROPOSED ALGORITHM AND FORMER ALGORITHMS

Experiment Number	NP	UB1	UB2	UB3	Vcomp	Vcom1	Vcom2	Vcom3	Previous Approaches						Proposed Approaches																				
									BlockV	BlockH	BlockD	CyclicV	CyclicH	CyclicD	BlockV	a	b	c	BlockH	a	b	c	BlockD	a	b	c	CyclicV	a	b	c	CyclicH	a	b	c	CyclicD	a	b	c
1	2	4	4	4	9	1	1	1	825	825	825	725	725	725	711	1	2	3	711	2	1	3	711	2	3	1	725	1	1	1	725	1	1	1	725	1	1	1
2	2	4	4	4	9	4	3	1	870	855	825	875	825	725	738	1	2	3	729	2	1	3	711	2	3	1	875	1	1	1	825	1	1	1	725	1	1	1
3	2	4	4	4	6	4	4	4	600	600	600	650	650	650	506	1	2	3	506	2	1	3	506	2	3	1	650	1	1	1	650	1	1	1	650	1	1	1
4	2	9	5	5	1	3	3	3	363	378	378	721	721	721	303	1	2	5	365	9	1	3	365	9	3	1	721	1	1	1	721	1	1	1	721	1	1	1
5	2	5	5	5	1	2	3	4	204	234	267	325	433	541	186	1	1	4	222	1	1	3	258	1	3	1	325	1	1	1	433	1	1	1	541	1	1	1
6	3	9	4	9	9	1	1	1	2690	2220	2690	1950	2115	1950	2010	1	2	5	1978	2	1	5	2010	5	2	1	1950	1	1	1	2034	1	1	5	1950	1	1	1
7	3	9	6	6	9	4	3	1	2911	2610	2514	2370	2310	2030	2150	1	4	6	2187	7	1	3	2049	7	3	1	2368	1	1	2	2310	1	1	1	2030	1	1	1
8	3	4	4	4	9	4	3	1	710	690	650	681	639	555	623	1	2	3	600	2	1	3	554	2	3	1	671	1	1	2	627	1	1	2	539	1	2	1
9	3	6	6	6	6	4	4	4	1366	1366	1366	1284	1284	1284	1146	1	3	5	1146	3	1	5	1146	3	5	1	1284	1	1	1	1284	1	1	1	1284	1	1	1
10	3	6	4	8	1	8	8	8	606	732	556	997	1135	1045	525	1	1	5	650	1	1	7	424	5	1	1	887	1	1	5	1135	1	1	1	975	5	1	1
11	3	6	4	4	1	6	8	5	305	429	294	452	631	421	258	1	1	3	370	5	1	1	259	5	1	1	427	1	1	2	631	1	1	1	421	1	1	1
12	3	8	8	9	1	2	3	4	603	712	806	877	1169	1432	471	1	1	9	564	1	1	9	681	4	5	1	822	1	1	9	1095	1	1	9	1362	1	3	1
13	3	12	8	9	1	2	3	4	881	977	1095	1244	1669	2029	658	1	5	7	804	10	1	1	963	9	2	1	1186	1	1	9	1575	10	1	1	1896	9	1	1
14	4	9	12	12	9	4	3	1	7096	7848	7608	6617	5862	5070	5581	1	3	10	5370	4	1	10	5108	4	10	1	6613	1	1	3	5862	1	1	1	5070	1	1	1
15	4	4	4	4	6	5	4	3	532	506	480	454	416	381	469	1	2	3	446	2	1	3	423	2	3	1	445	1	1	3	416	1	1	1	381	1	1	1
16	4	6	4	8	1	8	8	8	629	732	556	901	772	848	490	1	1	5	650	1	1	7	424	5	1	1	830	1	1	5	654	1	1	7	713	5	1	1
17	4	6	9	8	1	6	8	5	950	1061	740	1366	1702	1081	752	1	9	1	756	1	1	7	590	1	7	1	1276	1	9	1	1702	1	1	1	940	1	7	1
18	4	9	5	5	1	3	3	3	360	365	365	433	481	481	267	1	3	4	313	6	1	1	313	6	1	1	433	1	1	1	481	1	1	1	481	1	1	1
19	5	6	6	6	4	4	4	4	1110	1110	1110	988	988	988	914	1	2	5	914	2	1	5	914	2	5	1	988	1	1	1	988	1	1	1	988	1	1	1
20	5	4	4	4	6	5	4	3	447	408	369	447	408	369	381	1	1	3	348	1	1	3	315	1	3	1	381	1	1	3	348	1	1	3	315	1	3	1
21	5	9	9	9	6	5	4	3	2739	2596	2463	2702	2458	2214	1926	1	2	5	1818	2	1	5	1710	2	5	1	2351	1	1	5	2140	1	1	5	1929	1	5	1
22	5	4	4	4	1	8	8	8	361	361	361	361	361	361	307	1	1	3	307	1	1	3	307	1	3	1	307	1	1	3	307	1	1	3	307	1	3	1
23	5	4	4	4	1	6	8	5	281	361	241	281	361	241	239	1	1	3	307	1	1	3	205	1	3	1	239	1	1	3	307	1	1	3	205	1	3	1
24	5	9	5	5	1	3	3	3	350	365	365	373	354	354	229	1	2	4	313	6	1	1	313	6	1	1	337	1	1	3	312	6	1	1	312	6	1	1

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 2, February 2014

For better comparison, the obtained results of performed tests are depicted in figure 4.

