Efficient Algorithm for Hardware/Software Partitioning and Scheduling on MPSoC

Honglei Han
School of Computer Science and Software Engineering, Tianjin Polytechnic University, China
Email: honglei.han@gmail.com

Wenju Liu, Wu Jigang, Guiyuan Jiang
School of Computer Science and Software Engineering, Tianjin Polytechnic University, China
Email: liuwj@tjpu.edu.cn, {asjgwu, jguiyuan} @gmail.com

***Abstract*—Hardware/software (HW/SW) partitioning and task scheduling are the crucial steps of HW/SW co-design. It is very difficult to achieve the optimal solution as both scheduling and partitioning are combinatorial optimization problems. In this paper a heuristic solution is proposed for scheduling and partitioning on multi-processor system on chips (MPSOC). In order to minimize the overall execution time, the proposed algorithm assigns different priorities to different tasks according to their out-degree and the software execution time. Task with higher out-degree possesses higher priority. For the tasks with the same out-degree, the higher the software execution time, the higher the priority. The proposed algorithm initially searches for the critical path in the task graph, and then assigns the task with the highest benefit-to-area ratio to hardware implementation. The critical path and the available hardware area are updated during the iteration. The whole calculation process works until the available hardware area is not enough to implement a software task lying in the critical path. As a result, the hardware area is utilized as many as possible. Simulation results show that, the proposed algorithm can reduce the overall execution time up to by 38% in comparison to the latest work.**

***Index Terms*—Hardware/software partitioning, task scheduling, algorithm, MPSoC**

I. INTRODUCTION

With the rapid development of integrated circuit manufacturing technology, embedded systems are widely used in a variety of complex applications. How to shorten the product-to-market cycle and accelerate development efficiency has become a major concern in the field of embedded system design. HW/SW co-design remedies many shortcomings of traditional design methods and becomes a kind of mainstream technology of embedded systems development. HW/SW partitioning is a key step of HW/SW co-design and it plays a crucial role in improving the system performance [1]. A large number of contributions in research show that efficient techniques for HW/SW partitioning can achieve results superior to software-only solution.

Multi-processor-system-on-chip (MPSoC) reflects the major trends of today's architecture development. It represents a stronger computing power and higher integration, while the scheduling on such systems has become the main factor of the system performance [2].

The task scheduling on multiprocessor and HW/SW partitioning are classic combinatorial optimization problem. It has been shown that they are NP-hard problems [3]. In recent years, many researchers are committed to research how to improve the performance for MPSoC. Due to many indicators of system performance evaluation and many indexes with a relationship of mutual restraint, it is impossible to get a full sense of the optimal solution [4]. Thus only one metric can be minimized while other metrics merely satisfy the constraints. Most researchers concerned on execution time and how to shorten the execution time has turned into research of many scholars [5]. Most of these works tackle the scheduling and partitioning problems together [6, 7, 8, 9], while others focus purely on the partitioning problem.

The present studies in HW/SW partitioning usually formalize a partitioning problem as a graph bi-partitioning problem to find a global optimum which results in a scalable algorithm for general-purpose systems. Traditional approaches for hardware/software partitioning include hardware-oriented and software-oriented approaches. Hardware-oriented approach starts with a complete hardware solution and iteratively moves parts of the system to the software as long as the performance constraints are fulfilled. While software-oriented approach starts with a software program moving pieces to hardware to improve speed until the time constraint is satisfied [10].

Earlier works in constraint-driven HW/SW partitioning mainly take into account the hardware cost and performance constraints, while recent works have focused more on the problem of HW/SW partitioning for low-power systems. The work in [11] models HW/SW partitioning as an optimization problem with the objective of minimizing power consumption under the hardware

Manuscript received March 22, 2012; revised June 13, 2012; accepted June 14, 2012.
Project number: National Science Foundation of China under Grant No. 60970016 and No. 61173032.
Corresponding author: Honglei Han(honglei.han@gmail.com).

doi:10.4304/jcp.8.1.61-68

area constraints and execution time. It has been reported that HW/SW partitioning can reduce energy consumption by up to 99% [12, 13, 14]. A methodology targeting low-power warp processor that leverages voltage and frequency scaling to dynamically and substantially reduce power consumption without any performance degradation is presented in [15, 16].

