Resource Mapping and Scheduling for Heterogeneous Network Processor Systems

Liang Yang, Pavel Ghosh, Devesh Sinha, Arunabha Sen and Andrea Richa
Computer Science and Engineering Department
Arizona State University
Tempe, Arizona 85287-8809
{liang.yang, pavel.ghosh, devesh, asen, aricha@asu.edu}
Tushar Gohad
MontaVista Software
{tgohad@mvista.com}

Abstract

Task to resource mapping problems are encountered during (i) hardware-software co-design and (ii) performance optimization of Network Processor systems. The goal of the first problem is to find the task to resource mapping that minimizes the design cost subject to all design constraints. The goal of the second problem is to find the mapping that maximizes the performance, subject to all architectural constraints. To meet the design goals in performance, it may be necessary to allow multiple packets to be inside the system at any given instance of time and this may give rise to the resource contention between packets. In this paper, a Randomized Rounding (RR) based solution is presented for the task to resource mapping and scheduling problem. We also proposed two techniques to detect and eliminate the resource contention. We evaluate the efficacy of our RR approach through extensive simulation. The simulation results demonstrate that this approach produces near optimal solutions in almost all instances of the problem in a fraction of time needed to find the optimal solution. The quality of the solution produced by this approach is also better than often used list scheduling algorithm for task to resource mapping problem. Finally, we demonstrate with a case study, the results of a Network Processor design and scheduling problem using our techniques.

This research was supported in part by the Center for Embedded Systems (CES) at Arizona State University.

Categories and Subject Descriptors

B.8.2 [**Hardware**]: [Performance Analysis and Design Aids]

General Terms

Design Performance

Keywords

Network Processor, HW-SW Partitioning, Randomized Rounding, Codesign

1. Introduction

To meet twin goals of performance and flexibility, Network Processors (NP) were introduced by several vendors a few years ago. Evolution in Network Processor family in the last few years has seen increasing heterogeneity in the architectural design. This is evidenced by the introduction of specialized co-processors for classification and security and content addressable memory (CAM) for faster search. Such heterogeneity contributes to added complexity for two problems, (i) hardware-software co-design of Network Processors and (ii) mapping of application components to appropriate resources for optimal performance. Network Processors are required to support multiple applications, such as header parsing, table lookup, encryption/decryption for virtual private networks (VPN), network address translation (NAT) and voice processing. In the first problem, one needs to find the optimal architecture that can support all the specified applications. In the second problem, given the architecture, one needs to find the optimal mapping of the application components (task) to the available resources. In both the cases, one needs to find a task-to-resource mapping. The goal of the first problem is to find the task-to-resource mapping that minimizes the design cost subject to all design constraints. The goal of the second problem is to find the task-to-resource mapping that maximizes the performance subject to all architectural constraints.

The first problem was studied extensively by Thiele *et al.* in [11]. In their model of design space exploration problem, each application (e.g, encryption/decryption, voice processing) is represented by a directed acyclic graph (DAG). The nodes of this graph represents the application components (or tasks). A set of resources, such as general purpose processors, co-processor for classification (classifier), co-processor for security, micro-engines and others are available for execution of the tasks. The execution time of each task on each resource is known. The goal of the task-to-resource mapping problem is to find the optimal mapping that satisfies all the design and performance constraints. Figure 1 shows the DAGs corresponding to three common Network

