Low Power Mapping and Pipelined Scheduling Using Tabu Search

Liu Jin-Lin, Wu Kun-Yi, Kuang Shiann-Rong

Department of Computer Science Engineering, National Sun Yat-Sen University
Kaohsiung, Taiwan
jinlin.liu@m2k.com.tw

***Abstract*—**

An tabu search based approach for distributed embedded systems synthesis is presented in this paper. This approach maps each computation task of a particular application onto a system component with multiple supply voltages, schedules the execution of these computation tasks and communication time of global shared bus into pipeline stages. The objective is to minimize the total power consumption subject to the throughput constraint on the pipelined system architecture. The results of experiments show a very good performance to obtain the near-optimal solution compared with global optimal solution obtained by ILP(Integer Linear Programming) approach [1] and analysis the number of orderings in ordering pool to trade-off speed and quality of solution.

I. Introduction

Multimedia and wireless devices typically have intensive computations and an endless stream of input data with throughput constraints. Most of these devices are designed for portable applications, demanding for low-power and high-performance is greatly growing. But there are many trade-off between low-power and performance. System synthesis with power optimization try to solve the design problem which meet the timing constraints. The problem involves various steps such as mapping, scheduling, and dynamic voltage scaling. All step are highly complex and dependent to each other. For example, scheduling cannot be performed without the execution time of computation tasks, which are available only after PE instances and supply voltages being selected. And the voltage selection problem must be considered simultaneously to trade-off performance and power consumption. We try to develop an integrated approach to cope with these design steps simultaneously.

The work in [2] presents a power-delay model to show that dynamic voltage scaling(DVS) can provide drastic power reduction. For given multiple discrete supply voltages and sequential tasks with arbitrary time constraints, a voltage assignment technique which produces a feasible task schedule with optimal processor energy consumption. [3] provide the optimal voltage allocation model with add-on non-uniform load capacitances model.

The system synthesis problem composite of scheduling and mapping problem together. Except for the optimal approaches [4, 5], various heuristic approaches including iterative improvement, constructive method, such as genetic algorithm, simulated annealing and tabu search [7] have been used to synthesize system architectures. In these three algorithm, tabu search provide a solution quality in a short time[6]. However, all these approaches mentioned so far assume that the computation tasks in the system execute sequentially so that they cannot construct efficient pipelined architecture for stream applications. In contrast, [9]–[11] have proposed approaches to build the pipelined system architecture.

Schmitz [12] have demonstrated the efficiency of system architecture synthesis with DVS technique in reducing the power consumption. But it did not perform pipelined scheduling in the system. Our approach is based on tabu search that minimizes the total power consumption of pipelined system architecture including DVS components under throughput constraints. All of mapping, pipelined scheduling, and voltage selection are considered.

The remainder of this paper is organized as follows. Section 2 describes the integrated system architecture synthesis problem of distributed embedded systems for multimedia applications. Section 3 explains our algorithm to solve the integrated problem. Section 4 applies the proposed approach to some examples and discusses the experimental results. Finally, we give a conclusion.

II. Problem Description

A. Task Graph

Typically multimedia applications are dominated by data-flow constructs and can be described as a task graph at a coarse level of granularity. It may be simplified as an pure, non-hierarchical data-flow computation nodes and communication nodes,and each node is executed the same number of times. This is a very restricted semantic but useful in streaming application, the most execution time is spent on the inner loop, called computation kernel. And we focus on the repeat behavior of computation kernel in streaming application.

We extract the computation kernel as the task graph $G(V, E, I, O)$, a directed acyclic graph, where V denotes a set of functional nodes and E denotes a set of communication edges. Moreover, I and O are dummy nodes called the input and output nodes which are used to model the I/O environment and specify the throughput constraint. $F_i \in V$ is a functional node and $C_n(F_{i1}, F_{i2}) \in E$ is a communication edge from source functional node F_{i1} to destination functional node F_{i2}. For the dependency relation in G, the functional node should

be activated after all its input edges are finished. And the output edges cannot be activated until the source functional node is finished. Considering the task graph depicted in Figure 1, C_2 and C_3 can be activated after I is finished, and F_3 cannot be activated until both C_4 and C_3 are finished.