Heuristic algorithms have often been used to solve the partitioning models, as exact algorithms become intractable when the problem size is large [17]. The work in [18] used ILP and genetic search for HW/SW partitioning. The authors asserted that the capability of ILP is a deterministic approach only for small problems, while genetic search is more favorable for large-scale problems.

Other reported approaches for HW/SW partitioning include dynamic programming algorithm [19], integer linear programming [20], branch and bound [21], greedy algorithm [22], simulated annealing [12, 23, 24], tabu search [10, 23] and genetic search [25, 26]. These methods are only for smaller partition problem. The work in [23] compared HW/SW partitioning algorithms based on genetic search, simulated annealing and tabu search, and concluded that the tabu search approach outperforms its counterparts.

For task scheduling on multiprocessor, a heuristic strategy based technique is used to explore the search space in order to get the feasible near optimal solution [11]. Although not always being able to get the optimal solution by the heuristic strategy, a near-optimal solution can be found within acceptable time.

In Ref. [27], area efficiency is the major consideration in the hardware/software partitioning process. The HW/SW partitioning was modeled as an optimization problem to minimize the area occupied by hardware under two given constraints in power consumption and execution time. The heuristic algorithm was proposed by extending an efficient idea for 0-1 knapsack problem.

In Ref. [28], the power consumption was considered as the key element. This problem was modeled as an optimization problem to minimize power consumption under two constraints: area penalty and execution time. An efficient algorithm was proposed by extending 0-1 knapsack problem similarly.

A few algorithms below all considered the execution time as the major factor. In Ref. [29], the computing model was extended, in which communication penalties between neighboring components were considered. A heuristic algorithm for path-based HW/SW partitioning was proposed and an efficient algorithm based on tabu search was used to refine the heuristic solutions to the nearly optimal ones. In Ref. [10], the HW/SW partitioning problem was reduced to a variation of knapsack problem that was approximately solved by searching 1D solution space to reduce time complexity. In Ref. [11], a heuristic strategy was adopted to move the task with the highest benefit-to-area ratio to hardware for implementation. Then the tabu search was used to optimize the solution obtained by the heuristic algorithm. In Ref. [30], an efficient algorithm was proposed to make a fast partitioning for the solution obtained by scheduling. The certain tasks on the same processor which had the longest execution time were moved to hardware for implementation on the condition that the hardware area was enough. Otherwise, the partitioning failed.

Most of the previously reported work can work perfectly within their own co-design environments. However, due to the lack of benchmarks and the large differences in co-design environments, it is not possible to compare the results obtained.

This paper presents a solution for task scheduling on multi-processor and HW/SW partitioning problems. A priority mechanism has been proposed based on the existing scheduling algorithm which is level by level. According to their out-degree and software execution time to determine their priorities of different tasks in order to ensure the task with higher out-degree or greater software computation time can be given a higher priority to be scheduled. The core task can be implemented predictably by the proposed scheduling algorithm to reduce unnecessary waiting time. In addition, the original HW/SW partitioning algorithm is also improved. It proposes the concept benefit-to-area ratio and moves the task with the highest benefit-to-area ratio to be implemented on hardware instead of software. Update the available hardware area and the critical path during each of the iteration. It works until there is not enough hardware area meeting the requirement of a software task any more. The proposed HW/SW partitioning algorithm is more universal and can fully utilize the hardware resources to improve system efficiency.

This paper is organized as follows. In Section 2, some statements and task graph model are introduced. As for Section 3, we describe the proposed algorithms. In Section 4, we show some experimental results and the analysis. Then we conclude this paper in Section 5.

II. TASK GRAPH AND RELATED STATEMENTS

The task graph is given as a Data Flow Graph (DFG), denoted as G (V, E, SW, HW, C, A), which is a Directed Acyclic Graph (DAG) in general.

V : the set of nodes and each node represents one task;

E : the set of edges which denote the dependence relationship between two nodes;

SW: the software execution time of the task node;

HW : the hardware execution time of the task node;

C : the communication costs between two tasks corresponding to the two vertex of one directed edges;

A : the hardware area required by the nodes implemented on hardware;

The following example shows a task graph, where $V=\{T_1, T_2, T_3, T_4\}$, $E=\{<T_1, T_2>,< T_1, T_3>,< T_2, T_4>,< T_3, T_4>\}$. Take the node T_1 for instance, $SW(T_1)$ is 70, $HW(T_1)$ is 18, $A(T_1)$ is 9 and $C <T_1, T_2>$ is 44.

