

LOOP SCHEDULING TO MINIMIZE COST WITH DATA MINING AND PREFETCHING FOR HETEROGENEOUS DSP

Meikang Qiu¹ Zhiping Jia² Chun Xue¹ Zili Shao³ Ying Liu¹ Edwin H.-M. Sha¹ *

ABSTRACT

In real-time embedded systems, such as multimedia and video applications, cost and time are the most important issues and loop is the most critical part. Due to the uncertainties in execution time of some tasks, this paper models each varied execution time as a probabilistic random variable. We propose a novel algorithm to minimize the total cost while satisfying the timing constraint with a guaranteed confidence probability. First, we use data mining to predict the distribution of execution time and find the association rules between execution time and different inputs from history table. Then we use rotation scheduling to obtain the best assignment for total cost minimization, which is called the HAP problem in this paper. Finally, we use prefetching to prepare data in advance at run time. Experiments demonstrate the effectiveness of our algorithm. Our approach can handle loops efficiently. In addition, it is suitable to both soft and hard real-time systems.

KEY WORDS

Probability, scheduling, data mining, prefetch, heterogeneous

1 Introduction

In high level synthesis, Cost (such as energy, reliability, etc.) minimization has become a primary concern in today's real-time embedded systems. In DSP systems, some tasks may not have fixed execution time. Such tasks usually contain conditional instructions and/or operations that could have different execution times for different inputs. It is possible to obtain the execution time distribution for each task by sampling or profiling [1]. In this paper, we use a data mining engine to do the prediction. First, the data mining engine collects data into the log. Then do clustering and use unsupervised method to find the distribution pattern of all random variables, i.e., the execution times. Finally, the engine builds the distribution function for each execution time that has uncertainty.

In heterogeneous DSP, i.e., there are multiple functional unit (FUs) type for each task to choose from. Different FU type has different execution time and cost, which may relate to energy, area, etc. A certain FU type

may execute the task slower but with less cost, while another type will execute faster with higher cost. Prior design space exploration methods for hardware/software codesign of embedded systems [2] guarantee no deadline missing by considering worst-case execution time of each task. These methods are pessimistic and will often lead to over-designed systems with high cost. In this paper, we use probabilistic approach and loop scheduling to avoid over-design systems. We compute the best type assignment at compile time to minimize expected value of total energy consumption while satisfying timing constraints with guaranteed probabilities for real-time applications.

We design new rotation scheduling algorithms for real-time applications that produce schedules consuming minimal energy. In our algorithms, we use rotation scheduling [3] to get schedules for loop applications [4, 5]. The schedule length will be reduced after rotation. Then, we assign different FU to computations individually in order to decrease the cost of processors as much as possible within the timing constraint.

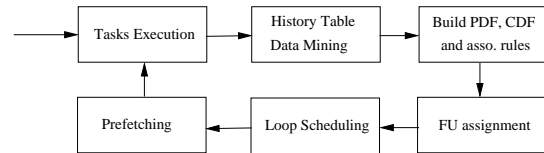


Figure 1. The basic implementation steps of our method

In summary, our approach includes three steps, which are shown in Figure 1. First, we use data mining to predict the execution time pattern and time-input association from history table. Second, we use rotation scheduling to obtain the best assignment for total cost minimization. This includes finding the best FU type assignment in each iteration and rotation scheduling for Q iterations. Finally, we use prefetching to prepare data in advance at run time. The experimental data show that our algorithms can get better results on cost saving than the previous work.

The rest of the paper is organized as following: In Section 2, we give motivational examples. The models and basic concepts are introduced in Section 3. In Section 4, we propose our algorithms. We give the related work in Section 5. The experimental results are shown in Section 6, and the conclusion is shown in Section 7.

