2011 International Conference on Information and Electronics Engineering
IPCSIT vol.6 (2011) © (2011) IACSIT Press, Singapore

Using an Evolutionary Algorithm for Scheduling of Two-Level Nested Loops

AliReza Hajieskandar [1], Shahriar Lotfi [2 +]

[1] Electrical and computer engineering Departments, Islamic Azad University Bonab Branch
[2] Computer Science Departments, University of Tabriz

Abstract. 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. Super compilers can detect the hidden parallelism in programs and next convert a sequential program into a parallel one. Most of computer programs use nested loops. Parallel execution of associated loops might accelerate the execution of in-question programs. So the parallelization of nested loops is a key challenge in shortening the computer program run-times. One of underlying stages in parallelization is scheduling tiled space for iterating nested loops. As the problem is a NP-Hard one, using traditional search methods for solving such programs does not fit the case. So evolutionary algorithms must be engaged to solve these kinds of problems. In this paper, based on the general wave-front method, one distinctive approach is developed. Practical results show that our solution approach which is inspired by heuristics provides better solutions than previous approach alternative solutions in the literature.

Keywords: nested loops, Iteration Tiled Space, Scheduling, Wave Fronts and Evolutionary Algorithm.

1. Introduction

Important feature of parallel compilers in converting sequential programs into Parallel ones is specifying the highest possibility of parallelization [7]. Researchers in the literature have come to the point that loops consume time significantly more than other time –taking commands. If one could run loops in parallel, the speed of program execution increases notably. Converting nested loops into parallel ones is accomplished through the following orderly stages: In stage one; we should create a time-table [1] which preserves the data dependencies in the loops [12]. if instruction S executes before T and generates some data which instruction T will use it somehow, T is called a data dependent on S. 2)In stage two; In order to achieve better parallelization, the iteration space is tiled . A set of loop iterations running in one processor is called a tile [6]. In the third stage a desired parallel code is produced corresponding to the shape and size of produced tiles automatically for the iteration space of stage II. Finally in the stage four, the tiling space of the former stage is scheduled in a wave-front approach. Due to the fact that this is an NP-Hard Problem, applying definitive search methods does not seem to be reasonable for these kinds of problems.

In this paper an evolutionary approach is developed for tile scheduling in order to reduce the overall time needed for executing all solution tiles.

The remainder of this paper is organized as follow: in Section two Scheduling problem definitions was presented. In section three we review Genetic algorithms does work tools. Section Four explains related works in the context, in section five proposed approach was presented and in finally in Section six, the assessment and Practical results obtained from proposed approach was presented.

2. The problem

+ Alireza Hajieskandar. Tel.: +989144058474; fax: +984127230393.
E-mail address: **hajieskandar@gmail.com**.

In this section be paid to introduce the problem. An example of nested loops, the corresponding iteration space and its Tiled Iteration Space comes as in Figure 1. In this figure each point (i, j) stands for a nested loop iteration as the edges represent data dependencies as well.

```
for i := 0 to 5 do
  for j := 0 to 5 do
    A[i, j] := A[i − 1, j] + A[i, j − 1];
  end for
end for
```

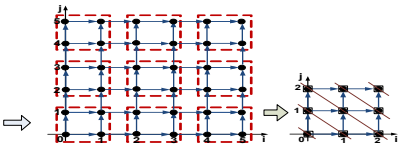


Fig. 1: nested loop, Iteration space for above and it's Tiled Iteration Space

The aim (goal) is allocate a certain number of processors to the existing tiles in the tiling space, so the inputs of our problem are the tiled iteration space and number of processors. In this tiled space, each tile $J^S=\left(J_1^S,J_2^S\right)$ have two prerequisite tiles as $J'^S=\left(J_1^S-1,J_2^S\right)$ and $J''^S=\left(J_1^S,J_2^S-1\right)$. This means that while the execution of prerequisite tiles has not been finished completely, J^S can not start execution. The cost of connection and message sending between the current tile and the prerequisite ones was shown by $V_{comm}(1)$ and $V_{comm}(2)$ respectively. Where, V_{comm} (1) and V_{comm} (2) is the volume of communication passing through an edge of the tile J^S with the neighboring tile in respect for the first dimension, $\left(J'^S\right)$ and second dimension $\left(J''^S\right)$. As long as J^S and its neighboring tiles are executed over different processors we have to tolerate the existing costs, but if the processors be identical the connection cost will be zero.

