16th Int. Conference on Parallel and Distributed Systems, August 13-15 2003, pp. 181-186, Reno Nevada USA

Heterogeneous Hardware-Software System partitioning using Extended Directed Acyclic Graph

Matthew Jin and Gul N. Khan
Electrical and Computer Engineering,
Ryerson University, 350 Victoria Street,
Toronto, Ontario, Canada, M5B 2K3
jinma,gnkhan@ee.ryerson.ca

ABSTRACT
In this paper, we present a system partitioning technique in which the input system specification is based on C++ language. The proposed technique processes data and precedence dependencies simultaneously in one graph representation DADGP, which is an extension of Directed Acyclic Graph (DAG). The DADGP (Directed Acyclic Data dependency Graph with Precedence) based partitioning technique minimizes the communication overhead as well as overall system execution time under real-time deadline. It also tries to minimize the cost of target system in terms of hardware area.

Keywords: Hardware/Software Co-design, System partitioning, Mapping and Scheduling.

INTRODUCTION

In order to meet the market demand, designers now need to produce complex embedded systems in a shorter period of time. Therefore, waiting until the final implementation of the system before understanding the hardware-software interactions is no longer acceptable. Hardware-software tradeoffs must be analyzed early in the design cycle to reduce the design and development time. However, current hardware-software co-design techniques cannot effectively handle hardware and software integration [1].

Edwards and Forrest addressed the hardware-software partitioning by finding the bottleneck in the software and moving that critical region to hardware [2]. Their method does not take data transfer into consideration, and overall improvement has not been as good. Mapping and Implementation-Bin Selection (MIBS) [3] partitioning method uses Directed Acyclic Graph (DAG) to represent computational blocks and data dependencies. The Global Criticality Local Phase (GCLP) algorithm first traverses the DAG and maps each node to either hardware or software to minimize execution time and hardware area. However, the algorithm does not take communication overhead into account when calculating the objective function for the execution time. A similar partitioning algorithm was also proposed by Ondghiri and others [4] with a variation in the different search technique to explore hierarchical design space. Their result showed that the optimal solution exists at some level between the two extremes of high and low granularity level.

Most of the partitioning algorithms employed either dependency or execution graph as an input to generate a new set of partitioned hardware and software blocks. These algorithms seem to work in ideal cases, and are not applicable in most real applications due to restricted specifications, constraints, and complexity. The partitioning algorithm presented in this paper has tried to overcome some of these limitations by using C++ as system specification language and by considering a number of new system constraints during partitioning. We present some improvements in the hardware-software co-synthesis components. Our partitioning algorithm processes C++ based system specification to generate optimal hardware-software partition set. The use of C++ language as a system specification language has allowed flexible design representation in a unified design environment. The algorithm converts the system specification into a Directed Acyclic Data Dependency Graph with Precedence (DADGP) for further analysis. The DADGP is then mapped to appropriate software and hardware modules, and scheduled for further verification. If the scheduled result is satisfactory under certain constraints (e.g. improvement in the overall performance), the hardware mapping is accepted, and the algorithm continues until all the given constraints (hardware area, overall execution time, cost) are satisfied.

SYSTEM SPECIFICATION

Defining a design specification to describe system level behavior is a challenging problem. It requires high abstraction as well as fine details to reduce ambiguities during hardware software co-design. Traditionally, these specifications were written in plain English to describe its system constraints and functionalities. However, system description in English language can bring ambiguities and misinterpretations among designers creating high Non Recurring Expenses. A lot of research has been done to create a unified co-design environment by proposing new design specification languages [5]. The idea is to capture the system specification more accurately by augmenting the HDL and high level programming languages to describe the entire system. MSC, SDL, PMSC and SystemC are some examples of theses languages [5, 6].

To incorporate the possibility of ever-changing demand, we use textual representation for specification and graphical representation for system partitioning and scheduling. C++ is used as the initial system specification language and through profiling, it is converted into a DADGP representation for partitioning and scheduling. The given system specification is translated into C++ in the form of modules so that each module can be evaluated, and mapped to the process space during profiling. For example, during the block-matching step of MPEG, sum-of-absolute-differences are calculated to measure the similarity between the macroblock and its search area. If one needs to develop an IP block to calculate the sum-of-absolute-differences, equation 1 is first translated into C++ specification. The partitioner uses the block matching C-code with its hardware module information (Figure 1a) to generate an initial DADGP (Figure 1b).