2 Motivational Examples

For the data mining engine, the working procedures are as follows. We first build up a execution time history table (ETHT). Then implement data cleaning. Next, do data

*¹ M. Qiu, C. Xue, Y. Liu, and E. H. M. Sha are with the Dept. of CS, Univ. of Texas at Dallas, Richardson, Texas 75083, USA. Email: {mxq012100, cxx016000, ying.liu, edsha}@utdallas.edu. ² Z. Jia is with School of Computer Science and Technology, Shandong University, Jinan, Shangdong, China. Email: zhipingj@sdu.edu.cn. ³ Z. Shao is with the Dept. of Computing, Hong Kong Polytechnic Univ., Hung Hom, Kowloon, Hongkong. Email: cszshao@comp.polyu.edu.hk.

[†]This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, NSF IIS-0513669, Microsoft, USA, and HK POLYU A-PH13, HK

integration, transformation, and filtering (selection). Finally we find the association rule of the random variables and the pattern of distribution function of each execution time.

Data mining engine works at compile time. Based on the obtained distribution function of each execution time, we will use loop scheduling to find the best assignment for cost minimization. Then at run time, needed data will be prefetched in advance. Based on the computed best assignment, we can prefetch data in certain time ahead with guaranteed probability. For example, if node A select type F1, then we prefetch data for node B in 2 time unit in advance, and it will guarantee with 100% that node B can be executed on time with needed data.

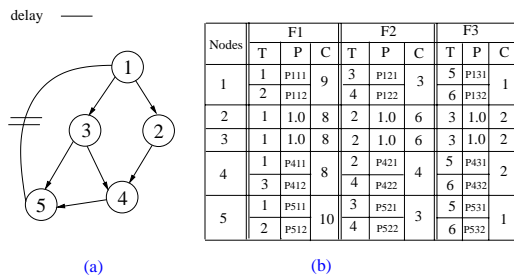


Figure 2. (a) A PDFG. (b) Parameter table

Assume an input PDFG (*Probability Data Flow Graph*) shown in Figure 2(a). Each node can select one of the three different FUs: F_1 , F_2 , and F_3 . The execution times (T), and expected cost (C) of each node under different FU types are shown in Figure 2(b). The input PDFG has five nodes. Node 1 is a multi-child node, which has two children: 2 and 3. Node 5 is a multi-parent node, and has two parents: 3 and 4. The execution time T of each node is modeled as a random variable. For example, When choosing R_1 , node 1 will be finished in 1 time unit with probability P_{111} and will be finished in 2 time units with probability P_{112} . The corresponding probabilities (P) to T are still unknown. P_{111} represents the first kind of variation of execution time for node 1 under type F1.

Data mining engine is used to predict the probabilities. We first profile the data and build the historic table of execution time of nodes. Then we use data mining techniques to discover the distribution function of each execution time that is not fix. Figure 3 (a) shows the obtained distribution of each random variable. Base on (a), Figure 3 (b) computes the cumulative distributed function (CDF) of each random variable.

For initial schedule graph in Figure 4 (c), We use the *HAP_Heu* algorithm in [6] to generate the minimum total cost with computed confidence probabilities under the timing constraints. The results are shown in Table 1. Algorithm *HAP_Heu* [6] is used as a sub-algorithm of our *LSHAP* algorithm. The entries with probability that is equal to 1 (see the entries in boldface) actually give the results to the hard real-time problem which shows the worst-case scenario. For each row of the table, the C in each (P, C) pair gives the minimum total cost with confidence probability P under timing constraint j. For example, using our algorithm, at timing constraint 11, we

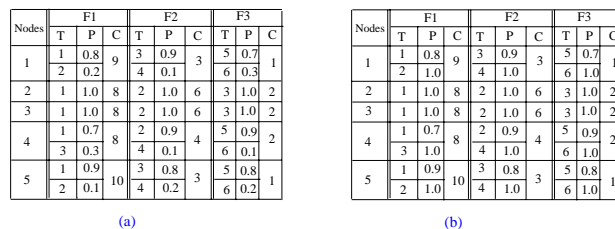


Figure 3. (a) Parameter table after data mining. (b) Parameter table with cumulative probabilities.