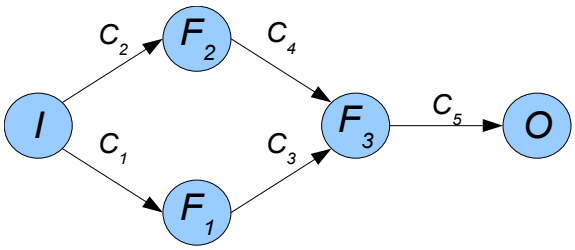

Figure 1: An example of task graph

B. Target Architecture

A distributed embedded system today usually consists of multiple heterogeneous resources (uPs, DSPs, ASICs, etc.), memory components, communication networks and communication interfaces. For the communication network, an on-chip shared bus is adopted in our target architecture. In addition, each component contains a buffer in order to temporarily store the data from/to communication link. For the buffering model, the out-going data can be transferred in the later time after finishing computation and the incoming data can be held before computation. Therefore, the scheduling is more flexible and the communication will affect less on the scheduling of PEs.

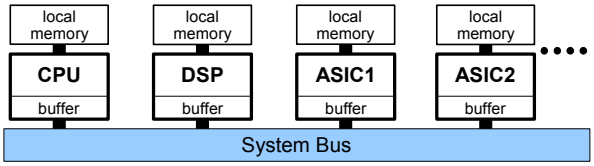

Figure 2: An example of target architecture

An example of target architecture, which consists of four PEs, one system bus is shown in Figure 2. Moreover, distributed PEs provide more power-saving opportunities. More independent PEs share these functional nodes, more slack time exists. As a result, the execution time of functional nodes may be extended in lower voltage level.

C. Pipeline Scheduling

For the repeated nature of stream applications, pipelined scheduling benefits the performance if there are enough PE resources. Pipelined scheduling also works in the system design low power system in throughput constraint. Dependency constraints could be resolved in pipelined stages. There are more opportunities to place longer execution time in lower voltage and to find a mapping that has less communication time.

The throughput constraint is the period of two consecutive input samples. Figure 3 depicts a two-stage pipelined scheduling in 120 throughput constraint. Communication and computation nodes are repeatedly executed on assigned PE instance.

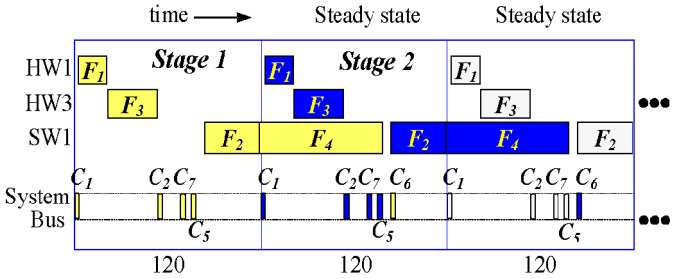

Figure 3: An example of pipelined scheduling

D. Dynamic Voltage Scaling

The DVS supported PE dynamically scales the supply voltage and operational frequency in accordance with the performance requirements of the applications to reduce power consumption. For preceding restrictions among tasks and the throughput constraint, there are many idle time slot distributed in each components. However, to execute a task in the lower voltage in DVS-enabled PEs consumes less energy and spends longer execution time than higher voltage. The energy-delay model in DVS components is shown as flowing Equation 1-5 :

$$f_{Vdd} = k \cdot \frac{(V_{dd} - V_t)^2}{V_{dd}} \quad (1)$$

$$P_{Vdd} = C_L \cdot N_{0\to1} \cdot f_{Vdd} \cdot V_{dd}^{\,2} \quad (2)$$

$$T_{Vdd+\Delta v} = \frac{V_{dd} + \Delta v}{(V_{dd} + \Delta v - V_{th})^2} \cdot \frac{(V_{dd} - V_{th})^2}{V_{dd}} \cdot T_{Vdd} \quad (3)$$

$$P_{Vdd+\Delta v} = \frac{(V_{dd} + \Delta v) \cdot (V_{dd} + \Delta v - V_{th})^2}{V_{dd} \cdot (V_{dd} - V_{th})^2} \cdot P_{Vdd} \quad (4)$$

$$E_{Vdd+\Delta v} = P_{Vdd+\Delta v} \cdot T_{Vdd+\Delta v} \quad (5)$$

Equation 3 is derive by Equation 1 and 2. where P_{Vdd} and T_{Vdd} denotes operational power and execution time of a task mapped onto a specific PE in V_{dd} voltage. V_{th} is threshold voltage a PE corresponding the manufacture characteristics. We can evaluate the $E_{Vdd+\Delta v}$ and $T_{Vdd+\Delta v}$ by Equation 3, 4 and 5 in the difference voltage Δv.