PE component	PE0	PE1
sub	6	5
abs	4	8
sum	3	2

Figure 1a: Hardware Module Data

$$\sum_{1\le i}\sum_{j\le n}\left|M(i,j)-S(i_x,j_y)\right| \quad \text{.....equation(1)}$$

M(i,j) is the object, and S(i,j) is the search space matrix. The block matching C code

```
for ( i=0 ; i<= 16 ; i++) {
      for (j=0;j<=16;j++) {
      temp= abs(sub (M(i,j),S(i,j)));
      result=sum(result,temp);
      }
}
```

HARDWARE-SOFTWARE PARTITIONING

The system-partitioning algorithm can be subdivided into three major components, namely profiling of C++, LD_path search, and mapping of LD_path and Scheduling. LD_path search and its mapping are repeated until an adequate solution is reached. The partitioning system flow chart is shown in Figure 2.

Following assumptions are made for our partitioning technique.

- Initial target architecture is of one processor core
- Each DADGP node is executable with at least one PE
- All the nodes of DADGP can be implemented in hardware and software.
- Inter PE communication is estimated by the amount of data transferred, and the transfer rate is constant.
- Communication overhead is zero between the two PE if that two nodes are executed by the same PE
- The partitioner has all the necessary information including: execution time of each node for different PEs, cost of adding PE, and inter PE communication overhead, as well as initial system constraints (required execution time, maximum hardware area).

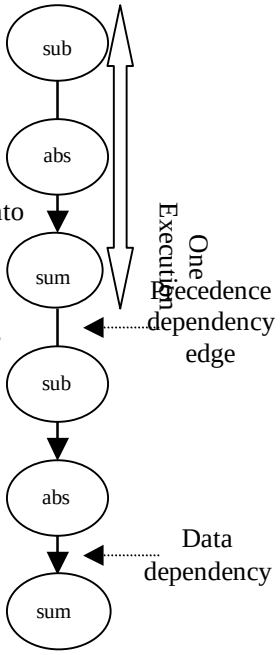

Figure 1b: DADGP Block Matching

Profiling

If one can translate the system specification into software that fulfills the deadline requirements on the target platform then that is the optimal solution. However, in most cases, all software solution is not possible for real-time embedded systems, and hardware software partitioning must be performed. It is also vital to execute the C++ system specification on the actual target platform to accurately collect the profiling data. The software profiling is a useful step to check for all software solution. The profiler translates each module of C++ system specification into nodes with the following information:

- Execution time of each module
- Start and completion time of each module
- Amount of data transfer
- The caller(s) of the module
- The child(s) of the module
- Module identification
- Execution order

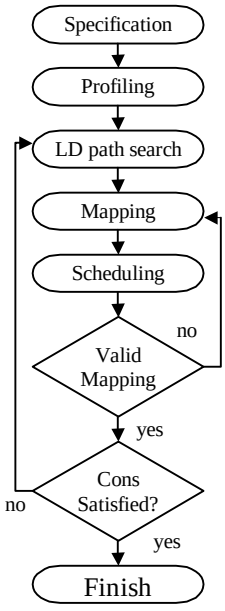

Figure 2

The profiler uses the above information to generate the DADGP. The unique characteristic of DADGP is that it contains both data and precedence dependent edges to represent the system. The data dependency edge is represented with an arrow symbol as shown in Figure 1b. Two nodes are connected with data dependency edge when they have a producer-consumer relationship. The precedence dependency edge is represented by a line to connect two independent nodes together. The precedence dependency captures the order of execution between nodes and such nodes can be executed in parallel if desired. A more detailed discussion on DADGP is provided as part of the mapping and scheduling section.

Mapping and Scheduling

DADGP is a super set of DAG with the only difference of having two types of connecting edges. Our contribution to DAG is the introduction of precedence dependency edges to explicitly represent the independence and the execution orders between nodes. The DAG representation is not algorithmically friendly to capture non-parallel executions of independent nodes. Exposing the independence and introduction of parallelism between independent nodes are not always the best decision when optimizing the execution time of a system due to inter-PE communication overheads. The incorporation of DADGP to our partitioning method has allowed us to only expose the necessary parallelism to capture a wide rage of solutions.