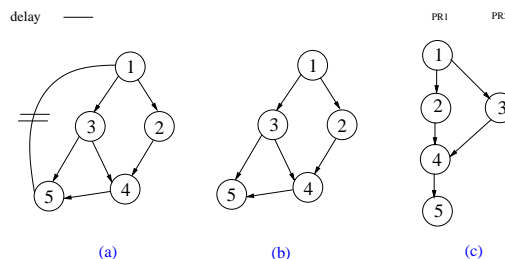


Figure 4. (a) Original PDFG. (b) The static schedule. (c) The schedule graph

can get (0.90, 20) pair. The assignments are shown as "Ass_1" in Table 2. Assignment $A(v)$ represents the voltage selection of each node v . Hence, we find the way to achieve minimum total cost 20 with probability 0.90 satisfying timing constraint 11. While using the ILP and heuristic algorithm in [2], the total cost obtained is 32. The assignments are shown as "Ass_2" in Table 2.

For the new schedule graph shown in Figure 5(c), we get (0.90, 10) and (1.00, 12) pairs at timing constraint 11. For (0.90, 10) pair, node 5's type was changed to F_3 , then the T was changed from 4 to 6, and cost change from 3 to 1. Hence the total execution time is 11, and the total cost is 10. So the improvement of cost saving is 50.0% while the probability is still 90%. For (1.00, 12) pair, the execution times of nodes 2, 3 were changed from 1 to be 3 and node 1's was changed from 2 to 6. The total

T	(P, C)	(P, C)	(P, C)	(P, C)	(P, C)
4	0.50, 43				
5	0.65, 39				
6	0.65, 35	0.81, 39			
7	0.65, 27	0.73, 33	0.81, 35	0.90, 39	
8	0.81, 27	0.90, 35	1.00, 43		
9	0.58, 20	0.73, 21	0.81, 27	0.90, 32	1.00, 39
10	0.72, 20	0.81, 21	0.90, 28	1.00, 36	
11	0.65, 14	0.90, 20	1.00, 32		
12	0.81, 14	0.90, 20	1.00, 28		
13	0.65, 12	0.90, 14	1.00, 20		
14	0.81, 12	0.90, 14	1.00, 20		
15	0.50, 10	0.90, 12	1.00, 14		
16	0.72, 10	0.90, 12	1.00, 14		
17	0.90, 10	1.00, 12			
18	0.50, 8	0.90, 10	1.00, 12		
19	0.72, 8	1.00, 10			
20	0.90, 8	1.00, 10			
21	1.00, 8				

Table 1. Minimum C satisfying T with P.

		Node id	T	F	P	C
Ass.1	A(v)	1	2	F ₁	1.00	9
		2	3	F ₃	1.00	2
		3	3	F ₃	1.00	2
		4	2	F ₂	0.90	4
		5	4	F ₂	1.00	3
Total			11		0.90	20
Ass.2	A(v)	1	2	F ₁	1.00	9
		2	1	F ₁	1.00	8
		3	1	F ₁	1.00	8
		4	4	F ₂	1.00	4
		5	4	F ₂	1.00	3
Total			11		1.00	32

Table 2. The assignments with $T = 11$.

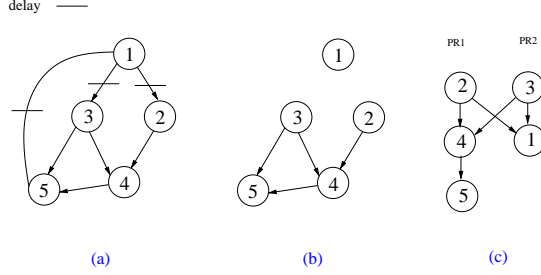


Figure 5. (a) The PDFG after retiming (b) The static schedule. (c) New schedule

cost were changed to 12, and the total execution time is still 11. Hence compared with original 32, the improvement of total cost saving is 62.5%. If we consider the cost of switch activity, we can get more practical results. For example, after rotation once, node 1 has changed from processor PR1 to PR2. Assume the cost of this switch is 1, then the final total cost is 13. Then the cost saving is 59.4% compared with previous scheduling and assignment.

3 Models and Concepts

In this section, we introduce the data mining and prefetching model, the system model, and the HAP problem.