Regarding the fact that kind of connections among processors are well-made in style, the connection cost for each and every processor is the same. On the other hand, given that the required time duration allocated for any tile execution is identical, the overall time needed for finishing J^S - tile execution can be achieved through this equation:

$$complerion\,time(J^s)=Max(complerion\,time(J'^s)+V_{comm}(1),complerion\,time(J''^s)+V_{comm}(2)) \quad (1)$$

The aggregate time duration for executing all current scheduled tiles, known as "makespan", is given by the following equation:

$$makespan=\max(complerion\,time(J^s))\ ,J^s\in Tiledspace \quad (2)$$

The underling objective for scheduling tiling space is to find an allocation plan for available processors to tiles which have the shortest makespan time duration.

3. Genetic algorithms, an overview

An Evolutionary algorithm including genetic algorithms are stochastic search methods inspired by the Darwin. In this procedure, individuals caring better solutions have higher possibilities to appear in next generations. This algorithm like many others consists of several related components [4 and 16]; from which most notable ones are: Chromosome, genetic population, fitness function, genetic operators and parameters of genetic algorithms. In performing genetic algorithm some chromosomes are selected as parents to produce next generation chromosomes using genetic operators [3]. After selecting parents from the current population they are inserted into the intermediate population. After applying genetic operators (Crossover and Mutation) over them the new population of chromosomes appears to survive as they are being inserted in the next generation, and then insert these new generations into the population by replacement operator [15].

4. Related Works

Various methods have been proposed for shortening time duration needed for the execution of parallel iteration loops [1, 5, 8, 10, 11, 14 and 17]. Most of these methods which concentrate on optimum scheduling stem on the original hyper-plane approaches [9]. The main objective of this method is to classify computations through using linear conversions into well-defined groups which are known as wave-fronts or hyper-planes. Some of these methods eliminate many inter-processor communications and allocate dependent tiles to a unique processor [10]. In some other methods the time duration needed for inter-processor connections is overlapped with the time required by processors' internal computations [8].

In [2] four methods of the “cyclic assignment”, the “mirror assignment”, The “cluster assignment” and the “retiling” for scheduling the tiling iteration space in a clustered system with the fixed number of SMP[1] nodes are prepared and developed over each of them two schemes including both the overlapping execution scheme and The non-overlapping execution scheme is implemented.

Parsa and Lotfi in [13] have suggested a new executable algorithm for n-dimensional non-rectangular tiles in irregular iteration spaces for production and scheduling of parallel wave-fronts. The have taken it for granted that all tiles with identical features are located in one wave-front in order to extract efficient parallel tiles.

5. Our proposed Strategy

Before description the proposed algorithm, to the wave-front concept explanation and its use in algorithm be paid.

5.1. What the Wave-front means?

A Wave-front contains a set of tiles that don’t have interdependencies and can be run in parallel. But these wave-fronts are executed sequentially in respect to each other [1 and 13]. Number of Wave-front is equal to i+j. Figure 2 shows the wave-front concept, and its number for the tiled space of figure 1.

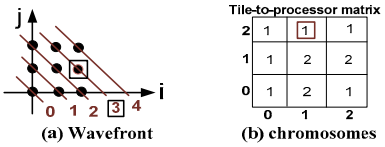


Fig. 2: Concept of wave-front, wave-front number and chromosomes

5.2. Encoding

In this way, available chromosomes do appear in the form of a matrix that is called the Tile-to-processor assignment matrix. This matrix shows the processor assignment to the available tiles. For this purpose, consider the part b of figure 2.

5.3. Fitness function

In genetic algorithm Fitness function is an indicator for chromosome survival. Due to the fact that the objective in the literature is to reduce the execution time of all tiles, so the Fitness function F(s) for the mentioned problem is defined as equation No.3, and The pseudo code for this function is depicted in figure 3.