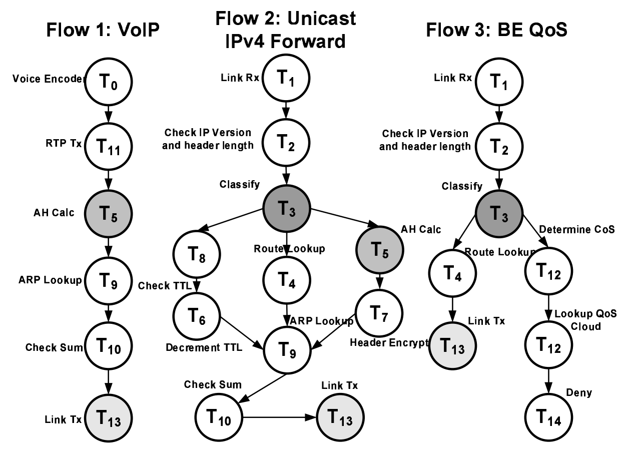


Figure 1. Task graphs for different flows

Processor applications, Voice-over-IP, IPv4 Forwarding and Best Effort Quality-of-Service. The bipartite graph of Figure 2 shows the execution time of a task T_i on a resource R_j. The execution time of task T_i on a resource R_j is shown on directed edge of the bipartite graph from T_i to R_j.

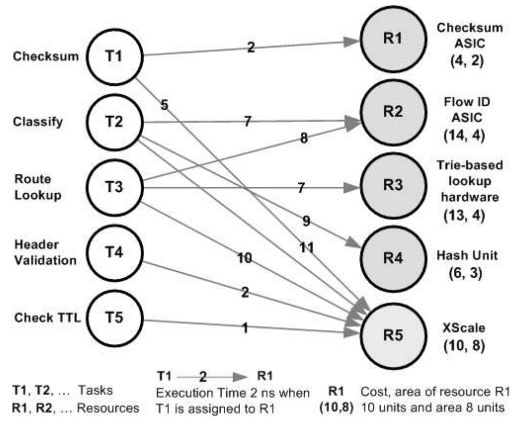
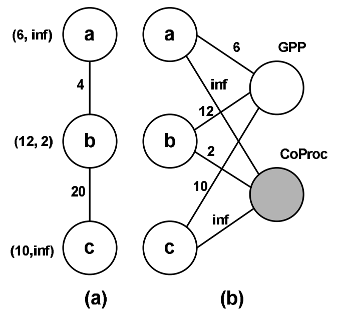


Figure 2. Task-Resource Mapping Graph

The second problem was studied by Ramaswamy *et al.* in [10]. In the first part of their paper they conduct extensive analysis of the application data to build the DAG that will be used for mapping. The nodes of this DAG is a set of instructions that may be viewed as the tasks in the first problem. In the second part of the paper [10], the authors provide an algorithm for mapping application DAGs to NP Architectures. This is a greedy algorithm based on widely known *list scheduling* scheme. Unfortunately, this greedy scheme can produce solutions those are far from optimal. Consider the DAG example shown in Figure 3. The DAG



	List Scheduling			ILP Solution			RR Algorithm Solution		
Task	Resource Mapped	Start Time	Finish Time	Resource Mapped	Start Time	Finish Time	Resource Mapped	Start Time	Finish Time
a	GPP	0	6	GPP	0	6	GPP	0	6
b	CoProc	10	12	GPP	6	18	GPP	6	18
c	GPP	32	**42**	GPP	18	**28**	GPP	18	**28**

Figure 3. DAG and Mapping Solutions

is comprised of three nodes (tasks), a, b and c. The tasks have to be mapped to two different resources, a General Purpose Processor(GPP) and a Co-Processor (CP). The CP is a specialized unit designed to execute the task b and it takes 2 units of time for the execution. Because the CP is a specialized processor for the task b, it may not be able to execute tasks a and c. This is captured in the bipartite graph model by assigning a weight of infinity to the directed edge from a to CP and c to CP. The execution times of three tasks on the GPP are 6, 12 and 10 respectively (shown in the Figure 3). The communication costs from GPP to CP are 4 and 20 time units for task pairs (a, b) and (b, c), respectively. Communication cost is assumed to be zero for pair of tasks executing on the same processor. The table in Figure 3 shows the mapping and start, finish times of the tasks using the list scheduling algorithm [10], the optimal solution and the randomized rounding (RR) technique based solution. In case of the list scheduling algorithm, task b will be mapped to the CP (since, communication cost(4) + processing cost on CP(2)=6 < processing cost on GPP(12)). Thus the greedy list scheduling technique will be trapped in a local minimal point. In this case, this local optimal point (finishing time = 42) is much worse than the global optimal point (finishing time = 28) obtained from the RR algorithm.