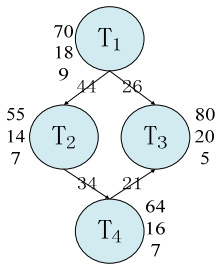


Figure 1. Task graph model

The task scheduling is actually a mapping process of all tasks to p processors. That is to say, divide the set V of task nodes to p subsets V_i ($1 \leqslant i \leqslant p$). The tasks in each subset V_i are assigned to the same processor P_i. If the two nodes with dependency are assigned to the same processor, the communication cost between them is free. The formal description of the scheduling process is as follows.

Scheduling priority is assigned according to the following rules:

a. The smaller the level of the node, the higher the priority;

b. The greater the out-degree, the higher the priority;

c. The greater the software cost, the greater the priority.

If all the parent nodes of the current node are scheduled and executed, the current node is ready.

The task graph after scheduling is denoted as G_s= (V_s, E, T_s, C_s, A), where V_s is the set of task nodes, which consists of p subsets called V_i and subjects to the constraints of (1).

$$\begin{cases} V_s = \bigcup_{i=1}^{p} V_i = V \\ Vi \bigcap Vj, \quad 1 \le i, j \le p \quad and \quad i \ne j \end{cases} \tag{1}$$

T_s is denoted as (2), which represents the actual execution time of each task.

$$\forall v \in V, \quad T_s(v) = SW(v) \tag{2}$$

$C_s(u,v)$,which is described as (3), denotes the actual communication time between two tasks.

$$\forall < u, v > \in E, u \in V_i, v \in V_j, (i, j \in [1, ..., p]),$$
$$C_s(u,v) = \begin{cases} 0, & i = j \\ C(u,v), & i \ne j \end{cases} \tag{3}$$

The goal of the task scheduling is to make the $max(Len(\ P_i(V_i)\ )\ )$ as small as possible, where $P_i(V_i)$ is the sequence of the tasks on the processor P_i .

Benefit-to- area ratio [15] is calculated by (4).

$$\text{Benefit-to-Area Ratio:} \quad \text{BAR} = \frac{SW - HW}{area} \tag{4}$$

Critical path is the scheduling sequence of tasks on the processor with the longest scheduling length, namely $Len(\ P_i(V_i) = max(Len(\ P_i(V_i)\ )\ )$.

The HW/SW partitioning is to make a division of all the subsets of nodes as well. The task with the max BAR in the critical path V_i will be moved to hardware implementation iteratively. By these steps, the computation time of tasks will be reduced greatly, so that the overall execution will be minimized. After HW/SW partitioning, the task graph is represented as G_{sp}=(V_{sp}, E, T_{sp}, C_{sp}, A). V_{sp} is the set of task nodes, which contains p+1 subsets, denoted as (5), where V_h is the set of the tasks of hardware implementation, which is derived from the set V_i.

$$V_{sp} = (\bigcup_{1 \le i \le p} V_i) \bigcup V_h \tag{5}$$

T_{sp} is the time of each task after HW/SW partitioning, which is described as (6).

$$\forall v \in V_{sp}, \quad \begin{cases} T_{sp}(v) = HW(v), & (v \in V_h) \\ T_{sp}(v) = SW(v) & (v \in V_i, 1 \le i \le p) \end{cases} \tag{6}$$

For C_{sp}, it is the same as C_s and represented as (7).

$$\forall < u, v > \in E, u \in V_i, v \in V_j (i, j \in [1, ..., p] \cup \{h\}),$$
$$C_{sp}(u,v) \begin{cases} 0, & i = j \\ C(u,v), & i \ne j \end{cases} \tag{7}$$

III. ALGORITHMS FOR SCHEDULING AND PARTITIONING

Reducing the overall execution time as much as possible is the common goal for task scheduling and HW/SW partitioning. In order to solve these hard problems, this paper proposed a solution, which consisted of two parts: one is task scheduling on multi-processor, which determines the priority of each task according to the task's layer number, out-degree and software execution time. The other is HW/SW partitioning, which moves the task with the highest benefit-to-area ratio in the critical path to be implemented on hardware during each of the iteration, and then update the critical path and the available hardware area, until the area does not meet the hardware requirement of a task in the critical path any longer.