Data Mining and Prefetching Model: Our data mining engine has several major steps [7], which is shown in Figure 6: First, We build up a execution time history table (ETHT). In each iteration, the engine will store the execution time of each node into the table. From ETHT, we do data cleaning, to remove noise and inconsistent data. Then data integration and transformation. Next do data mining in order to extract data pattern. Finally, we obtain the probability distribution function of each execution time and the association rule between inputs and execution time selection. For example, in the Figure 3, we obtained $P_{111} = 0.8$ and $P_{112} = 0.2$, which means node 1 with FU type 1 will be finished in 1 time unit with 80% probability and be finished in 2 time unit with 20% probability. We can also obtain the hidden association between different inputs and execution time. For instance, we may find the node 1 will always be fin-

ished in 2 time unit when the input is a large integer m , such as $m \geq 100$. We may not know why this happened, but we can use this hidden association and select the right execution time and FU type.

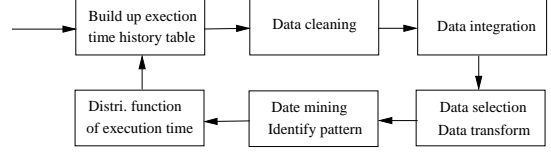


Figure 6. The basic structure of our method

For prefetching, suppose there is a on chip memory and a main memory. The on chip memory is small but fast; and the main memory is large but slow. Since the limited space of on chip memory, the engine will prefetch data that will be needed for later tasks based on the prediction of execution time of current task. For example, if the prefetching operation time is 2 cycles, and the current task A is estimated to be finished in 5 cycles from now, then the engine need to prefetch the data for the next tasks B 3 cycles before now.

System Model: *Probabilistic Data-Flow Graph* (PDFG) is used to model applications of embedded systems. A cyclic PDFG $G = \langle U, ED, d, T, F \rangle$ is a node-weighted and edge-weighted directed graph, where $U = \langle u_1, \dots, u_i, \dots, u_N \rangle$ is the set of nodes; $F = \langle F_1, \dots, F_j, \dots, F_M \rangle$ is a FU set; the execution time $T_{F_j}(u)$ is a random variable; $ED \subseteq U \times U$ is the edge set that defines the precedence relations among nodes in U . $d(ed)$ is a function to represent the number of delays for any edge $ed \in ED$, The edge without delay represents the intra-iteration data dependency; the edge with delays represents the inter-iteration data dependency and the number of delays represents the number of iterations involved. *Static Schedules:* From the PDFG of an application, we can obtain a static schedule. A static schedule of a cyclic PDFG is a repeated pattern of an execution of the corresponding loop. The DAG *directed acyclic graph* is obtained by removing all edges with delays in the PDFG.

An *assignment A* is a function from domain U to range F , where U is the node set and F is the FU set. For a node $u \in V$, $A(u)$ gives selected FU type of node u . In a PDFG G , $T_{F_j}(u)$, $1 \leq j \leq M$, represents the execution times of each node $u \in V$ when running at FU type F_j ; $T_{F_j}(u)$ is either a discrete random variable or a continuous random variable. We define $X(t)$ to be the *cumulative distribution function* (abbreviated as *CDF*) of the random variable $T_{F_j}(u)$, where $X(t) = P(T_{F_j}(u) \leq t)$. When $T_{F_j}(u)$ is a discrete random variable, the CDF $X(t)$ is the sum of all the probabilities associating with the execution times that are less than or equal to t . If $T_{F_j}(u)$ is a continuous random variable, then it has a *probability density function (PDF)*. If assume the PDF is f , then $X(t) = \int_0^t f(s)ds$. Function $X(t)$ is nondecreasing, and $X(-\infty) = 0$, $X(\infty) = 1$.