Clearly this approach does not find the optimal solution and in fact the difference between solution produced by this approach and the optimal solution can be arbitrarily large. For this reason, in this paper we present a Randomized Rounding (RR) based approach for the solution of the task-to-resource mapping and scheduling problem. We evaluate the efficacy of our RR approach through extensive simulation. The simulation results demonstrate that this approach produces near optimal solutions in almost all instances of the problem in a fraction of time needed to find the optimal solution.

The mapping and scheduling problems in embedded systems have been studied extensively by the research community in recent years due to its importance in determining the performance and the cost of the system. Most of the studies in this arena have focused their attention on the optimal design of systems that supports a *single application* [1, 6, 7]. However, a Network Processor System (NPS) may have to support multiple applications of the type mentioned earlier. Although a vast majority of the mapping and scheduling studies focus on single application systems and as such they are not particularly useful in the Network Processor domain. There have been only a few studies, where the issues of multi-application systems have been addressed [5, 8]. Since our focus is on the design of Network Processor systems, we will discuss efforts undertaken by others in multi-application domain, but will refrain from discussion of single-application systems. Mapping and scheduling of multi-application systems may be defined in one of the following two ways [5]:

> Definition 1: Given a set $APP = \{APP_1, APP_2, ..., APP_k\}$ of applications each specified by a DAG, where each application APP_j has a set of constraints (e.g., timing constraint D_j, area constraint A_j, etc.), find the mapping that minimizes the design cost (in monetary terms) while satisfying all the design constraints.

> Definition 2: Given a set $APP = \{APP_1, APP_2, ..., APP_k\}$ of applications each specified by a DAG, where each application APP_j has a set of constraints (e.g., timing constraint D_j, area constraint A_j, etc.), find the mapping that maximizes system performance while satisfying all the design constraints.

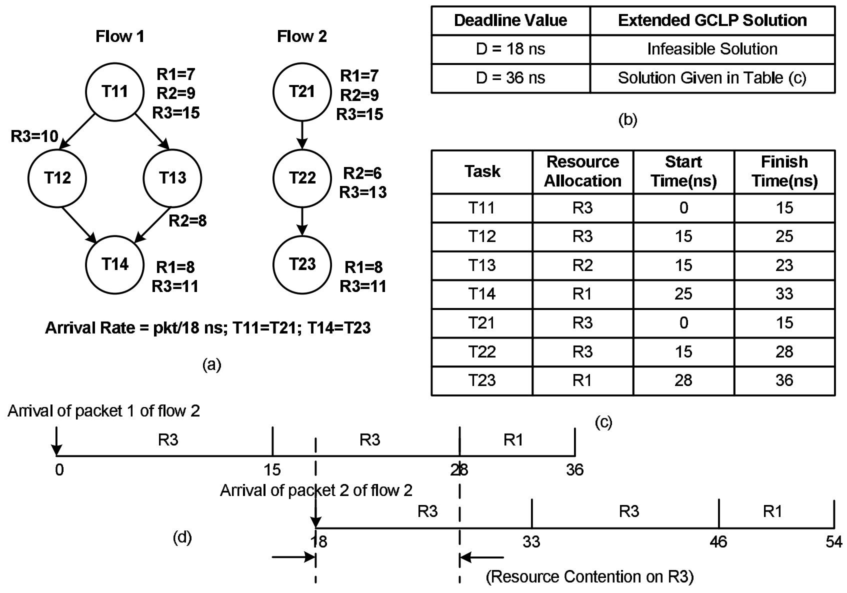
The first definition can be used before the system is designed and the second definition can be used after the system is designed. A profile-guided automated mapping compiler was developed for runtime performance enhancement in [12]. In addition to [10], this may be viewed as an example of the second definition.