The Longest Delay path (LD_path) represents the longest execution route in a DADGP. LD_path depends on the number of node hops, execution time of each node, and its corresponding inter PE communication overhead as given in equation 2.

$$\text{LD\_time} = \sum_{i=1}^{N} n_i + \sum_{j=1}^{Q} e_j \quad \text{.....equation (2)}$$

where node $n_i \in \{ n_1 \dots n_N \}$ that must be connected with any type of edges $e_j \in$ E (all edges Q).

LD_path can be found with the following equation.

LD_Path= Max [LD time (p_k)] .....equation (3)

where M is the total number of path in a given graph and k varies from 1 to M.

p=$\{ n_i \dots n_N , e_j \dots e_{N-1} \}$ is a path from one root to any ending node.

P=$\{ p_1 \dots p_M \}$ is a collection of all paths from one root to any ending node.

Finding an LD_path in DADGP is similar to finding a bottleneck in the system. The LD_paths can be used to improve the overall execution time by mapping one of the LD_path nodes to hardware. A repeated searching and mapping of DADGP reduces the search space, and improves the convergence rate for an optimal solution. The LD_path searching algorithm is given below:

```
Assume L = {l_1 ... l_N}
for ( i = 1; i++; i <= N )
      P = Find_path( l_i )
max = 0;
for ( i = 1; i++; i <= M ){
      temp = LD_time( p_i );
      if ( temp > max ) then {
      max = temp;
      path = p_i ;
      }
}
```

Mapping and scheduling of DADGP is the most sophisticated and important step of our partitioning algorithm. One of the nodes in the LD-path is mapped to the optimal hardware. However, to make such critical decision for a PE to be a dedicated hardware unit or a software task, following factors are taken into consideration.

- Parallelism in DADGP nodes
- PE resource limitation
- PE Execution time of nodes
- Inter PE communication
- Hardware area (cost)

The partitioning algorithm starts by finding the LD_path in a given DADGP, and the execution time of LD_path is also calculated. All the nodes in the LD_path are mapped to appropriate hardware one at a time and scheduled to calculate the overall system execution time. A node that allows maximum improvement of system execution time is finally mapped as hardware, and the DADGP is updated according to the final mapping decision. This process is repeated for all the LD_paths as explained in the partitioning algorithm given below.

Initialize the following:

```
LD_path = {n_1 ... n_A , e_1 ... e_B} /* A = number
of node, B = number of edges*/
Previous_system_EXE = ∞;
while( (System_EXE > Required_EXE) AND
(Current_HW_area > Required_HW_area)){
      LD_path = Find_LD_path(graph);
//Find the LD_path
      LD_path_EXE = LD_time(LD_path);
// Calculate execution time of LD_path
      Min_EXE = ∞;
   for ( i = 1 ; i++ ; i <= A ){
      G = map(DADGP, n_i );
/*G is a new DADGP with the node n_i ∈
LD_path nodes mapped as HW.*/
      (EXE, S, G_prime) = schedule(G);
// G_prime = updated G, S = schedule of
// G,EXE = execution time of G
      if ( EXE < Min_EXE ) then{
            Min_EXE = EXE;
            graph = G_prime;
            Final_S = S;
      }
   }
if (previous_system_EXE < Min_EXE )
then return(graph);
previous_system_EXE = Min_EXE;
}
return (G, Final_S);
```

The working of scheduling function **schedule** (G) is summarized below:

a) Start scheduling from the root of DADGP

b) Traverse down the tree and schedule the earliest starting time node
c) If a node is connected with precedence dependency edge, check to see whether exposing parallelism can eliminate that edge. If a precedence dependency edge is eliminated, the structure of the DADGP may change and some nodes can be disconnected from the original graph resulting in two separate DADGP. In this case, the new root of the disconnected DADGP must be combined to make one DADGP by connecting it self and the original root to a new dummy node called "start".
d) If multiple descendents (or roots) exist, force schedule all descendents (or roots) by adding necessary PE if required
e) Update PE resource library and generate the total execution time by using the following equation:

EXE = LD_path_EXE – n_i _exe + HW_exe + IPC – Hidden_node_EXE

where, n_i _exe = execution time of a node that is currently in interest, HW_exe = HW execution time of node n_i.

