Scheduling signal processing tasks for antenna arrays with simulated annealing

Daniil A. Zorin
Department of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
juan@lvk.cs.msu.su

Abstract — **The problem dealt with in this paper is the design of a parallel embedded system with the minimal number of processors. The system is designed to solve signal processing tasks using data collected from antenna arrays. Simulated annealing algorithm is used to find the minimal number of processors and the optimal system configuration.**

Keywords — optimization, scheduling, hardware design, embedded systems, simulated annealing

I. INTRODUCTION

Antenna array is hardware used to collect data from the environment, it is often employed in areas such as radiolocation and hydroacoustics [1]. Radiolocation tools have to process signals with a fixed frequency and have hard deadlines for the data processing time. At the same time, the size of the antenna array is limited, therefore, in order to maintain high accuracy, algorithms with significant computational cost have to be used to process signals. The co-design problem of finding the minimal necessary number of processors and scheduling the signal processing tasks on it arises in this relation. This paper suggests the application of simulated annealing algorithm to this problem. The purpose of this work is to show how the simulated annealing algorithm (discussed, for instance, in [2] and [3]) can work with real-world industrial problems.

In Section 2 we define the problem of scheduling for systems with antenna arrays and show the structure of signal processing algorithms used in such systems. It is explained how this problem can be formulated in the way that allows to use simulated annealing. The simulated annealing algorithm itself is discussed in Section 3. Experimental results obtained with the algorithm are given in Section 4.

II. PROBLEM FORMULATION

Systems used in radiolocation and hydroacoustics use a set of sensors to collect data from the environment. This set is called antenna array, and it is the most valuable and complicated part of the system. The size of the antenna array is fixed, so it is preferable and cheaper to build the system with a smaller antenna array and effective performance.

The problem solved with the antenna array system is finding the coordinates of the source of the signal. Traditional signal processing algorithms are based on fast Fourier transforms (FFT). However, their potential solution capabilities are limited by the sizes of the antenna array. With a small array, the FFT will be performed on a small set of points, which can lead to low accuracy. Alternative methods based on automatic interference filtration [4] and on correlation matrix expansion (also shortened to CME) [5] can give accurate results even with smaller antenna arrays. They use several samples collected with the small array over a period of time to get very precise solutions. However, their computational complexity is significantly higher.

Assume that the antenna array has K elements (sensors) and works in frequency interval (-B, B). The interval is split into L parts, and each of them is processed separately until the final result is computed on the last stage of the algorithm. We also need the number of support vectors used in CME method, M_0. The sampling frequency is $1/\tau=aB$, where $a\geq 2.5$ is the coefficient in the Kotelnikov-Nyquist theorem. The system waits until a sample of n points is collected from the sensors, and then starts the processing algorithm. Therefore, the execution deadline is n/τ, the time until the new sample is collected. If the deadline is broken, the sensors' buffer will overflow.

Stage	Name	Complexity	Input size	Output size
1	Normalization	$O(aL)$	-	aL
2	FFT	$O(aL \cdot \log_2 aL)$	aL	1
3	Vector multiplication	$O(K^2)$	1	K^2
4	Computing eigenvalues, matrix reversal	$O(K^3)$	K^2	K^2
5	Computing signal source coordinates	$O(K)$	K^2	K
6	Vector multiplication	$O(K)$	K	1
7	Vector comparison	$O(K)$	1	K
8	Vector comparison	$O(LK)$	K	-

Table 1. Steps of the CME method

The steps of the algorithm, their respective computational complexities and the size of the data processed on each step are shown in Table 1. For simplicity, all figures are given only for the CME method. However, the general scheme of the automatic filtration method is the same, the difference lies on step 5, where the complexity becomes $O(K^2)$.

The signal processing runs on a multiprocessor system. It is assumed that processors are identical. Processors have fixed clock rate and reliability. The processors are interconnected, data transfer rate is fixed.