In addition to supporting multiple applications, Network Processor systems have to process a stream of packets continuously arriving at its input ports. To meet performance goals, it may be necessary to allow more than one data unit to be inside the embedded system at any given time. A *data unit* is smallest chunk of data on which the system operates. In a networking application, the data unit may be a *packet.* Although the notion of a *packet* is associated with the networking domain, we will use it in a much broader sense in this paper and will use the terms *data unit* and *packet* interchangeably. The possibility of multiple data units being inside the embedded system at the same time adds to the complexity of the design process, as it opens up the possibility of resource contention between successive data units. We propose two techniques for detection and elimination of such resource contention. To the best of our knowledge, such resource contention issues were not studied in earlier papers.

Kalavade et al.[5] were one of the earliest researchers to study partitioning and scheduling problem for multi-function (multi-application) systems. In their model they assumed that only one data unit (or packet) will be inside the embedded system at any given time. However, this is a strong assumption and it may be difficult to meet in practice. To elaborate the issue, we provide the following example.

We assume a network processor has to process data at the line rate of 20 Gbps and the packet size is 45 Bytes with no inter-packet arrival gap. The NP has only 18 nanoseconds to process each packet. This scenario is illustrated in Figure 4. The task graphs corresponding to two applications, the resources on which the tasks can be executed and the time needed for execution of the tasks on the resources is shown in Figure 4(a). From the figure, the task T_{11} can be executed on resources R_1, R_2 and R_3 and it takes 7, 9 and 15 nanoseconds respectively. With Kalavade's assumption of only one packet inside the system at any given time, there will be *no feasible solution.* On increasing the deadline from 18ns to 36ns, all the tasks can be processed. However, increasing the deadline will be result in existence of a second packet in the system before the first one has left. There may be resource contention between the packets within the system now. Kalavade's Extended GCLP (Global Criticality Local Phase)approach [5] does not provide any safeguard against this as shown in 4(c).If the packets belong to application 2 (specified by task graph T2), the first packet will use resource R_3 from time 0ns to 28ns (0 to 15 for executing T_{21} and 15 to 28 for T_{22}) and R_1 from 28ns to 36ns. The second packet that enters the NP system at time 18ns, will require the resource R_3 between the times 18ns to 46ns, (for processing T_{21} from 18 to 33 and T_{22} from 33 to 46). As the first packet needs R_3 from 0ns to 28ns and the second packet needs it from 18ns to 46ns, there is resource contention between these two packets. In section 3.2, we provide mechanisms for detection and elimination of such resource contentions.

The contribution of this paper is twofold. First, we propose a methodology for finding the least cost mapping of tasks to resources, satisfying all the constraints. As a first step toward partitioning, an Integer Linear Programming (ILP) formulation is developed from the system level specification. To reduce computational time of ILP, the integrality constraints are relaxed and the corresponding LP is solved. Randomized rounding technique is used to convert the fractional values assigned to the variables by the LP to integral values. Second, the initial solution is evaluated for resource contentions. If this initial solution is resource contention free, we are done. If this solution is not free from resource contention, then we change the deadline for completion of tasks in a systematic way and repeat the entire process. This process is discussed in detail in section 3. Finally, we evaluate the efficacy of our methodology through extensive simulation using the software package



Deadline Value	Extended GCLP Solution
D = 18 ns	Infeasible Solution
D = 36 ns	Solution Given in Table (c)

(b)

Task	Resource Allocation	Start Time(ns)	Finish Time(ns)
T11	R3	0	15
T12	R3	15	25
T13	R2	15	23
T14	R1	25	33
T21	R3	0	15
T22	R3	15	28
T23	R1	28	36

(c)

Figure 4. Resource Contention in GCLP Sol.

TGFF (Task Graph For Free) [2].

It may be noted that we propose no new technique to build the task graph (DAG). Accordingly, any method, including the one proposed in [10], can be utilized for this purpose. The mapping problem discussed in this paper assumes that the task graph is provided as a part of the input specification.

2. Mapping and Scheduling application DAGs to NP Systems

This section describes randomized rounding approach followed by the methods handling resource contention among packets.