IPC = value introduced by mapping node n_i to additional HW, hidden_node_EXE = smaller execution time value between two parallel modules.

EXPERIMENT RESULTS

The following example demonstrates the ability of our partitioning algorithm to consider multiple descendants indirectly without the added calculation complexity. Considering GDL algorithm [7], and its complexity, the scheduling inherits the typical weakness of conventional list scheduling. The GDL scheduler improved on list scheduling by quantifying the scheduling effects on the

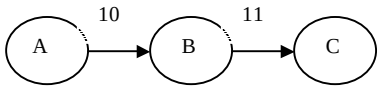

Figure 3a: Without Precedence

	P0	P1
A	1	2
B	2	2
C	20	1

Figure 3b: Module Data Table

descendants of a node. However, considering just the first descendants are not enough to estimate the global effect. Consider the example shown in Figure 3a and 3b. The scheduled result from the GDL heuristic is presented in Figure 4a in the form of a Gantt chart. When node A is scheduled on P0, the GDL fails to consider the effect of processor selection for node C. The final scheduling result suffers from a large inter-PE communication overhead between A and B. It is however obvious that a better scheduling result will be to assign all the nodes to P1 as shown in Figure 4b.

The DAG of Figure 3a does not have any precedence dependency edges and therefore there are no parallelisms. Obtaining the LD_path of Figure 3a is simple because there is only one path. Assuming that P0 is target processor and P1 is an additional PE hardware, the initial DADGP of the system is shown in Figure 5 (note that it is all software solution given by the profiler). The partitioner will first move node C to P1 to reduce the total execution time (C is the Min[EXE]). In the next iteration, node B is moved to P1 to reduce the inter communication time with C. In the third iteration, node A is finally moved to P1 to reduce the inter PE communication with B, and the optimal scheduling is obtained as shown in Figure 4b.

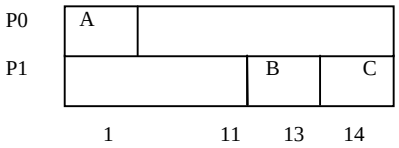

Figure 4a: GDL Scheduled Results

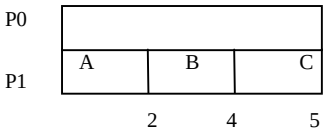

Figure 4b: Optimal Scheduling

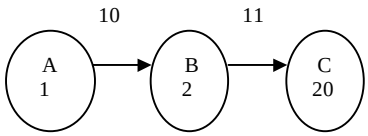

Figure 5: All Software Solution

Now we compare our partitioning method with the partitioning algorithm presented by Yen and Wolf [8]. The example is taken from Hou and Wolf [9] that contains DAG and a data table as shown in Figure 6. The period 240 indicates the number of execution cycle of the DAG and deadline values indicate the finish time constraint of a node. In partitioning the DAG of Figure 6, the partitioning algorithm by Yen and Wolf's suggests that all nodes be scheduled on HW1 to meet the deadline constraints of 350 units/cycle. Their solution gives an execution time of 322 units with a cost of $50. The overall system execution time is equal to 77280 (240*322) time units. Furthermore, the fastest optimal solution obtained was to schedule all tasks to HW2 [8].

This results in an execution time of 147 units with a cost of $100 while the overall execution time is equal to 35280 (240*147) time units.

	PE		
Tasks	**SW** Cost=$20	**HW1** Cost=$50	**HW2** Cost=$100
A	76	42	20
B	460	185	80
C	110	32	22
D	82	63	25