Based on the above explanation, the problem of integrated system architecture synthesis for embedded streaming applications can be described as follows. Given a task graph G(V, E, I, O), a resource library with the corresponding implementation information including PE's execution time, PE's power consumption, communication time, communication energy on each communication node, a throughput constraint and stage number constraint , the problem is to (1)map each $F_i \in V$ to a PE instance run at a feasible voltage to execute its function; (2)schedule function computation and data communication in pipeline; (3)assign each $C_n(F_{i1}, F_{i2}) \in E$ to an internal communication, or the shared system bus; such that (4)the throughput constraint, stage number constraints are satisfied; (5)total power consumption are minimized.

III. Algorithm

Our approach simultaneously determines PE instance mapping, voltage mode and pipelined scheduling for all nodes of application solved by the tabu search.

Tabu search[15] is an adaptive procedure for solving complex optimization problems. The solution is encoded as **bits**. Neighbors are generated from all solutions that are exactly one bit different from initial solution in the first move or current solution (the local optimal solution in the last move). But not all solutions are admissible, such as tabu list and intensification avoid some neighbors being a current solution in the move. In the progress of searching, the local local optimal solution in the admissible neighbors in each move will be found out by burst force and the global optimal solution will be improved by best current solution.

Tabu search can solve variable complex problems. In our proposed algorithm, we modified tabu search and a pipelined scheduling to solve the system architecture and scheduling problem that we mentioned in the section 2. We explain our system in detail as follows.

A. Tabu Search for Low Power in Distributed Embed System

There are two major procedures in proposed synthesis system. One is the modified tabu search designed for solving the mapping problem (PE instance mapping and voltage mapping). The other is the pipelined scheduling to schedule all selected neighbors from tabu search. For the pipelined scheduling is most time-consuming procedure, a fast neighbor filter is designed to drop bad neighbor mappings while generating neighbors. Finally, the acceptable neighbor mappings in the move have to get a schedule ordering to be scheduled in pipelining, and the system drops the neighbors that can not be scheduled. The diversification try to find another current solution if the cost of current solution is the same for several moves. And the diversification helps the system for finding a new searching path in order to escape the local optimal solution.

Figure 4 presents an overview of the synthesis system. The initial solution is decided by CNPT [13] after preprocessing the input data. We do not apply voltage optimization on initial solution with CNPT and voltage level of all tasks are assigned at maximal voltage level. However, CNPT is a efficient start point to find a feasible PE instance mapping.

A flow chart of our approach is shown in Figure 4. The fat arrows indicate the constant data generated once at the start-up procedure. There are four user-defined constant data as input in our system: (1)Task graph describes the application by functional nodes, communication nodes, data dependencies, and stage time constraint, (2)MLib provides the energy cost and execution time of each functional node at different voltage modes. The execution time and energy of communication node are also listed in MLib, and (3)constraints of pipelined scheduling and tabu search. All above data is prepared before performing tabu search.

B. Encoding of Neighbors and Cost Function

The set of tabu bits, a mapping solution, is encoded as voltage level v and PE instance res together in a corresponding functional task, from task 1 to task n, shown in Equation 6. The two composite attributes, res and v generate more neighbors than single attribute in each move. However, cost evaluation is more efficiently by Equation 7 and MLib. ΣE_{task} is the sum of the energy cost which is looked up by voltage level and PE type in every task by mapping solution and MLib. ΣE_{comm} is the sum of communication energy in every communication task. In the specific communication task transfer from functional $task_i$ to $task_j$, if $PE_i \neq PE_j$, the energy is obtained from MLib. Otherwise, the energy is 0 in the communication task for transferring between the same resource instance.

$$tabu\ bits = \{(res, v)_{t1}, (res, v)_{t2}, \cdots, (res, v)_{tn}\} \quad (6)$$

$$E_{total} = \sum E_{task} + \sum E_{bus} \quad (7)$$

$$E_{bus}(C_{task_i, task_j}) = \begin{cases} E_{lib}(C_{task_i, task_j}) \in MLib, if\ (i \neq j) \\ 0, if\ i = j \end{cases} \quad (8)$$

Our objective is to find a set of tabu bits (system architecture) and scheduling to minimize E_{total} in the time and dependency constraints.