The workflow of the program is shown in Figure 6. The nodes represent subprograms and the edges represent dependencies between them. First, preprocessing, including FFT, is applied to the collected data, and the corresponding nodes are in the topmost row. They implement steps 1-3 from Table 3. The number of nodes is the same as K, the size of the antenna array. Then all data is sent to each of the L subprocesses and CME is performed. The CME can be divided into steps; each step implies some operations on the matrix (nodes CME_i_stage_j, where i runs from 1 to L, and j enumerates the stages). In Figure 6, there are three stages that correspond to steps 4 and 5 in Table 1. In the latter half the CME is broken into M_0 parallel threads for step 6 and joined into one thread on step 7 (nodes CME_i_pstage_j_k, where i runs from 1 to L, j runs from 1 to M_0, and k enumerates the stages). Then all data is collected for final processing on step 8 (CME_final).

All nodes perform simple computations with matrices and vectors, such as FFT or matrix multiplications, so the complexity of each subprogram (also called 'task' hereafter) is known. Since processor performance is known, it is possible to calculate the execution time of each node, as well as the amount of data sent between the nodes. So, we reach the following mathematical problem statement [2].

The signal processing program can be represented with its data flow graph G = {V, E}, where V is the set of vertices (corresponding to the tasks) and E is the set of edges (corresponding to the dependencies of the tasks). Each vertex is marked by the time of execution of the corresponding task and each edge is marked by the time of data transfer. The set of processors denoted by M is given.

Processor redundancy implies adding a new processor to the system and using it to run the same tasks as on some existing processor. In this case the system fails if both processors fail. The additional processor is used as hot spare, i.e. it receives the same data and performs the same operations as the primary processor, but sends data only if the primary one fails. With switch architecture used, this does not cause any delays in the work of the system.

A schedule for the program is defined by task allocation, the correspondence of each task with one of the processors, and task order, the order of execution of the task on the processor. Formally, a schedule is defined as a pair (S, D) where S is a set of triplets (v, m, n) where $v \in V$, $m \in M$, $n \in \mathbb{N}$, so that $\forall v \in V$: $\exists! s=(v_i, m_i, n_i) \in S: v_i=v$; and $\forall s_i=(v_i, m_i, n_i) \in S$, $\forall s_j=(v_j, m_j, n_j) \in S$: $(s_i \neq s_j \wedge m_i=m_j) \Rightarrow n_i \neq n_j$.

D is a multiset of elements of the set of processors. Substantially m and n denote the placement of the task on a processor and the order of execution for each version of each task. The multiset D denotes the spare processors: if processor m has k spares, it appears in D k times.

A schedule can be represented with a graph. The vertices of the graph are the elements of S. If the corresponding tasks are connected with an edge in the graph G, the same edge is added to the schedule graph. Additional edges are inserted for all pairs of tasks placed on the same processor right next to each other.

According to the definition, there can be only one instance of each task in the schedule, and all tasks on any processor have different numbers. Besides these, one more limitation must be introduced to guarantee that the program can be executed completely. A schedule S is correct by definition if its graph has no cycles. Otherwise the system would reach a deadlock where two processors are waiting for data from each other forever.

For every correct schedule the following functions are defined: t(S) – time of execution of the whole program, R(S) – reliability of the system, M(S) – the number of processors used.

Given the program G, *tdir*, the hard deadline of the program, and *Rdir*, the required reliability of the system, the schedule S that satisfies both constraints ($t(S) < tdir$ and $R(S) > Rdir$) and requires the minimal number of processors is to be found.

Theorem 1. The optimization problem formulated above is NP-hard.

III. SIMULATED ANNEALING ALGORITHM

The proposed algorithm of solution is based on simulated annealing [6]. For simplicity, the model used in this study does not consider software reliability, so operations and structures related to that are omitted here. This does not affect the algorithm's performance because it simply works as if the software reliability is always maximal.

The following three operations on schedules are used.

Add spare processor and *Delete spare processor*. Adds or removes a hot spare to the selected processor.

Move vertex. This operation changes the order of tasks on a processor or moves a task on another processor. It is obligatory to make sure that no cycles appear after this operation. The analytic form of the necessary and sufficient condition of the correctness of this operation is given in [4].

Theorem 2. If A and B are correct schedules, there exists a sequence of operations that transforms A to B such that all interim schedules are correct.

Each iteration of the algorithm consists of the following steps:

Step 1. Current approximation is evaluated and the operation to be performed is selected.

Step 2. Parameters for the operation are selected and the operation is applied.