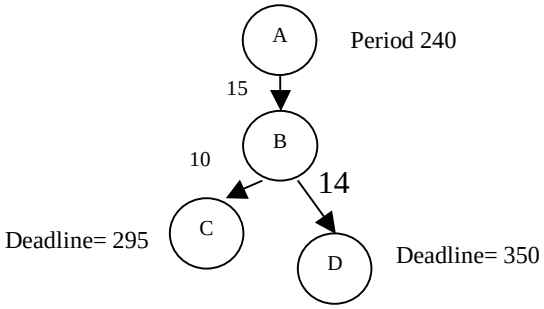

Figure 6: DAG and Data table from Hou & Wolf

If one needs to design even higher performance system with tight deadlines by using the same data as in Figure 6, our DADGP partitioning algorithm have advantages over the existing partitioning algorithms. The DAG can be easily transformed to DADGP as given in Figure 7 showing just two cycles of operation. Initially, all the nodes are mapped and scheduled to SW (target processor) and the execution time of one cycle is 728 time units with a cost of $20. The overall system execution time is equal to 174720 (240*728) time units. Our algorithm starts by mapping nodes that will optimize overall execution time and minimize cost. Eventually, the algorithm will take advantage of the repeated execution components. It will then map node A to HW1 from the 2nd execution cycle. This mapping will force the scheduler to add additional HW1 to expose parallelism as shown in Figure 8a and 8b.

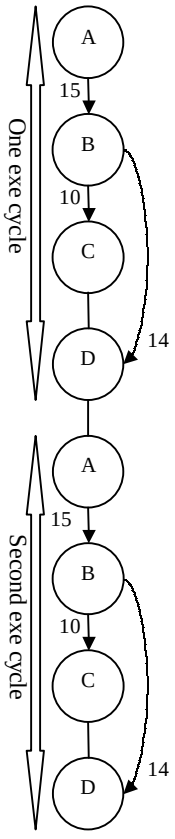

Figure 7

The scheduling of mapped DADGP of Figure 8a is shown in Figure 8b, where the execution of each cycle is still 322 time units. However the overall system execution time has improved significantly due to exploring parallelism. The DADGP algorithm exposes various combinations of solutions from decreasing order of execution time and cost, to be selected by the designer according to their demand. At some point during the execution of our algorithm, HW2 will be mapped and scheduled for each parallel execution. In this case, the overall execution time of the system is equal to 17640 (240/2 * 147) time units. This gives 50% increase in speed as compared to the optimal solution provided by Yen and