$$f(s)=\frac{1}{makespan(s)} \qquad (3)$$

SchedulingCompletionTime=0;
Foreach Wavefront no., wn, lwn≤wn≤hwn **do Foreach** Processor Pi, $0 \le i \le N-1$ **do** ProcessingTime(Pi)=0;
  Foreach tile $j^s=(j_1^s,j_2^s)$ on wavefront wn such that $j_1^s+j_2^s=$ wn **do**
    Find Processor no., Pi, assigned to j^s; Assume $j^s=(j_1^s-1,j_2^s)$ and $j''^s=(j_1^s,j_2^s-1)$;
    Find Processor no., Pi' and Pi'' assigned to j'^s and j''^s, respectively;
    ProcessingTime(Pi)+ = V_{comp} .T_{comp} :computation time of the tile;
    (1)if j^s is only dependent on j'^s **then if** $Pi \neq Pi'$ **then** ProcessingTime(Pi)+= $V_{comm}(1).T_{comm}$:communication time1
    (2)**else if** j^s is only dependent on j''^s **then if** $Pi \neq Pi''$ **then** ProcessingTime(Pi)+= $V_{comm}(2).T_{comm}$:communication time2
    (3)**else if** j^s is dependent on j'^s and j''^s **then**
      if $Pi' \neq Pi''$ **then if** $Pi \neq Pi', Pi''$ **then** ProcessingTime(Pi)+= Max($V_{comm}(1).T_{comm}$, $V_{comm}(2).T_{comm}$)
      else if $Pi \neq Pi', Pi''$ **then** ProcessingTime(Pi)+= $V_{comm}(1).T_{comm}$ + $V_{comm}(2).T_{comm}$
  ProcessingTime(wn)=Max(ProcessingTime(Pi)):$\forall Pi \in [0..N-1]$; SchedulingCompletionTime+= ProcessingTime(wn);

Fig. 3: Pseudo code for Fitness Function

5.4. Selection Operator

For selecting parent chromosome the Roulette wheel selection and the Tournament selection methods have been used.

5.5. Crossover Operator

For crossing the selected pairs with each other the K-point crossover method or the uniform crossover method is used. The K-point crossover can be completed through rows (horizontal) or through columns (vertical). The engaged crossover operator for K-point crossover method incorporates two stages. In the first

[1] Symmetric Multi-Processors

stage, row-based approach or column based one is selected randomly to complete the crossover operation. In the next stage, the selected operator operates as planned. The pseudo-code for selecting row or column based approach is shown in fig.4.

```
1. Function crossover (Ap1, Ap2):(Ac1, Ac2)   // randomly select one of forms for crossover: column or row
2.   p←a randomly real number in the range [0, 1]
3.   if p<0.5 then Ac←HorizontalCrossover(Ap1, Ap2) // crossover the assignment matrix in row form
4.   else Ac←VerticalCrossover(Ap1, Ap2) // crossover the assignment matrix in column form
```

Fig. 4: The pseudo-code for selecting row or column crossover operator.

5.6. Mutation Operator

The mutation operator is applied through using either the bit-flipping or gene-swap method. In bit-flipping method, one of entries from the allocation matrix is selected stochastically to substitute for a digit ranging from 1 to N. (N, total number of processors present), and in swapping genes approach, two entries are selected randomly from the allocation matrix and their contents are interchanged.

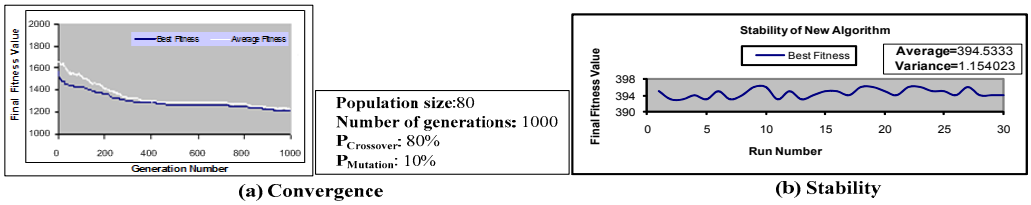
6. Assessment and practical results

In this section outcomes arise from implementing the offered algorithm is assessed (GALS[2]) and compared to those of alternative algorithms in the context. Algorithms are rendered in Delphi 7. Algorithms used for comparison are the BlockH[3], the BlockV[4], the CyclicH[5] and the CyclicV[6] as well. These Algorithms in [10] were described.

The comparison of algorithm is made from the view points of convergence toward the optimum solution, the stability of produced solutions and the quality of them.

To conduct a comparison on the convergence, a problem in the size of 30×30 with 8 processors is scheduled using this new algorithm. Part a of Figure 7 shows the best fitness and the average amount for each generation. This diagram shows how the offered algorithm converges at some points.

For surveying the stability of the proposed algorithm, we ran it 30 times for a sample problem, say 10×10-dimention one, and incorporate the best fitness number into the diagram. The results are shown in part b of Figure 7. As apparent in the figure the achieved results using different runs of the algorithm are close to each other. This points out that the proposed algorithm is a stable algorithm.