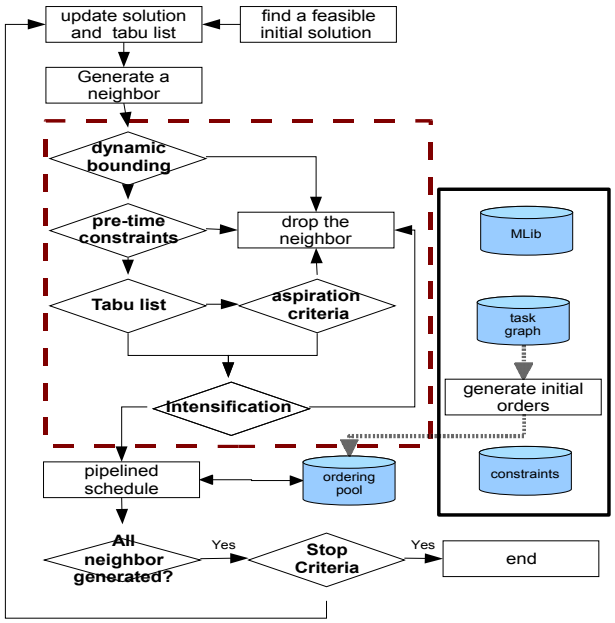

Figure 4: Data flow of our approach

C. Neighbor Selection

The selection strategy of neighbors in each move is important for searching efficiency. To efficiently reduce the number selected neighbors, we use a fast timing analysis that drops many impossible and unwanted solutions. A mapping solution is generated as a neighbor from current solution must be tested in the neighbor selection presented in the dash border of Figure 4. Only the selected neighbors can be sent forward to

pipelined scheduling procedure. The neighbor solution pass the scheduling will replace the best solution in the move.

Dynamic bounding drops the mapping worse than the best solution passed the scheduling in this move. Since we select only one neighbor that consume minimal energy (the local optimal of the move) among the neighbors of the move. And we can evaluate the cost of the mapping before scheduling. We can drop the mapping solution never be the local optimal solution.

Pre-timing constraints efficiently drops the impossible mapping solution before scheduling. We use a fast timing analysis from solution bits. The two attributes of a bit, the PE mapping and voltage level mapping of functional nodes, are known in each mapping solution. We can sum up execution time of all functional nodes for each different PE instance. If the sum of execution time in the same PE is larger than timing constraint (stage time constraint), the mapping solution is trivial to be dropped in this move and not be sent forward to scheduling procedure.

The tabu list helps us to avoid falling into a local optimization. In the process of tabu search, the current solution has exactly one bit different between two continuous moves except diversification and aspiration criteria is meet. A tabu bit in the tabu list is the changed bit between previous and current solution. The tabu list avoid the current solution being the same as the attributes in the bits of tabu bits in next *tabu length* move. The aspiration criteria gives a chance for neighbors from tabu list with more strict rule. The cost of mapping has to be smaller than the best solution in all moves until now.

Intensification will drop the mapping same as any local optimal solution in previous moves. This mechanism prevents the repeat solution and searching path.

D. Schedule Orderings Generating

The candidates of schedule ordering are generated by the dependency relations of task graph, dramatically impacts the scheduability of solutions. In a small task graph, we can cover all possible orderings. But the number of possible orderings in large application increase in the O(n!) complexity.

For the large case, the orderings will be randomly generated in limited N_{order} number of orderings pool for scheduling all neighbors in tabu searching. The pool of ordering will be dynamically replaced with the new and better orderings. The N_{order} effects our searching time in linear time complexity. However, a large N_{order} do not effect much in quality of solution. A reasonable N_{order} number is chosen for high speed and acceptable quality of solution will be discuss in our experiments.

E. Pipeline Scheduling

The pipelined scheduler will check if the neighbor can be scheduled in the ordering from the ordering pool. The pipelined scheduler is a time-constrained LIST based algorithm to decide the start time and end time of each nodes. Since the mapping information is already contained in the bits and orderings are assigned by the ordering pool, the scheduler can schedule the nodes in a fast speed. However, the pipelined issue should be considered while scheduling. The scheduler will check that the scheduled interval of an functional nodes is not overlapped with other nodes mapped onto the same instance; or the interval of an communication node is not overlapped with other communication nodes on the global bus. If there are any two nodes overlapped on the same instance or bus, the scheduler will try to delay the latter overlapped node in order to find a slack to schedule the node.