Retiming [8] is an optimal scheduling technique for cyclic DFGs considering inter-iteration dependencies. It can be used to optimize the cycle period of a cyclic PDFG

by evenly distributing the delays. Retiming generates the optimal schedule for a cyclic PDFG when there is no resource constraint. Given a cyclic PDFG $G=(U, ED, d, T, F)$, retiming r of G is a function from U to integers. For a node $u \in U$, the value of $r(u)$ is the number of delays drawn from each of incoming edges of node u and pushed to all of the outgoing edges. Let $G_r = (U, ED, d_r, T, F)$ denote the retimed graph of G with retiming r , then $d_r(ed) = d(ed) + r(u) - r(v)$ for every edge $ed(u \rightarrow v) \in ED$. *Rotation Scheduling* [3] is a scheduling technique used to optimize a loop schedule with resource constraints. It transforms a schedule to a more compact one iteratively in a PDFG. In most cases, the minimal schedule length can be obtained in polynomial time by rotation scheduling.

Definitions: Define the *HAP (heterogeneous assignment with probability)* problem as follows: Given M different FU levels: F_1, F_2, \dots, F_M , a PDFG $G = (U, ED, d, T, F)$ with $T_{F_j}(u)$, $P_{F_j}(u)$, and $C_{F_j}(u)$ for each node $u \in U$ executed on each FU type F_j , a timing constraint L and a confidence probability P , find the FU for each node in assignment A that gives the *minimum expected total cost C with confidence probability P under timing constraint L* .

4 The Algorithms

To solve the HAP problem, we propose a high efficient algorithm, *LSHAP*, to minimize the total cost while satisfying timing constraints with guaranteed probabilities. The LSHAP algorithm is shown in Algorithm 4.1. In LSHAP algorithm. Algorithm *HAP_Heu* [6] is used as a sub-algorithm of our *LSHAP* algorithm.

Algorithm 4.1 LSHAP Algorithm

Require: M different FU levels, a PDFG, the timing constraint L , and the rotation times Q .

Ensure: An optimal voltage assignment to minimize cost while satisfying L

- 1: Build up the Execution Time History Table (ETHT).
 - 2: Use data mining to predict the PDF of each varied execution time.
 - 3: Build up the probability distribution function table and CDF table for each varied execution time.
 - 4: Find the association rules between execution times and different inputs.
 - 5: rotation $\leftarrow 0$;
 - 6: while (rotation $< Q$)
 - 7: Get the static schedule (a DAG) from input PDFG.
 - 8: Get scheduling graph from the DAG.
 - 9: Using algorithm *HAP_Heu* [6] to get the near optimal assignment of FU types for the schedule graph.
 - 10: Using Rotation scheduling [3] to retime the original PDFG and rotate down the first row.
 - 11: rotation++;
 - 12: Output results: retime r , assignment A , and the minimum total cost E_{min} .
 - 13: Use online prefetching to reduce cost while satisfying timing constraints with guaranteed probability.
-

LSHAP algorithm includes three major steps. First, we build up the execution time history table (ETHT) and use data mining to predict the execution time pattern from ETHT. Second, we use an high efficient method to obtain

the assignment with minimized total cost. This includes finding the FU assignment with cost minimization in each iteration and using rotation scheduling to obtain the best one for Q iterations. Finally, we use prefetching to prepare data in advance at run time. The experimental data show that our algorithms can get better results on energy saving than the previous work.

In finding the FU assignment with cost minimization in each iteration, we use dynamic programming method traveling the graph in a bottom up fashion. For the ease of explanation, we will index the nodes based on bottom up sequence. Given the timing constraint L , a PDFG G , and an assignment A , we first give several definitions as follows:

1. G^i : The sub-graph rooted at node u_i , containing all the nodes reached by node u_i . In our algorithm, each step will add one node which becomes the root of its sub-graph.
2. $C_A(G^i)$ and $T_A(G^i)$: The total cost and total execution time of G^i under the assignment A . In our algorithm, each step will achieve the minimum total cost of G^i with computed confidence probabilities under various timing constraints.
3. In our algorithm, table $D_{i,j}$ will be built. Here, i represents a node number, and j represents time. Each entry of table $D_{i,j}$ will store a link list of (Probability, Cost) pairs sorted by probability in an ascending order. Here we define the (*Probability, Cost*) pair $(P_{i,j}, C_{i,j})$ as follows: $C_{i,j}$ is the minimum cost of $C_A(G^i)$ computed by all assignments A satisfying $T_A(G^i) \leq j$ with probability $\geq P_{i,j}$.