(a) Convergence (b) Stability

Fig. 7: Convergence and Stability of new algorithm

To conduct the next test on the quality of the produced solutions and to survey the efficiency of it, the offered algorithm as well as traditional alternative algorithm were implemented according to different sizes and a variety of dimensions. The results are summarized in the following tables (Fig 12). Inevitable that the Experiments are sensible which in them V_{comp} than V_{comm} be upside, Otherwise the tilde step not exactly done. Because of the variety and too Experiments, some of them are bring here. Some precise attention to the results reveals that the bipartite genetic algorithm culminates at better results relative to the other algorithms, so it is an efficient algorithm.

In below table, Parameters means: UB: Upper Bound, NP: Number of Processors, V_{comp}: Volume of Computation, V_{comm}: Volume of Communication, PS: Population Size, NG: Number of Generation, PM: Mutation Probability, PC: Crossover Probability.

TABLE I. THE RESULT OF TESTS

[2] Genetic Algorithm for loop scheduling
[3] Horizontal Blocks Algorithm
[4] Vertical Blocks Algorithm
[5] Horizontal Cyclic Algorithm
[6] Vertical Cyclic Algorithm

Number Of Experiments	Input Parameters										Makespan Of Algorithms					
	UB1	UB2	NP	Vcomp	Vcom1	Vcom2	PS	NG	PM	PC	BlockV	BlockH	CyclicV	CyclicH	GALS	
															AVG	Best
1	14	14	3	10	1	1	80	1000	0.1	0.8	1266	1266	930	930	914	913
2	9	9	3	9	2	5	80	1000	0.2	0.7	762	780	650	800	620	618
3	3	7	2	2	8	10	80	1200	0.2	0.8	102	92	164	196	82	64
4	8	8	3	9	7	5	80	1200	0.2	0.8	475	455	507	447	422	418
5	2	7	3	6	8	8	80	1200	0.2	0.8	166	150	168	208	133	128
6	14	5	2	10	1	1	80	1000	0.2	0.7	814	553	525	525	501	499
7	5	5	2	2	8	10	80	1200	0.2	0.8	96	108	186	222	78.4	72
8	9	5	3	9	5	7	80	1200	0.2	0.8	413	329	326	370	317	305

7. Conclusions and future works

In this paper an evolutionary algorithm is suggested for scheduling the iteration space of tow level nested loops. The comparisons were made from three view points of convergence toward the optimum solution, the quality of produced solutions and the stability characteristics of them. The achieved results highlight the fact that the new proposed algorithm achieves better solutions in 78% cases relative to other alternative algorithms. Another advantage of this algorithm is its stability as the yielded solutions by this algorithm are very similar and coherent with a low standard deviation.

The future work can focus on the overgeneralization of the algorithm toward problems with higher dimensions, the parallel implementation of genetic algorithm, incorporating other stochastic algorithms for addressing to the problem, as well as combining the genetic algorithm with the learning automatic.

8. References

[1] T. Andronikos, M. Kalathas, F. M. Ciorba, P. Theodoropoulos, and G. Papakonstantinou. An Efficient Scheduling of Uniform Dependence Loops. *Athens*, *Greece*, 2004, pp. 1-10.

[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] T. Bak. Evolutionary Algorithms in Theory and Practice. *Oxford University*, 1996.

[4] R. Bianchini and Ch. Brown. Parallel Genetic Algorithms on Distributed Memory Architecture. *Prentice-Hall*, 1993.

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

[6] A. Darte, G. A. Silber, and F. Vivien. Combining retiming and scheduling techniques for loop parallelization and loop tiling. *Ecole Normale Supérieure de Lyon Unité de recherche associée au CNRS n°1398*, November 1996.

[7] R. Eigenmann and J. Hoeflinger. Parallelizing and Vectorizing Compilers. *Proceedings of the IEEE*, January 2002.

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

[9] L. Lamport. The parallel execution of DO loops. *Comm. of the ACM*, Vol. 37, No. 2, February 1974, pp. 83-93.

[10] Sh. Lotfi and S. Parsa. Parallel Loop Generation and Scheduling. *Journal of Supercomputing*, February 2009.

[11] M. Obitko. Genetic Algorithms. 1998.

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

[13] S. Parsa and Sh. Lotfi. Wave-Fronts Parallelization and Scheduling. *IEEE, 4th International Conference on Innovations in Information Technology* (Innovations'07), Dubai, UAE, November 18-20, 2007, pp. 382-386.

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

[15] L. J. Pit. Parallel Genetic Algorithms. *Master thesis*, Leiden University, 1995.

[16] R. Poli, W.B. Langdon and N.F. Mcphee. A field guide to genetic programming. 2008.

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