Like Equation 8, from the dependency relations and mapping information provided by solution bits, the scheduler can decide the communication time T_{bus} in Equation 9. And energy of communication E_{bus} has the similar formulation for cost evaluation in tabu search.

$$T_{bus}(C_{task_i,task_j}) = \begin{cases} T_{lib}(C_{task_i,task_j}) \in MLib, if\ (i \neq j) \\ 0, if\ i = j \end{cases} \quad (9)$$

```
procedure schedule(mapping, ordering)
begin
        initial schd;
         foreach fn in mapping  in order of ordering {
            foreach comm in  fn.in_edge {
               schd[bus]←FirstFit(comm, schd[bus]);
               if(isFail(schd[bus]) ) return FAIL;
            }
            instance← mapping[fn].PEinst;
            schd[instance]←FirstFit(fn, schd[instance]);
            if(isFail(schd[instance] ) ) return FAIL;
         }
        return schd, mapping, ordering;
      }
end

procedure Pipeline_Schedule
begin
   initial current_solution← MAX;
   foreach mapping  in selected_neighbors {
       if(current_solution.energy > mapping.energy) {
     foreach ordering in ordering_pool {
          schd←schedule(mapping, ordering);
          if(isSuccess(schd) ) {
            update_pool(ordering_pool, ordering);
            current_solution ← mapping;
            return schd, mapping, ordering;
          }
       }
       repeat 1.. Norder {
         new_ordering ← gen_new_ordering();
          schd←schedule(mapping, new_ordering);
          if(isSuccess(schd) ) {
            update_pool(ordering_pool, new_ordering);
            current_solution ← mapping;
            return schd, mapping, new_ordering;
          }
       }
      return FAIL_MAPPING;
     } }
end
```

Figure 5: The sudo code of pipeline scheduling

We try each mapping that is performed neighbor selection to schedule in all N_{order} orderings. If there is not any ordering can make this neighbor be scheduled, we try the other new N_{order} randomly generated orderings until there is an successful ordering. In the best case, in totally $2N_{order}$ orderings being

tested. The solution will be dropped after failed testing in all orderings. Otherwise, the new successful ordering replaces one of orderings in original N_{order} ordering pool and the tabu search updates current solution with new solution. Figure 5 briefly describes our pipeline scheduling method. We schedules each functional node *fn* in orderings from ordering pool. There is second chance to schedule in other new random ordering.

FirstFit procedure decides the start time in *schd* structure by first-fit policy. It searches for the earliest time slot to fit the inputed execution/communication time with dependency and pipelined constraints in specified PE instance. In the schedule procedure, the *ordering* is the ordering to schedule the functional node, *fn*. We schedule all the preceding communication nodes of functional nodes, *fn.in_edge*. Finally, we schedule *fn* into the *sched* of mapped *PEinst*. If there is any successful ordering in the mapping, we update current solution and ordering pool to the successful mapping and successful ordering, respectively. The updated *current_solution* can skip scheduling some neighbors that do not has the less cost than the cost of *selected_neighbors* in this move, which mentioned in neighbor selection.

IV. Experimental Results

The experiment list in Table 1 shows the performance of our approach. All task graphs and resource libraries are generated by TGFF[14]. There are 2 different PEs supporting DVS in our experiment. These experiments run on the AMD opteron 2.0G MHz machine. We compare the quality with the optimal solution solved by ILP [1].

The node number tells how many functional nodes in the task graph. N_{order} is the number of ordering in the pool. Optimal power is obtained by ILP solver in the same system model and objective as our tabu approach.

Note that N_{order} column and optimal power column with the star notation in some cells. The N_{order} with single star use all possible ordering. It is occurred in the small cases which has fewer number of schedule ordering. For the huge time in ILP model, the optimal power solution with double star did not finish the ILP solving procedure. It may not be the real optimal solution. In the very large cases, a feasible solution may not be found in 4 hours. We note the optimal solution as N/A for not available solution. But the solutions of ILP are all run at least 4 hours. They are still good benchmarks to our system.

We solve these problems in 800 moves in all experiments. The run-time of our approach in tabu time column is in the unit of seconds. It can be observed that the run-time is very fast. In the experiment of 30 nodes case, we can solve it in 2 minutes. Moreover, the power difference shows the solution quality will be better then ILP solutions solve in 4 hours. And the difference will be noted with a negative number.

The second experiment shows the relation between N_{order} and normalized average cost by the step of moves. Figure 6 shows the cost is decreased by the move number. The average cost is converge near 600 moves. However, more N_{order} is not always a better results. In fact, 100 is the best number among 50, 100 and 500 in our experiments. It tell us that don't use too large N_{order}. A large N_{order} spends more run-time (linear complexity) without better solution.