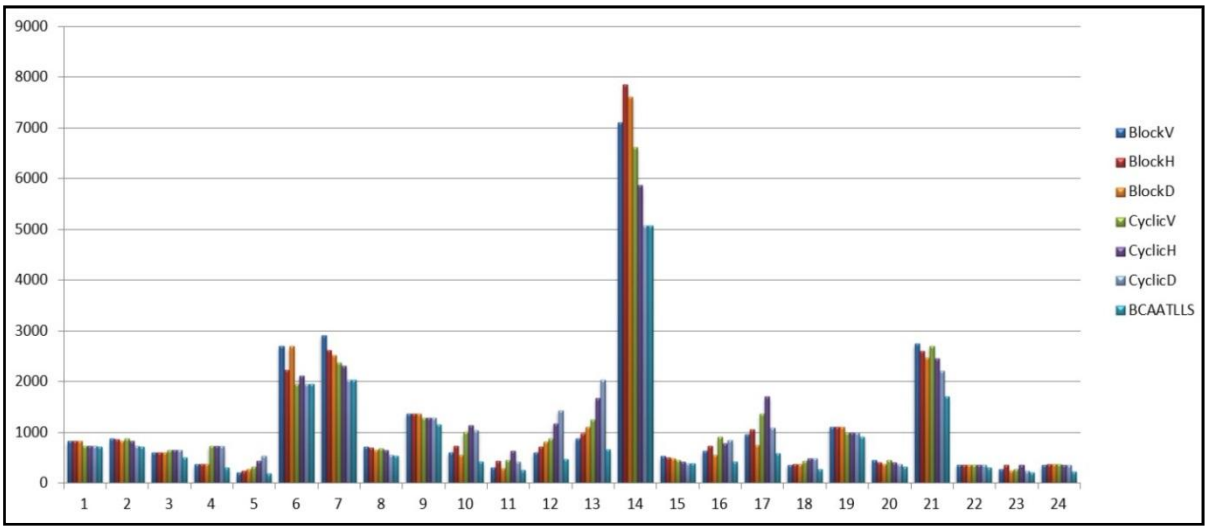

Fig. 4. The comparison of results obtained by running the proposed algorithm and former algorithms

VII. CONCLUSION AND FUTURE WORKS

In this paper, taking the benefits of angle shifts, an algorithm was proposed which due to its potential in improving the loading balance among processors and consequently optimizing of their usage, minimized the execution time of tiles. As he results show, the proposed algorithm (BCAALS) yielded better results in 93% of cases relative to the Block or Cyclic methods.

On the stability and reliability of the proposed algorithm it just suffices to note that as the proposed algorithm is not stochastic, its generated solutions are stable and completely (100%) reliable.

Using random algorithms for solving this problem, the parallel implementation of random algorithms such as the Genetic Algorithm and mixing the Genetic Algorithm with other intelligent searching algorithms such as Learning Automata.

ACKNOWLEDGMENT

This manuscript is original and is un-published work and is not be under-consideration for publication elsewhere.

REFERENCES

[1] T. Andronikos, M. Kalathas, F. M. Ciorba, P. Theodoropoulos, and G. Papakonstantinou, An Efficient Scheduling of Uniform Dependence Loops, Computing Systems Laboratory, Department of Electrical and Computer Engineering, National Technical University of Athens, Zographou Campus, Athens, Greece, pp. 1-10, 2004.

[2] M. Athanasaki, E. Koukis, and N. Koziris, Scheduling of Tiled Nested Loops onto a Cluster with a Fixed Number of SMP Nodes, 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, IEEE, 2004.

[3] M. Athanasaki, A. Sotiropoulos, G. Tsoukalas, N. Koziris, and P. Tsanakas, Hyperplane Grouping and Pipelined Schedules: How to Execute Tiled Loops Fast on Clusters of SMPs, Journal of Supercomputing, vol. 32, pp. 197-226, 2005.

[4] D. K. Chen and P. C. Yew, An Effective Execution of Non-Uniform Do Across Loops, pp. 1-6, February 1995.

[5] A. Darte, L. Khachiyan, and Y. Robert, Linear Scheduling Is Nearly Optimal, pp. 73-81, 1991.

[6] A. Darte, Y. Robert, and F. Vivien, Scheduling and Automatic Parallelization, 2000.