A. Scheduling on Multi-processor

A-star algorithm is used in scheduling process and it contains two lists: *Open* and *Closed*. The cost functions are denoted as (8) (9) (10).

$$g(current) = \begin{cases} g(parent) + C + current\_SW, \\ Communication\_is\_not\_free \\ g(parent) + current\_SW, \\ Communication\_is\_free \end{cases} \tag{8}$$

g (s) represents the cost required from the start node to the node s.

$$h(current) = \max\{h(parent) + C\} \quad (9)$$

h (s)is the estimated cost required from the node s to the target node.

$$f(current) = g(current) + h(current) \quad (10)$$

f (s) denotes the cost function value of the node s.

The scheduling process consists of four steps.

Firstly, initialize the *Open* and *Closed* list. There is only the start node in the *Open* list. The *Closed* list is empty. The *Open* list stores ready nodes in descending order of priority.

Secondly, the first node in the *Open* list is the current node. Calculate the cost function of the current node and assign the task to the processor, which the value of the cost function is the smallest when the task is assumed to be assigned to.

Thirdly, determine whether all its child nodes are ready when the current node has been executed.

At last, if the child node is ready, insert it into the *Open* list in the corresponding position; otherwise do nothing. Put the scheduled node into *Closed* and one time of iteration is over.

Repeat the steps above, until all nodes is scheduled. The formal description of the scheduling algorithm is as follows.

Input: The task graph and the number of processors;
Output: The scheduling sequence and the scheduling length;
Algorithm *Scheduling* ;
begin
  Initialize the lists *Open* and *Closed* ;
  while (*Open* is not empty**) do**
    current: = *Openfirst*;
    for i : = 0 **to** p-1 **do** /**p is the processors number */
        $f_i(current)$: = $g_i(current)$ + $h(current)$;
            /*cost function*/
    end for
    $f(current)$: = $\min\{f_i(current)\}$, i : = 0 **to** p-1;
    if (the children of current node are Ready)
        then update the *Open*;
    end if
    Transfer the scheduled node to *Closed*;
    Openfirst++;
  end while
end

Assuming the task graph contains N nodes, E directed edges, the system has p processors, the maximum out-degree and in-degree of task graphs are all constant k. According to the pseudo-code, initiating the Open and *Closed* list takes O(1) time. The iteration of "while" will be repeated N times. During each time of the iteration, the function f, g and h are calculated once. The complexity of calculating f is O(1). For g, the complexity is O (p * (k * N) + p) = O (pN). The value of function h can be computed by once task graph traversal, and the complexity is O (N). Therefore, the complexity of the scheduling algorithm is O(1)+O(N)*(O(1)+O(pN)+O(N)) = O(pN^2).

B. HW/SW Partitioning based on Critical Path

The HW/SW partitioning starts after scheduling. Assume the overall hardware area is A. First of all, select the critical path, which is the sequence of tasks on a processor with maximum scheduling length. Secondly, move the task with the highest benefit-to-area ratio in the critical path to be implemented on hardware instead of software. At last, update the available hardware area and the length of the critical path. Repeat the steps above, until the available hardware area is not enough for a task lying on the critical path. The formal description of the partitioning algorithm is as follows.

Input: The scheduling sequence on different processors;
Output: The task sequence implemented by the hardware and the last scheduling length.
Algorithm *Partitioning*;
begin
while (the available hardware area is enough for limitation)
do C := the critical path;
  max:=the task with the *greatest benefit-to-area ratio*;
  if (the available hardware area is satisfied)
  then Move the *max* to hardware implementation;
      Update the hardware area and the length of the critical path;
  else Test the next task on the critical path with *smaller benefit_to_area ratio*;
  end if
end while
end

IV. EXPERIMENTAL RESULTS AND ANALYSIS

Experimental environment is set as follows: windows XP system, Intel Core2 CPU 2.20GHz, Memory 2GB, the software is VC++6.0, and the programming language is C. Set that there are three processors and the hardware area A are 40 and 60. Task graphs are generated randomly and still use the benchmark-RGBOS (Random Graphs with Branch-and-bound obtained Optimal Solutions), which is used in the comparison article [30] for testing. RGBOS is a set of randomly generated graphs, which contains three sub-sets with different Communication to Computation Ratio (CCR) 0.1, 1.0, 10.0. Each subset also contains 12 graphs and the number of nodes from 10 to 32 increased by 2 [31]. The software cost and the hardware area are generated randomly. The execution time of hardware (FPGA) equals one third to one fifth of the execution time of software (processor) [32]. Experimental data obtained the average of 20 experiments. The results are as follows.