Table 1
Experimental Result List

node number	N_{order}	optimal power	tabu power	tabu time	power difference
4	*2	156	156	0.91	0
5	*8	193	193	1.34	0
6	*8	193	210	2.20	17
7	*11	**334	334	1.88	0
8	*28	**343	365	4.81	12
9	45	**503	503	7.76	0
10	50	**480	482	8.23	2
11	55	**553	562	11.30	9
12	60	**601	615	12.96	14
13	65	**752	702	17.41	-50
14	70	**957	839	20.42	-118
15	75	**930	823	18.37	-107
26	130	N/A	1538	96.26	N/A
27	135	N/A	1756	80.37	N/A
28	140	N/A	1746	95.11	N/A
29	145	N/A	2122	97.21	N/A
30	150	N/A	2056	113.13	N/A

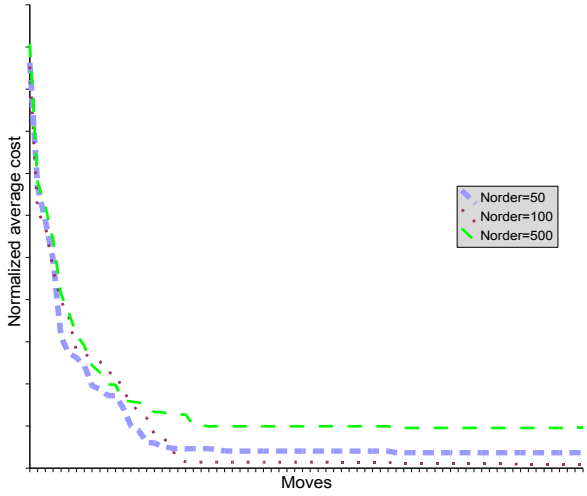

Figure 6: average power consumption in diffrent N_{order} in 1500 moves

V. Conclusion

An efficient approach to explore the system design parameters including PE mapping, voltage mapping and pipeline scheduling by tabu search method is proposed in this paper. In the experiment results, we compare the solution with ILP-based approach. It shows an excellent performance end high quality solution in our approach. And provide a dynamic ordering pool to maintain the good schedule ordering. Finally, we analysis the trade-off between run-time and solution quality in moves count and N_{order} .

Acknowledgment

This work was supported in part by the National Science Council, R.O.C., under Grant NSC 95-2221-E-110-017

References

[1] J. L. Liu, S. R. Kuang and S. F. Hsiao, "Integrated Architecture Synthesis of Distributed Embedded Systems for Multimedia Applications" in *Proc. International Computer Symposium* pp. 224-229, 2006.

[2] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. Int. Symp. on Low Power Electronics and Design*, pp. 197–202, 1998.

[3] W.-C Kwon and T.-W Kim: 'Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors ', *ACM Transactions on Embedded Computing Systems*, Vol. 4,No 1, February, 2005, Pages 211-230

[4] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," in *Proc. Design Automation for Embedded Systems*, vol. 2, pp. 165–193, March 1997.

[5] H. Liu and D. F. Wong, "Integrated partitioning and scheduling for hardware/software co-design," in *Proc. International Conference on Computer Design: VLSI in Computers and Processors*, pp. 609–614, Oct. 1998.

[6] S. Banerjee and N. Dutt, "Efficient search space exploration for HW-SW partitioning," in *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis*, pp. 122–127, 2004.

[7] S. Porto and C. Ribeiro: "A Tabu search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints", *International Journal of High Speed Computing,* , Vol. 7, No. 1, pp. 45-71, 1995

[8] T. Wiangtong, P. Y. K. Cheung and W. Luk "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware–Software Codesign" *Journal of Design Automation for Embedded Systems*, Volume 6, Number 4, pp. 425-449, 2002

[9] S. Bakshi and D. D. Gajski, "Partitioning and pipelining for performance-constrained hardware/software systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Vol. 7, No. 4, pp. 419–432, 1999.

[10] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Vol. 7, No. 1, pp. 92–104, 1999.

[11] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Vol. 10, No. 3, pp. 193–208, 2002.

[12] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in *Proc. Design, automation and test in Europe Conf.*, pp. 514–521, March 2002.

[13] T. Hagras, J. Janeček 'A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems' *Proc. Int. Symp. on Parallel and Distributed Processing*, Vol. 31, No. 7 pp. 653 - 670 , 2005

[14] TGFF http://ziyang.ece.northwestern.edu/tgff/

[15] F. Glover, M. Laguna "Tabu Search"Kluwer Academic Publishers, 1997