Usually, there are redundant pairs in a link list. We use Lemma 4.1 to cancel redundant pairs.

Lemma 4.1. *Given $(P_{i,j}^1, C_{i,j}^1)$ and $(P_{i,j}^2, C_{i,j}^2)$ in the same list:*

1. *If $P_{i,j}^1 = P_{i,j}^2$, then the pair with minimum $C_{i,j}$ is selected to be kept.*
2. *If $P_{i,j}^1 < P_{i,j}^2$ and $C_{i,j}^1 \geq C_{i,j}^2$, then $C_{i,j}^2$ is selected to be kept.*

In every step of our algorithm, one more node will be included for consideration. The information of this node is stored in local table $B_{i,j}$, which is similar to table $D_{i,j}$, but with cumulative probabilities only on node u_i . A local table store only data of probabilities and consumptions, of a node itself. Table $E_{i,j}$ is the local table storing only the data of node u_i . The building procedures of $B_{i,j}$ are as follows. First, sort the execution time variations in an ascending order for each R . Then, compute the CDF (cumulative distributive function) under each R . Finally, let $L_{i,j}$ be the link list in each entry of $B_{i,j}$, insert $L_{i,j}$ into $L_{i,j+1}$ while redundant pairs canceled out based on Lemma 4.1. We use the algorithm *HAP_Heu* proposed by Qiu et al. [6] to solve FU assignment with cost minimization in each iteration, and algorithm *HAP_Heu* is used as a sub-algorithm of our *LSHAP* algorithm.

5 Related Work

Data Mining: Data mining is an analytic process designed to explore data in search of consistent patterns and/or systematic relationships between variables, and then to validate the findings by applying the detected patterns to new subsets of data. The ultimate goal of data mining is prediction. The process of data mining consists of three stages: (1) the initial exploration, (2) model building or pattern identification with validation/verification, and (3) deployment, i.e., the application of the model to new data in order to generate predictions.

Data mining is more oriented towards applications than the basic nature of the underlying phenomena [7]. In other words, Data mining is relatively less concerned with identifying the specific relations between the involved variables. For example, uncovering the nature of the underlying functions or the specific types of interactive, multivariate dependencies between variables are not the main goal of Data Mining. Instead, the focus is on producing a solution that can generate useful predictions.

Predictive data mining is the most common type of data mining and one that has the most direct applications. The goal of predictive data mining is to identify a statistical or neural network model or set of models that can be used to predict some response of interest [9, 10]. Text mining is another kind of popular data mining. While data mining is typically concerned with the detection of patterns in numeric data, very often important information is stored in the form of text. Unlike numeric data, text is often amorphous, and difficult to deal with. Text mining generally consists of two parts. One part is the analysis of multiple text documents by extracting key phrases, concepts, etc. The other part is the preparation of the text processed in that manner for further analyses with numeric data mining techniques, e.g., to determine co-occurrences of concepts, key phrases, names, addresses, product names, etc.

Prefetching: There are two general ways of instruction prefetching [11, 12]. The first one is called *Sequential instruction prefetching*. The simplest form of instruction prefetching is next line prefetching. In this scheme, when a cache line is fetched, a prefetch for the next sequential line is also initiated. A number of variations of this basic scheme exist, with different heuristics governing the conditions under which a prefetch is issued. Common schemes include: always issuing a prefetch for the next line (next-line always), issuing a prefetch only if the current line resulted in a miss (next-line on miss) and issuing a prefetch if the current line is a miss or is a previously prefetched line (next-line tagged). Next-N-line prefetch schemes extend this basic concept by prefetching the next N sequential lines following the one currently being fetched by the processor. The benefits of prefetching the next N-lines include, increasing the timeliness of the prefetches and the ability to cover short nonsequential transfers.