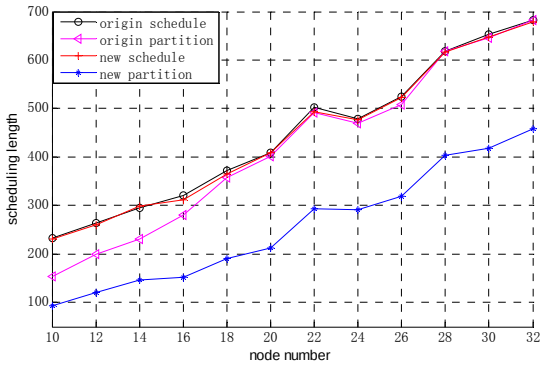


Figure 2. *A*=40,*CCR*=0.1
Hardware area is 40 and the Communication to Computation Ratio (CCR) is 0.1

As shown in figure 2, it is the result of the case of "A=40, CCR=0.1". The smaller the value is, the better the quality of the algorithm is. It can be seen that the results are significantly better than the original one by optimizing the HW/SW partitioning algorithm, about 39.6% improvement. Especially when the number of nodes is 20, the improvement equals 47.1%.

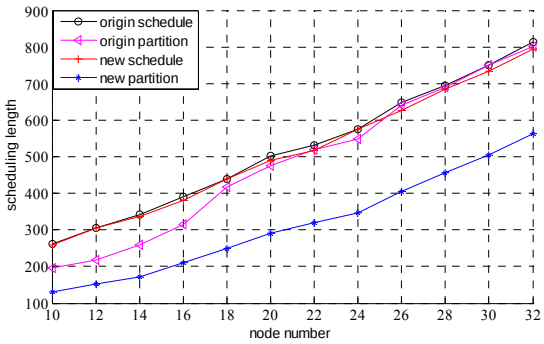


Figure 3. *A*=40,*CCR*=1.0
Hardware area is 40 and the Communication to Computation Ratio (CCR) is 1.0.

As shown in figure 3, the improvement is about 34.9% through the optimization of the HW/SW partitioning algorithm. The improvement is the largest when the number of nodes is 18, and the corresponding rate is 40.4%. With the increase in the number of nodes, the existing partitioning strategy has been almost invalid, while the improved partitioning strategy can fully utilize hardware area to reduce the total execution time as much as possible.

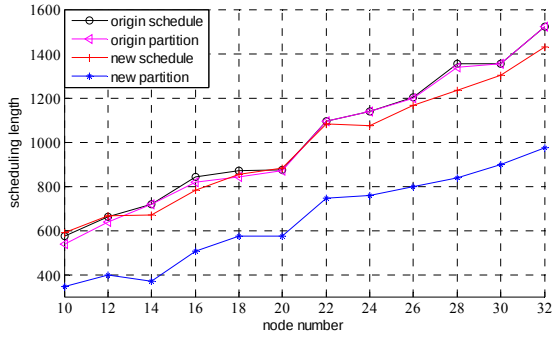


Figure 4. *A*=40,*CCR*=10.0
Hardware area is 40 and the Communication to Computation Ratio (CCR) is 10.0.

As shown in figure 4, the improvement is more obvious through the optimization of the scheduling algorithm, compared with the two cases above. Especially when the number of nodes is 28, the improvement is about 9%. The total improvement can be up to 48.6%, when the number of nodes is 14. It has been shown that the existing HW/SW partitioning algorithm is totally ineffectual in the whole process.

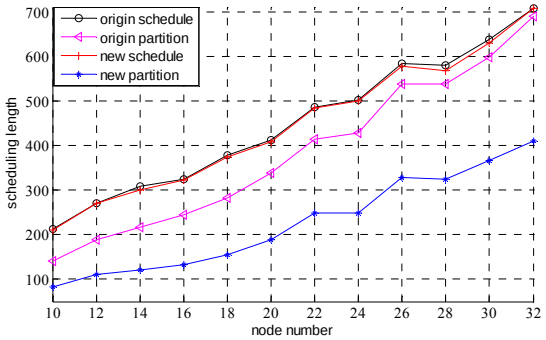


Figure 5. *A*=60,*CCR*=0.1
Hardware area is 60 and the Communication to Computation Ratio (CCR) is 0.1.