Step 3. If the resulting schedule is better than the current one, it is accepted as the new approximation. If the resulting schedule is worse, it is accepted with a certain probability.

Step 4. Repeat from step 1.

The number of iterations of the algorithm is pre-determined.

If the reliability of the system is lower than required, spare processors and versions should be added, otherwise they can be deleted. If the time of execution exceeds the deadline the possible solutions are deleting versions or moving vertices. The selection of the operation is not deterministic so that the algorithm can avoid endless loops.

When the operation is selected, its parameters have to be chosen. For each operation the selection of its parameters is nondeterministic, however, heuristics are employed to help the algorithm move in the direction where the new schedule is more likely to be better.

The selection of the operation is not deterministic so that the algorithm can avoid endless loops. The reliability limit and the deadline can either be satisfied or not. Probability of selecting each operation, possibly zero, is defined for each of the four possible situations depending on the time and reliability constraints (*tdir* and *Rdir*): both constraints satisfied, both constraints not satisfied, reliability constraint is satisfied while the time constraint is not, and vice versa. These probabilities are given before the start of the algorithm as its settings.

Some operations cannot be applied in some cases. For example, if none of the processors have spare copies it is impossible to delete processors and if all versions are already used it is impossible to add more versions. Such cases can be detected before selecting the operation, so impossible operations are not considered.

When the operation is selected, its parameters have to be chosen.

Add spare processor. Processors with fewer spares have higher probability of being selected for this operation.

Delete spare processor. A spare of a random processor is deleted. The probability is proportional to the number of spare processors.

The probabilities for these operations are set with the intention to keep balance between the reliability of all components of the system.

Move vertex. If $t(S) < tdir$, the main objective is to reduce the number of processors. With a probability of *pcut* the following operation is performed: the processor with the least tasks is selected and all tasks assigned to it are moved to other processors. With a probability of 1-*pcut* the movement of a task is decided by one of the three strategies described below.

If $t(S) > tdir$, it is necessary to reduce the time of execution of the schedule. It can be achieved either by moving several tasks to a new processor or reallocating some tasks. The parameters for the operation are chosen according to one of the three strategies: delay reduction, idle time reduction or mixed.

Delay reduction strategy. The idea of this strategy emerges from the assumption that if the time of the start of each task is equal to the length of the critical path to this task in graph G, the schedule is optimal. The length of the critical path is the sum of the lengths of all the tasks forming the path and it represents the earliest time when the execution of the task can begin.

For each element *s* it is possible to calculate the earliest time when *s* can start, i.e. when all the tasks preceding the current one are completed. The difference between this time and the moment when the execution of s actually starts according to the current schedule is called the delay of task *s*. If some task has a high delay, it means that some task preceding it is blocking its work, so the task before the one with a high delay has to be moved to another processor.

The task before the task with the highest delay is selected for Move Vertex operation. If the operation is not accepted, on the next iteration the task before the task with the second highest delay is selected, and so on. The position (pair (m, n) from the triplet) is selected randomly among the positions where the task can be moved without breaking the correctness condition.

Figure 1 gives an example of delay reduction. Task 3 does not depend on task 4, so moving task 4 to the first processor reduces the delay of task 3, and the total time decreases accordingly.

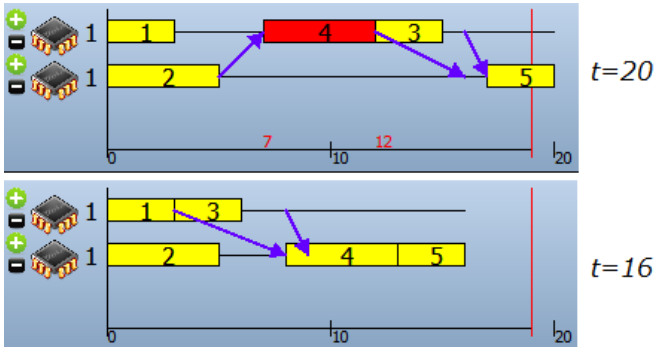

Figure 1. Delay reduction strategy

Idle time reduction strategy. This strategy is based on the assumption that in the best schedule the total time when the processors are idle and no tasks are executed due to waiting for data transfer to end is minimal.