The other one is called *Nonsequential instruction prefetching* [13–16]. Nonsequential prefetch prediction is closely related to branch prediction. history-based schemes, such as the target prefetcher, is one of the main styles of prefetcher specifically targeted at nonsequential prefetching, History table is a table is used to re-

tain information about the sequence of cache lines previously fetched by the processor. As execution continues, the table is searched using the address of each demand fetched line. If the address hits in the table, the table returns the addresses of the next cache lines that were fetched the previous times the active cache line was fetched. Prefetches can then be selectively issued for these lines. In this scheme, the behavior of the application is predicted to be repetitious and prior behavior is used to guide the prefetching for the future requirements of the application. The prediction table may contain data for all transitions, or just the subset that relate to transitions between non-sequential cache lines.

6 Experiments

This section presents the experimental results of our algorithms. We build up a data mining engine and predict the probability distribution function of execution time at compile time. Then compute the best assignment and prefetch the data needed in advance. We compare our algorithm with list scheduling and the ILP techniques in [17] that can give near-optimal solution for the DAG optimization. Experiments are conducted on a set of benchmarks including 4-stage lattice filter, 8-stage lattice filter, voltera filter, differential equation solver, RLS-languerre lattice filter, and elliptic filter. M different FUs, F_1, \dots, F_M , are used in the system, in which a processor under F_1 is the quickest with the highest cost and a processor under F_M is the slowest with the lowest cost. The execution times for each node are in Gaussian distribution. Each application has timing constraints. This assumption is good enough to serve our purpose to compare the relative improvement among different algorithms. For each benchmark, we conduct the experiments based on 5 processor cores. The experiments are performed on a Dell PC with a P4 2.1 G processor and 512 MB memory running Red Hat Linux 9.0.

The experimental results with 5 processors are shown in Table 3. In the table, columns “List”, “ILP”, “LSHAP” represent the results obtained by list scheduling, the ILP in [17], our *LSHAP* algorithm, respectively. Columns “Cost” represents the cost and columns “%I” represents the improvement of our algorithm over the ILP in [17] with different probability. The total average improvement of *LSHAP* is shown in the last row.

From the experimental results, we can see that our algorithms achieve significant cost saving compared with the ILP in [17]. On average, *LSHAP* shows a 30.8% reduction in hard real-time, and reductions of 42.9%, 49.3% with probability 0.9 and 0.8, respectively, for soft real-time DSP systems. The reason of such big improvement is because we use loop scheduling to shrink the schedule length and assign the best possible FU to minimize cost while satisfying time constraint with guaranteed probabilities.

7 Conclusion

This paper combined loop scheduling and data mining and prefetching to solve the heterogeneous FU assignment problem. We proposed an algorithm, *LSHAP*, to

TC	List	ILP	LSHAP					
			0.8		0.9		1.0	
			Cost	%I	Cost	%I	Cost	%I
differential equation solver, 5 processors								
40	1018	865	432	50.1	506	41.5	602	30.4
50	887	755	382	49.4	438	42.0	533	29.4
60	826	702	352	49.8	389	44.6	471	32.9
70	643	643	331	48.5	362	43.7	455	29.2
80	706	600	303	49.5	348	42.0	409	31.8
4-stage lattice filter, 5 processors								
100	1986	1687	862	48.9	973	42.4	1178	30.2
110	1548	1316	661	49.8	754	42.7	915	30.5
120	1304	1108	579	47.8	604	45.5	756	31.8
130	1171	996	515	48.3	583	41.4	685	31.1
140	982	835	423	49.3	478	42.8	572	31.5
volterra filter, 5 processors								
80	1991	1692	869	48.6	973	42.5	1182	30.2
90	1368	1163	578	50.3	652	43.9	807	30.6
100	1233	1048	541	48.4	587	44.0	711	32.2
110	1164	989	501	49.4	571	42.3	696	29.6
120	1036	882	432	51.0	512	42.0	615	30.3
Average Imp. over ILP			49.3	–	42.9	–	30.8	

Table 3. The cost comparison for the schedules generated by list scheduling, the ILP in [17], and the LSHAP algorithm.

give the efficient solutions. We first use data mining to obtain the PDF of execution times, which are modeled as random variable in this paper. Then, by taking advantage of the uncertainties in execution time of tasks, we give out FU assignments and scheduling to minimize the expected total cost while satisfying timing constraint with guaranteed probabilities. We repeatedly regroup a loop based on rotation scheduling and decrease the energy by voltage selection as much as possible within a timing constraint. Finally, we prefetch the data needed in advance at run time. Our approach can handle loops efficiently. Experimental results show the significant cost-saving of our approach.