From figure 5, the original partitioning algorithm becomes more effective on the case of "*A*=60, *CCR*=1.0". By this token, the existing HW/SW partitioning algorithm only applies to that the hardware resource is of more abundant. The proposed algorithm works well, no matter the hardware size is small or large. The improvement is up to 45.4% over the existing algorithm on the graphs with 18 nodes.

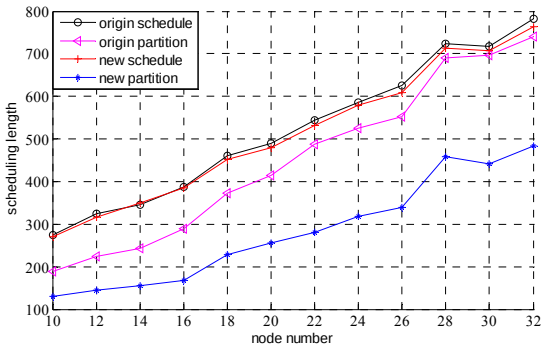


Figure 6. *A*=60,*CCR*=1.0
Hardware area is 60 and the Communication to Computation Ratio (CCR) is 1.0.

Figure 6 shows the result of "*A*=60, *CCR*=1.0". The result of the algorithm proposed here is better significantly for the improvement is nearly up to 37.2%. The improvement is the most satisfying in the node number 22, which the value is 42.4% approximately.

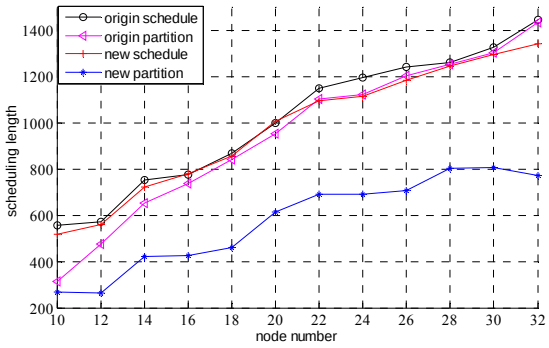


Figure 7. *A*=60,*CCR*=10.0
Hardware area is 60 and the Communication to Computation Ratio (CCR) is 10.0.

As shown in figure 7, the existing partitioning algorithm becomes worse with the increasing of granularity on the case of "*A*=60, *CCR*=1.0". But the proposed partitioning algorithm is almost not affected from the granularity, and the improvement is 37.9%. In the number of nodes 32, the improvement is the most obvious, 46.2%.

In summary, the scheduling length can be greatly reduced by the HW/SW partitioning algorithm proposed in the paper and the improvement is up to 38%. The contributions of our algorithms are as follows: firstly, the proposed partitioning algorithm can make full use of the hardware resource to minimize the scheduling length as much as possible. But the existing partitioning strategy can only work in the case of the hardware area abundant. Secondly, the proposed partitioning algorithm is universal, which can always minimize the scheduling length by utilizing the hardware area fully, no matter how much the extent of the differences in paths is. While the original partitioning algorithm is merely suitable for the case that the critical path is much longer than the other paths, and in the other cases, it almost does not work. At last, the proposed partitioning algorithm is more realistic for the hardware area is very limited in MPSoC. That is to say, the original partitioning algorithm can work in the ideal case rather than a common one.

V. Conclusions

In this paper, a task scheduling approach with priority mechanism has been proposed for multi-processor system, and a kind of universal HW/SW partitioning algorithm has been designed to further minimize the overall execution time. The scheduling algorithm can reduce unnecessary waiting time by assigning different priorities to different tasks on the basis of their out-degree and software execution time, so as to minimize the overall execution time. The HW/SW partitioning algorithm successfully shortens the execution time by moving the task with the highest benefit-to-area ratio in the critical path iteratively, no matter how much the hardware area. Experiment results shows that the novel algorithms are able to reduce the overall execution time up to 38% over the latest work.

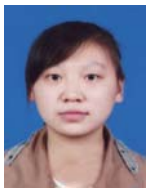
Acknowledgment

This work was supported by the National Science Foundation of China under Grant No. 60970016 and No. 61173032.