For each position (m, n) the idle time is defined as follows. If n=1 then its idle time is the time between the beginning of the work and the start of the execution of the task in the position (m, 1). If the position (m, n) denotes the place after the end of the last task on the processor m, then its idle time is the time between the end of the execution of the last task on m and the end of the whole program. Otherwise, the idle time of the position (m, n) is the interval between the end of the task in (m, n-1) and the beginning of the task in (m, n).

The task to move is selected randomly with higher probability assigned to the tasks executed later. Among all positions where it is possible to move the selected task, the position with the highest idle time is selected. If the operation is not accepted, the position with the second highest idle time is selected, and so on.

The idle time reduction strategy is illustrated in Figure 2. The idle time between tasks 1 and 4 is large and thus moving task 3 allows reducing the total execution time.

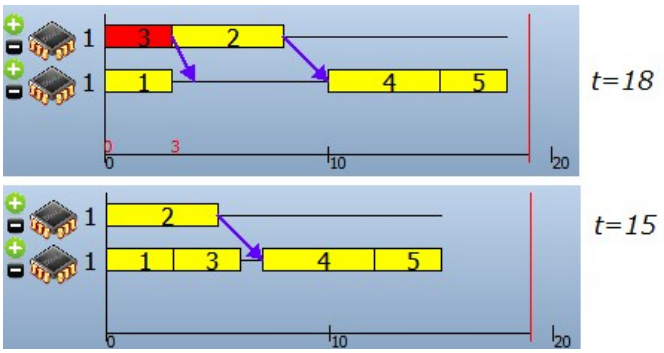

Figure 2. Idle time reduction strategy

Mixed strategy. As the name suggests, the mixed strategy is a combination of the two previous strategies. One of the two strategies is selected randomly on each iteration. The aim of this strategy is to find parts of the schedule where some processor is idle for a long period and to try moving a task with a big delay there, prioritizing earlier positions to reduce the delay as much as possible. This strategy has the benefits of both idle time reduction and delay reduction, however, more iterations may be required to reach the solution.

After performing the operation a new schedule is created and time, reliability and number of processors are calculated for it. Depending on the values of these three functions the new schedule can be accepted as the new approximation for the next iteration of the algorithm. Similar to the standard simulated annealing algorithm, parameter d modeling the temperature is introduced. Its initial value is big and it decreases after each iteration.

The probability to accept a worse schedule on step 3 depends on the parameter called temperature. This probability decreases along with the temperature over time. Temperature functions such as Boltzmann and Cauchy laws [7] can be used as in most simulated annealing algorithms

Theorem 3. If the temperature decreases at logarithmic rate or slower, the simulated annealing algorithm converges in probability to the stationary distribution where the combined probability of all optimal results is 1.

IV. EXPERIMENTS

Figure 7 shows the solution found by the algorithm for the problem shown on Figure 6. The system has been successfully reduced to 4 processors.

In real systems the size of the array is a power of 2, usually between 256 and 1024 (radiolocation systems use smaller arrays), and the number of frequency intervals (L) is a power of 2, usually 32 or 64. For evaluation purposes, other values of L were tested as well. The value of M_0 is normally between 1 and 4.

In general, the majority of computations are performed after the initial processing on the K antennas and constitute the $L*M_0$ parallel sequences of nodes in the program graph, Therefore, the quality of the algorithm can be estimated by comparing the number of processors in the result with the default system configuration where $L*M_0$ processors are used. The following graphs (Figures 3-5) show the quotient of these two numbers, depending on L, for radiolocation problem. Lower quotient means better result of the algorithm.