[7] I. Drositis, T. Andronikos, A. Kokorogiannis, G. Papakonstantinou, and N. Koziris, *Geometric Pattern Prediction and Scheduling of Uniform Dependence Loops*, 2001.

[8] G. Goumas, A. Sotiropoulos, and N. Koziris, *Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping*, IEEE, pp. 1-10, 2001.

[9] A. Hajieskandar and S. Lotfi, *Parallel Loop Scheduling Using an Evolutionary Algorithm*, pp. 314-319, August 2010.

[10] A. Hajieskandar and S. Lotfi, *Using an Evolutionary Algorithm for Scheduling of Two-Level Nested Loops*, pp. 100-105, May 2011.

[11] A. Hajieskandar, S. Lotfi, and S.Ghahramanian, *Two Level Nested Loops Tiled Iteration Space Scheduling By Changing Wave-Front Angles Approach*, International Journal of Advanced Research in Computer and Communication Engineering, vol. 1, Issue 3, pp. 126-133, May 2012.

[12] L. Lamport, *The Parallel Execution of Do Loops*, Comm. of the ACM, vol. 37, No. 2, pp. 83-93, February 1974.

[13] S. Lotfi and S. Parsa, *Parallel Loop Generation and Scheduling*, Journal of Supercomputing, vol. 50, No. 3 pp. 289-306, 2009.

[14] N. Manjikian and T. S. Abdelrahman, *Scheduling of Wavefront Parallelism on Scalable Shared-Memory Multiprocessors*, pp. 1-10, 1996.

[15] D. I. Moldovan and J. Fortes, *Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays*, vol. C-35, No. 1, pp. 1-11, 1986.

[16] T. W. O'Neil, *Techniques for Optimizing Loop Scheduling*, Ph. D. Thesis, The Graduate School of the University of Notre Dame, Indiana, 2002.

[17] S. Parsa and S. Lotfi, *A New Approach to Parallelization of Serial Nested Loops Using Genetic Algorithms*, Journal of Supercomputing, vol. 36, No. 1, pp. 83–94, 2006.

[18] S. Parsa and S. Lotfi, *A New Genetic Algorithm for Loop Tiling*, Journal of Supercomputing, vol. 37, No. 3, pp. 249–269, 2006.

[19] S. Parsa and S. Lotfi, *Code Generation and Scheduling for Parallelization of Multi-Dimensional Perfectly Nested Loops*, pp. 20-22, February 2007.

[20] S. Parsa and S. Lotfi, *Wave-Fronts Parallelization and Scheduling*, pp. 382-386, November 18-20, 2007.

[21] D. L. Pean, H. T. Chua, and C. Chen, *A Release Combined Scheduling Scheme for Non-Uniform Dependence Loops*, Journal of Information Science and Engineering, vol. 18, pp. 223-255, 2002.

[22] J. Ramanujam and P. Sadayappan, *Tiling Multidimensional Iteration Spaces for Multicomputers*, Journal of Parallel and Distributed Computing, vol. 16, No. 2, pp. 108-230, 1992.

[23] F. Rastello and Y. Robert, *Automatic Partitioning of Parallel Loops with Parallelepiped-Shaped Tiles*, vol. 13, No. 5, pp. 460-470, May 2002.

[24] W. Shang and J. A. B. Fortes, *Time Optimal Linear Schedules for Algorithms with Uniform Dependencies*, IEEE Transactions on Computers, vol. 40, No. 6, pp. 723-742, 1991.

[25] J. Xue, *On Tiling as a Loop Transformation*, vol. 7, No. 4, pp. 409-424, 1997.

[26] C. T. Yang and K. Cheng, *An Enhanced Parallel Loop Self-Scheduling Scheme for Cluster Environments*, Journal of Supercomputing, vol. 34, pp. 315-335, 2005.

BIOGRAPHIES

AliReza Hajieskandar received the B.Sc. In Software Engineering from Islamic Azad University Shabestar Branch, Iran and the M.Sc. degree in Software Engineering from Islamic Azad University Qazvin Branch, Iran. He is preceptor of Software Engineering at the Islamic Azad University Bonab Branch. His research interests include compilers, super-compilers, parallel processing, evolutionary computing and algorithms.

Shahriar Lotfi received the B.Sc. in Software Engineering from the University of Isfahan, Iran, the M.Sc. degree in Software Engineering from the University of Isfahan, Iran, and the Ph.D. degree in Software Engineering from Iran University of Science and Technology in Iran. He is Assistant Professor of Computer Science at the University of Tabriz. His research interests include compilers, super-compilers, parallel processing, evolutionary computing and algorithms.

Simin Ghahramanian received the B.Sc. in Software Engineering from Islamic Azad University shabestar Branch, Iran, and the M.Sc. degree in computer system architecture from Islamic Azad University East Azarbaijan Science and Research Branch. She is preceptor of Software Engineering at the Sama Technical and Vocational Training College, Islamic Azad University, Bonab Branch. His researches interests include parallel processing, Network Computing and NOC Testing.