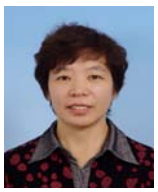
References

[1] M.B. Abdelhalim, S.E.-D.Habib, "An integrated high-level hardware/software partitioning methodology", Des Autom Embed Syst 15: pp.19-50, 2011.
[2] Wayne Wolf, Fellow, "Multiprocessor System-on-Chip (MPSoC) Technology", IEEE Transaction On Computer-Aided Design Of Integrated Circuits And Systems: Vol. 27, No. 10, October 2008.
[3] Cristina Boeres, Vinod E. F. Rebello, "Towards Optimal Static Task Scheduling for Realistic Machine Models: Theory and Practice", The International Journal of High Performance Computing Applications. Volume 17, No. 2, pp. 173-189, 2003 summer.
[4] YU-KWONG KWOK, ISHFAQ AHMAD, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", ACM Computing Surveys. Vol. 31, No. 4, December 1999.
[5] Wu Jigang, Thambipillai Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning", Information Processing Letters 98 pp.41-46, 2006.
[6] Dick, R.P. and Jha, N.K. Mogac, "A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems", In Digest of Technical Papers, IEEE/ACM International Conference on Computer-Aided Design, pp.522-529, 1997.
[7] Marisa L ó pez-Vallejo, Juan Carlos L ó pez, "On the hardware-software partitioning problem: System modeling and partitioning techniques", ACM Transactions on Design Automation of Electronic Systems (TODAES), vol.8, no.3, pp.269-297, 2003.
[8] Mei, B., Schaumont, P., AND Vernalde, S, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems", In Proceedings of the 11th ProRISC Workshop on Circuits, Systems and Signal Processing. 2000.
[9] Frank Vahid. "Partitioning sequential programs for cad using a three-step approach", ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 7, no. 3, pp.413-429, 2002.
[10] Wu Jigang, Thambipillai Srikanthan,Senior Member, et. al., "Algorithmic Aspects of Hardware/Software Partitioning: 1D Search Algorithms", IEEE TRANSACTIONS ON COMPUTERS.VOL. 59, NO. 4, APRIL 2010.
[11] Wu Jigang, Thambipillai Srikanthan, Tao Jiao, "Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design", Des Autom Embed Syst 12: pp.345-375, 2008.
[12] Henkel J, Ernst R, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques", IEEE Trans Very Large Scale Integr (VLSI) Syst 9(2):pp.273-289, 2001.
[13] Stitt, G. and Vahid, F, "The energy advantages of microprocessor platforms with on-chip configurable logic", IEEE Design & Test of Computers, vol.19, no.6, pp.36-43, 2002.
[14] Wan M., Ichikawa Y., Lidsky D., Rabaey J., "A power conscious methodology for early design space exploration of heterogeneous DSPs", In Procedings of the ISSS