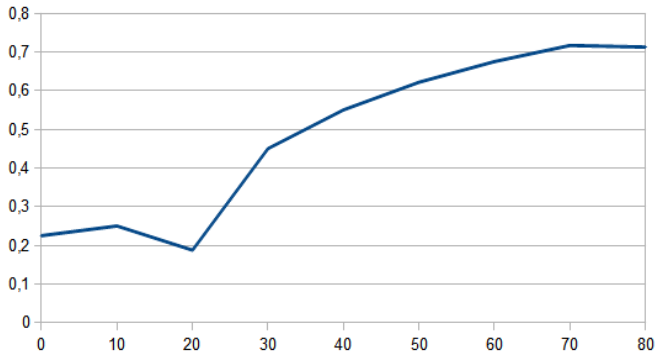

Figure 3. Optimization rate, M_0=2

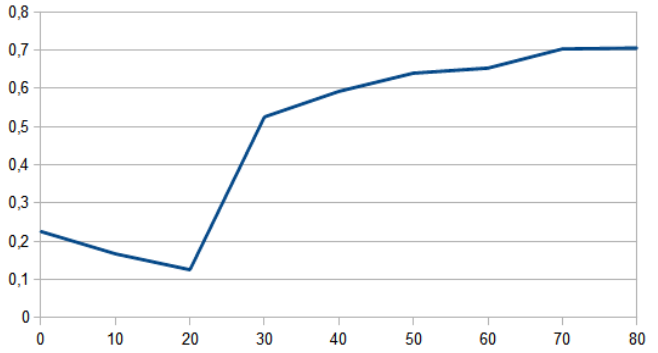

Figure 4. Optimization rate, M_0=3

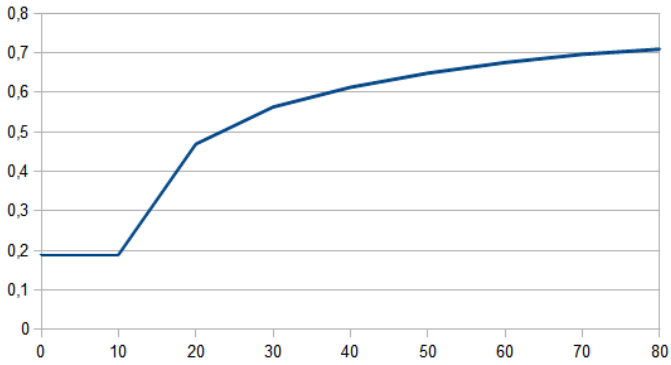

Figure 5. Optimization rate, M_0=4

As we can see, the algorithm optimizes the multiprocessor system by at least 25% in harder examples with many parallel tasks, and by more than a half in simpler cases.

CONCLUSIONS

Experiments with our tool testify that scheduling for antenna arrays can be done effectively with simulated annealing. The experimental data shows that the size of the system can be optimized by 25-30% without breaking deadlines and limits of reliability.

REFERENCES

[1] Kostenko V.A. Design of computer systems for digital signal processing based on the concept of ``open'' architecture.//Automation and Remote Control. – 1994. – V. 55. – №. 12. – P. 1830-1838.

[2] D. A. Zorin and V. A. Kostenko Algorithm for Synthesis of Real-Time Systems under Reliability Constraints // Journal of Computer and Systems Sciences International. 2012. Vol. 51. No. 3. P. 410–417.

[3] Daniil A. Zorin, Valery A. Kostenko. Co-design of Real-time Embedded Systems under Reliability Constraints // Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems (PDeS). Brno, Czech Republic: Brno University of Technology, 2012. P. 392-396

[4] Monzingo R. A., Miller T. W. Introduction to adaptive arrays, 1980 //Wiley New York. – P. 56-63.

[5] Widrow B., Stearns S. D. Adaptive signal processing //Englewood Cliffs, NJ, Prentice-Hall, Inc., 1985, 491 p. – 1985. – V. 1.

[6] Kalashnikov, A.V. and Kostenko, V.A. (2008). A Parallel Algorithm of Simulated Annealing for Multiprocessor Scheduling. Journal of Computer and Systems Sciences International, 47, No. 3, pp. 455-463.

[7] Wasserman F. Neurocomputer Techniques: Theory and Practice [Russian translation] //Mir, Moscow. – 1992. – 240 p.