2.1 Randomized Rounding Technique

As the first step of solving the mapping and scheduling problem for application DAGs to NP systems, we set up the Integer Liner Program (ILP) formulation of the problem. The ILP formulation of the problem is provided in Appendix. As it is well known, the solution of ILP may take considerable amount of time, specially if the application DAG has large number of nodes. For this reason, we do not solve the ILP. We relax the integrality constraints on the variables, i.e allowing the solution to have fractional (non-integer) values for the variables, and solve the corresponding Linear Program (LP). Solution of the LP can be obtained much faster than the solution of the ILP as LP is solvable in polynomial time. Relaxing the integrality constraint may give rise to fractional (non-integral) solution. However, fractional values associated with task-to-resource mapping for variables may not have any physical meaning. In this section, we describe the randomized rounding approach [9] to convert the fractional values to integer values.

2.1.1 Rounding Scheme

One of the key issues in a randomized rounding approach is to maintain the feasibility of the solution as we round fractional variables. The selection of an appropriate rounding scheme and the adoption of some randomization strategies helps reduce the probability of constraint violations and speed up the randomization procedure. To maintain feasibility, one should carefully understand the interdependencies amongst the variables. For a constraint of the form

$$a + b + c + d \leq 1$$

where a, b, c and d are variables in the range [0, 1], we know that only one of them is to be rounded up to one, while the remaining variables should be zero. Thus, we consider variables in groups, and define an ordering for rounding those variables as explained in the following sections.

Among the five decision variables defined in appendix, x, y, z and d are $0/1$ variables and s is a positive integer variable. The variables d and s are related (constraint (4)) and they determine the execution order and scheduling time of each flow. Accordingly, they can be grouped and d is rounded first. The variables x, y and z are responsible for task-to-resource mapping and inter-task relationship. These variables can be placed in another group. From constraints (5) and (6), the variables y and z are determined by the variable x and if x is rounded, it determines the values of y and z as well. We chose to round variable d before variable x.

The fractional value of variables is used as the probability to round them to the closest integer. For example, a binary variable p has a fraction value 0.35 after some ILP relaxation. In order to determine the value of p (0 or 1), we generate a random number between 0 and 1. If the number happens to be greater than or equal to 0.35, p is set to 1 and 0 otherwise.

2.1.2 Variable Fixing

The relaxed version of the ILP (i.e., the LP) may still produce some variables with integer (or binary) values. During the next iteration of the LP, we assign integer values (obtained during the previous iteration) to these variables and treat them as constants. We call this process *variable fixing*. By preceding variable fixing with variable rounding, the number of variables to be rounded is reduced and thus time can be saved later in the rounding process. However, the variable-fixing process may not always lead to a situation where all variables end up having integer values. In that case, we still need to use *randomized rounding* to turn the left non-integer variables into integers. the same priority order is followed during variable fixing process as in the variable rounding process.

2.1.3 Rollback Point selection

Some of the constraints may not be satisfied even after randomized rounding of some variables. In this case, we need to roll back and undo the applied randomized rounding steps. If during the rounding process, rollback has to be done several times, it will significantly increase the execution time. Thus selection of a good rollback point is critical to the efficient performance of the algorithm. When we choose rollback point, we need to consider the context in which the constraint violation takes place. For example, as indicated earlier, there are two randomized rounding stages involved in solving the problem. The variables d's are rounded first and the variables x's are done next. If any constraint violation is detected during the first rounding stage, we remove all the rounded d from the problem file and restart the randomized rounding procedure. However, if any constraint violation is detected in the second stage, the rounded d values from the first stage can be preserved and we only require to roll back to the beginning of the second stage.

2.1.4 Rounding Step Size

The rounding step size is defined as the number of variables being rounded in each iteration. As the computation time is related to the number of iterations, lesser number of iterations will imply lesser computation time. However, the probability of some constraint being violated is also large when several variables are rounded simultaneously. Obviously, such violations imply increased execution time. In our experiments, we found that the computation time is essentially independent of the number of variables rounded in each iteration. We choose only one constraint at random and only one variable in that constraint is rounded at a time.