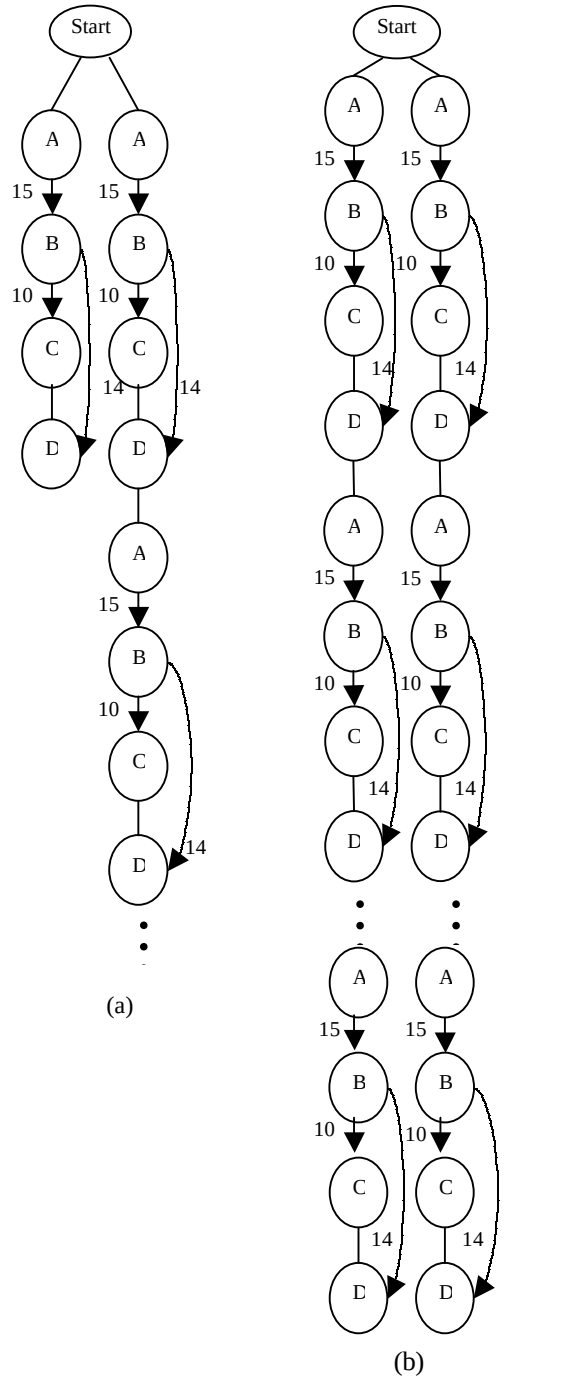

Figure 8: Mapped and Scheduled DADGP

Wolf. However, such performance gain comes with an increase in price, as the cost is now $200 due to the addition of extra HW2. The main advantage of DADGP partitioning algorithm is its ability to expose wide range of solution space, so that the designer can make intelligent decisions. Ultimately, the algorithm will continuously add more HW2 modules to (up to 240 units) allow 240 simultaneous executions. This solution will have overall system execution time of 147 units to execute the DAG graph (Figure 6) 240 times in one cycle. Nonetheless, such extreme solutions will never be exposed unless the constraints allow putting that many hardware.

CONCLUSION AND FUTURE WORK

In this paper, we have introduced a new way of using C++ as a specification language for hardware-software co-design. The proposed approach of system specification seems to be useful for many applications that can be described as a cluster of communicating processes with iterations. The proposed algorithm is able to explore a variety of partitioning solutions compare to other research work presented so far. Its ability to fine step optimize towards the final optimal partition set is a key in discovering a variety of solutions with diverse costs. This quality was obtained by simultaneously manipulating data dependency and precedence dependency as DADGP. The complexity of the proposed algorithm is also comparable with existing methods, and can be easily extended to consider different constraint conditions.

Co-synthesis phase of the design is a very fertile research area, especially when it comes down to automating the embedded system design. Many people are skeptical about such design because it would require emulating human designers, who typically invent new solutions to the design problems they encounter. Therefore the proposed algorithm has tackled the co-synthesis problem from the IP libraries approach. This approach limits the design space to the amount of library components. But it is more realistic approach than building an AI system. To complete the hardware-software design flow, the proposed co-synthesis mythology must include automatic hardware software generation to its implementation platform. Such work is under investigation with a Rapid Prototyping Platform (RPP) system consisting of ARM7TDMI processor core and Xilinx FPGA for multiple hardware implementations. Furthermore, for more accurate partitioning result, it is important to provide accurate profiling data to the partitioner. Therefore, building an accurate and diverse hardware IP library is also important to make the automation of architectural synthesis from hardware-software partitioner a reality.

ACKNOWLEDGEMENTS

This research is supported by a grant from NSERC Canada. The authors would also like to thank CMC for providing hardware software co-design CAD tools.

REFERENCES

[1] Lee Garber, David Sims, “In pursuit of Hardware-Software Codesign”, Computer, vol.31, No.6, pp. 12-14, June 1998.

[2] M. D. Edwards, J. Forrest, “Hardware/software partitioning for performance enhancement”, Proc. IEE Colloquium on Partitioning in Hardware Software Codesign, pp. 2/1-2/5, February 1995.

[3] E.A. Lee, A. Kalavade “The extended partitioning problem: hardware/software mapping and implementation-bin selection”, Proc. 6th IEEE Inter. Workshop on Rapid System Prototyping, pp. 12-18, 1995.

[4] H. Ondghiri, B. Kaminska and J. Rajski, “A hardware/software partitioning technique with hierarchical design space exploration”, Proc. IEEE Conference on Custom Integrated Circuits, pp. 95-98, 1997.

[5] G.D. Micheli and R.K. Gupta, “Hardware/Software Co-design”, Proc. IEEE, Vol. 85, No.3, pp. 349-365, March 1997.

[6] SystemC Inc. See http://www.systemc.com

[7] G.C. Sih and E.A. Lee, “A compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures”, IEEE Trans. Parallel and distributed systems, Vol. 4, No.2, pp. 175-187, February 1993.

[8] T-Y. Yen and W. Wolf,” Sensitivity-driven co-synthesis of distributed embedded systems”, Proc. 8th Inter. symposium on System Synthesis, pp. 4-9, 13-15 Sep. 1995.

[9] J. Hou and W. Wolf, ”Process Partitioning for Distributed Embedded Systems”, 4th Int. Workshop Hardware-software Codesign, pp. 70-76, March 1996.