References

[1] T. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. Wu, and J. Liu, “Probabilistic performance guarantee for real-time tasks with varying computation times,” in *Proceedings of Real-Time Technology and Applications Symposium*, 1995, pp. 164–173.

[2] Z. Shao, Q. Zhuge, C. Xue, and E. H.-M. Sha, “Efficient assignment and scheduling for heterogeneous dsp systems,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 16, pp. 516–525, Jun. 2005.

[3] L.-F. Chao, A. LaPaugh, and E. H.-M. Sha, “Rotation scheduling: A loop pipelining algorithm,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 229–239, Mar. 1997.

[4] Y. Chen, Z. Shao, Q. Zhuge, C. Xue, B. Xiao, and E. H.-M. Sha, “Minimizing energy via loop

scheduling and dvs for multi-core embedded systems,” in *Proceedings the ICPADS’05*, Fukuoka, Japan, 20–22 Jul. 2005, pp. 2–6.

- [5] F. E. Sandnes and O. Sinnen, “A new scheduling algorithm for cyclic graphs,” *Intl. J. of High Performance Computing and Networking*, vol. 3, no. 1, pp. 62–71, 2005.
- [6] M. Qiu, Z. Shao, Q. Zhuge, C. Xue, M. Liu, and E. H.-M. Sha, “Efficient assignment with guaranteed probability for heterogeneous parallel dsp,” in *Int’l Conference on Parallel and Distributed Systems (ICPADS)*, Minneapolis, MN, Jul. 2006, pp. 623–630.
- [7] J. Han and M. Kamber, *Data mining: Concepts and Techniques*, New York: Morgan-Kaufman, 2000.
- [8] C. E. Leiserson and J. B. Saxe, “Retiming synchronous circuitry,” *Algorithmica*, vol. 6, pp. 5–35, 1991.
- [9] S. M. Weiss and N. Indurkha, *Predictive data mining: A practical guide*, New York: Morgan-Kaufman, 1997.
- [10] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, New York: Springer, 2001.
- [11] V. Srinivasan, E. S. Davidson, G. S. Tyson, M. J. Charney, and T. R. Puzak, “Branch history guided instruction prefetching,” in *Proc. of the 7th Int’l Conference on High Performance Computer Architecture (HPCA)*, Monterrey, Mexico, Jan. 2001, pp. 291–300.
- [12] J. Tse and A. J. Smith, “Cpu cache prefetching: Timing evaluation of hardware implementations,” *IEEE Transactions on Computers*, vol. 47, no. 5, pp. 509–526, 1998.
- [13] D. Joseph and D. Grunwald, “Prefetching using markov predictors,” *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 121–133, 1999.
- [14] O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt, “Runahead execution: An alternative to very large instruction windows for out-of-order processors,” in *IEEE HPCA-9*, Feb. 2003.
- [15] L. Spracklen, Y. Chou, and S. G. Abraham, “Effective instruction prefetching in chip multiprocessors for modern commercial applications,” in *IEEE HPCA-11*, Feb. 2005.
- [16] C. Yang, A. Lebeck, H. Tseng, and C. Lee, “Tolerating memory latency through push prefetching for pointer-intensive applications,” *ACM Transactions on Architecture and Code Optimization*, pp. 445–475, Dec. 2004.
- [17] Y. Zhang, X. Hu, and D. Z. Chen, “Task scheduling and voltage selection for energy minimization,” in *DAC*, 2002, pp. 183–188.