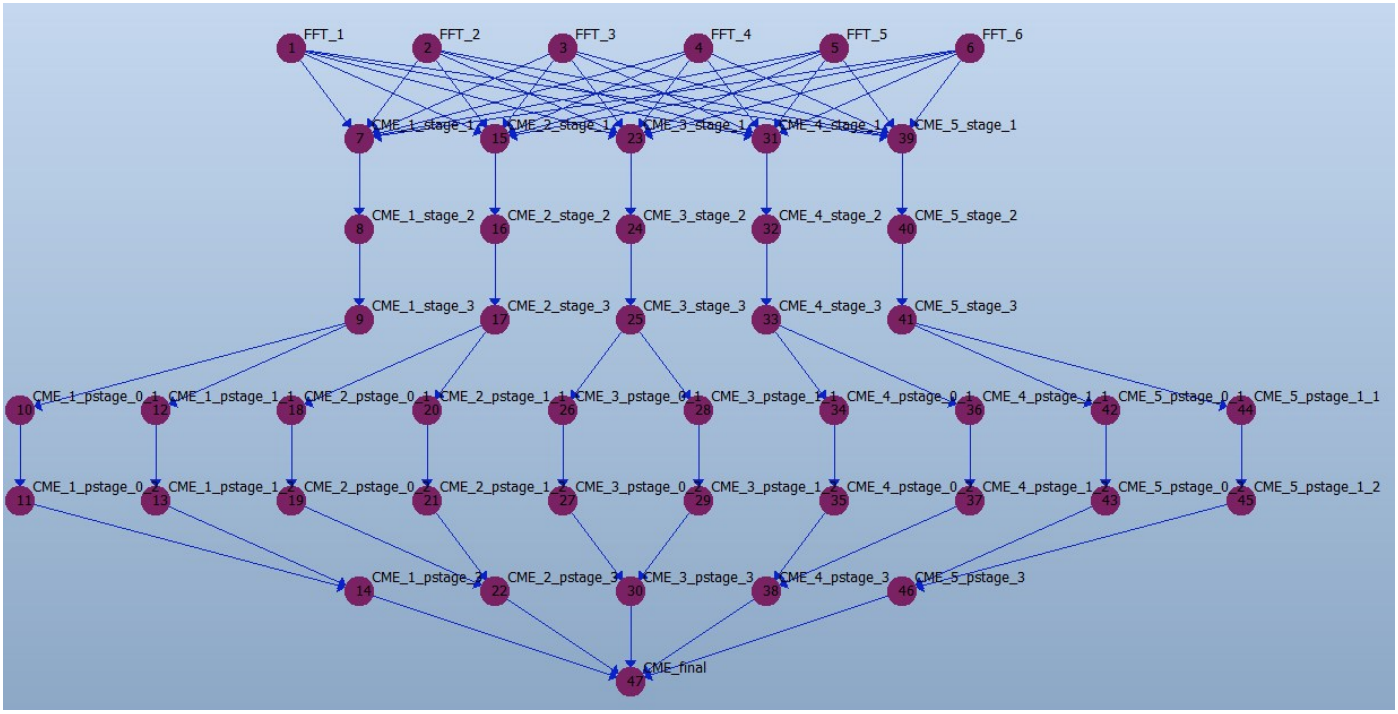

Figure 6. Signal processing workflow

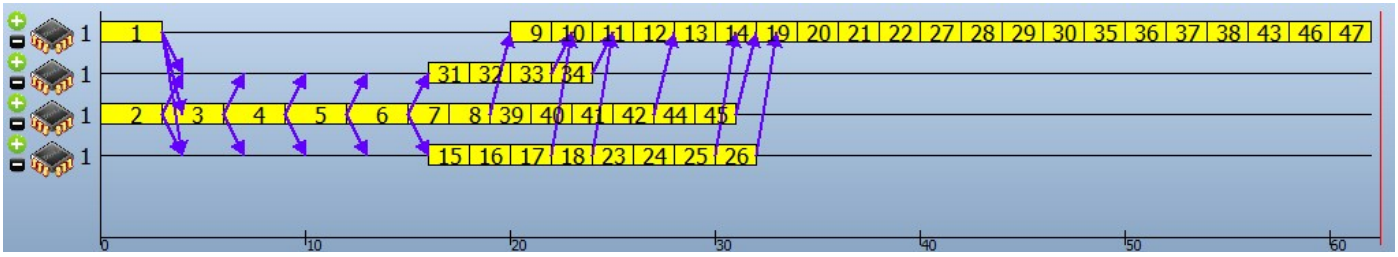

Figure 7. Schedule for the program from Figure 6