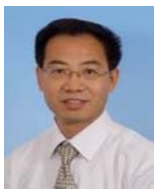
Custom Integrated Circuits Conference (CICC), pp.111-117, 1998.
[15] J. Mu, R. Lysecky, "Autonomous Hardware/Software Partitioning and Voltage/Frequency Scaling for Low-Power Embedded Systems", ACM Transactions on Design Automation of Electronic Systems (TODAES), vol.15, no.1, pp.1-20, 2009.
[16] Lysecky, R., "Low-power warp processor for power efficient high-performance embedded, systems", In Procedings of the Conference on Design Automation and Test in Europe (DATE), pp.141-146, 2007.
[17] Péter Arató, Zoltán Ádám Mann, András Orbán., "Algorithmic aspects of hardware/software partitioning", ACM Transactions on Design Automation of Electronic Systems (TODAES), vol.10, no.1, pp.136-156, 2005.
[18] Arato, P., Juhasz, S., Mann, Z.A., Orban, A., and Papp, D., "Hardware-Software partitioning in embedded system design", In Proceedings of the IEEE International Symposium on Intelligent Signal Processing, pp.197-202, 2004.
[19] Shrivastava Aviral,Kumar Mohit,Kapoor Sanjiv,Kumar Shashi and Balakrishnan M. , " Optimal hardware/software partitioning for concurrent specification using dynamic programming", In Proceedings of the IEEE International Conference on VLSI Design, pp. 110-113,2000.
[20] R. Niemann, P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming", Partitioning Methods for Embedded Systems, Design Automation for Embedded Systems 2 (2) pp.165-193 Special Issue,1997.
[21] K.S.Chatha, R. Vemuri, "Hardware-Software Partitioning and Pipelined Scheduling of Transformative Applications", IEEE Trans. Very Large Scale Integration Systems, vol. 10, no. 3, pp. 193-208, June 2002.
[22] K.S. Chatha, R. Vemuri. Magellan, "Multiway Hardware-Software Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs", Proc. Ninth Int'l Symp. Hardware/Software Codesign (CODES'01), pp. 42-47, 2001.
[23] T. Wiangtong, P.Y.K. Cheung, W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardwaresoftware Codesign", Design Automation for Embedded Systems, vol. 6, no. 4, pp. 425-449,2002.
[24] Karam SC, Ranga V, "Hardware-software partitioning and pipelined scheduling of transformative applications", IEEE Trans Very Large Scale Integr (VLSI) Syst 10(3):pp.193-208, 2002.
[25] Tahaee, S.A., Jahangir, A.H. and Habibi-Masouleh, H., "Improving the performance of heuristic searches with judicious initial point selection", In Proceedings of the 5th IEEE International Symposium on Embedded Computing, pp.14 - 19, 2008.
[26] Zou Yi, Zhuang Zhenquan,Chen Huanhuan, "HW-SW partitioning based on genetic algorithm". Proc. 2004 Congr. Evol. Comput. CEC2004, pp.628-633, 2004.
[27] Wu Jigang and Thambipillai Srikanthan, "Algorithmic aspects of area-efficient hardware/software Partitioning",J Supercomput, pp.38:223-235, 2006.
[28] Wu Jigang, Thambipillai Srikanthan, Chengbin Yan, "Algorithmic aspects for power-efficient hardware/software partitioning", Mathematics and Computers in Simulation , pp.79 :1204-1215,2008.
[29] Wu Jigang, Thambipillai Srikanthan, Ting Lei, "Efficient heuristic algorithms for path-based hardware/software Partitioning", Mathematical and Computer Modelling, pp.51:974-984,2010.
[30] Hassan Youness, Mohammed Hassan, Keishi Sakanushi, Yoshinori Takeuchi,Masaharu Imai, Ashraf Salem, Abdel-Moniem Wahdan, Mohammed Moness, "A High Performance Algorithm for Scheduling and Hardware-Software Partitioning on MPSoCs", Design & Technology of Integrated Systems in Nanoscal Era, 2009.
[31] Yu-Kwong Kwok, Ishfaq Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms", Journal of Parallel and Distributed Computing 59, pp.381-422, 1999.
[32] S. Banerjee, E. Bozorgzadeh, N. Dutt, "Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration", Proceeding of 42nd DAC'05 USA, pp.335-340, ACM, 2005.



Honglei Han was born in Hebei province in 1987. She received the BSc degree in Computer Science and Technology from Northwest Normal University, China, in 2009. She received her MSc degree from Tianjin Polytechnic University, China, in 2012.
Her research interests include in reconfigurable VLSI design and algorithmic aspects in hardware/software co-design.



Wenju Liu was born in Tianjin in 1963. She received the BSc degree in Computer from NanKai University, China, in 1985. Since 1985, she worked in the School of Computer Science and Software, Tianjin Polytechnic University, China.
Her research interests include in enterprise informatization and computer network security.



Wu Jigang was born in Jiangsu province in 1963. He received the BSc degree in computational mathematics from Lanzhou University, China, in 1983, and the doctoral degree in computer software and theory from the University of Science and Technology of China (USTC) in 2000.
He was with the Department of Computer Science of Lanzhou University, China, from 1983 to 1993, as an assistant professor followed by lecturer. He was with the Department of Computer Science and Engineering of Yantai University, China, from 1993 to 2000, as a lecturer followed by associate professor. He was with the Center for High Performance Embedded Systems, School of Computer Engineering, Nanyang Technological University, Singapore, from 2000 to 2009, as a postdoctoral fellow followed by research fellow. In 2009, he joined as a Tianjin distinguished professor and Dean of the School of Computer Science and Software, Tianjin Polytechnic University, China.
He has published more than 120 technical papers including journals in the IEEE Transactions, IEE Proceedings, and other reputed international journals. His research interests include in reconfigurable VLSI design, hardware/software co-design, parallel computing, and combinatorial search. He is a member of the IEEE.

Guiyuan Jiang was born in Shandong province in 1985. He received the BSc degree in Computer Science and Technology from Northwest University for Nationalities, China, in 2008. He received his MSc degree from Tianjin Polytechnic University in 2011, and now he is a PhD candidate. His research interests include algorithmic aspects in hardware/software co-design and parallel computing.