2.2 Resource Contention Problem

The total system cost in dollar value can be reduced further by relaxing the deadline for each packet and at the same time allowing multiple packets inside the system to satisfy the throughput constraint. Allowing multiple packets inside the system may introduce the problem of resource contention as already discussed in section 1, since more than one packet may access the same resource at the same instance of time. We have devised a technique to handle resource contention amongst tasks belonging to same application.

2.2.1 Exploration of Solution Space

The solution space of mapping and scheduling problem for NPSs can be described on a one-dimensional range as shown in Figure 5. If the deadline constraint is too strict, the ILP model may not return a feasible solution. On the other hand, making the deadline too relaxed will give a feasible solution with a lower cost as the ILP tries to allocate more than one task to a resource. This increases the chance of resource contention amongst packets. The cost generally increases on decreasing the deadline value, as the ILP has to include faster (more expensive) resources. The goal is to find a point in the solution space, which is feasible and resource contention-free with lowest possible dollar cost.

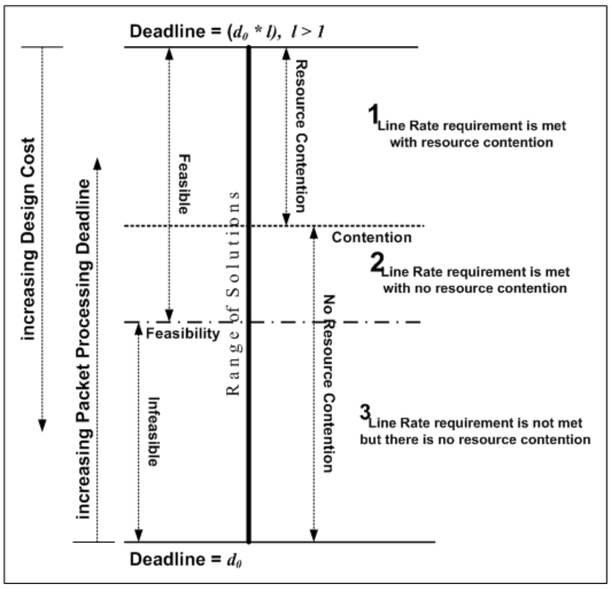

Deadline = (d_0 * l), l > 1
increasing Design Cost
increasing Packet Processing Deadline
Feasible
Range of Solutions
Resource Contention
1 Line Rate requirement is met with resource contention
Contention
2 Line Rate requirement is met with no resource contention
Feasibility
No Resource Contention
Infeasible
3 Line Rate requirement is not met but there is no resource contention
Deadline = d_0

Figure 5. Range of Solutions

2.2.2 Resource Contention Detection

On account of the existence of multiple packets inside the system, resource contention may be caused amongst the packets following the same path or different paths in a task graph. To describe the method for resource contention detection, the following terms are defined first:

Packet Flow Graph (PFG): It is defined as $G(V, E)$, where V is the subset of resources allocated by the ILP, with additional entry s and exit t nodes. An edge $e = (u, v) \in E$ indicates an allocation sequence between resource u and v. There is an weight function $w(e)$ associated with each edge. For each $w(e) = (x(e), y(e))$; $x(e)$ indicates the sequence number of allocation of the following resource and $y(e)$ indicates the corresponding execution time on the resource.

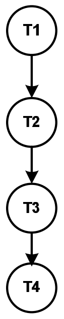
Resource Cycle Time: It indicates the maximum amount of timespan for which a resource is busy in executing a set of tasks for a packet, i.e. a resource is not available until it finishes all the tasks scheduled for a packet on it. For each resource $r \in V - \{s, t\}$ the resource cycle time is defined as:

$$T_{c_r} = \sum_{x(e)=a}^{x(e)=b} y(e) \qquad (1)$$

where $a = \min_{e \in \delta^-(r)}(x(e))$, and $b = \max_{e \in \delta^-(r)}(x(e))$, and $\delta^-(r)$ is the set of incoming edges to node r.
Maximum Cycle Time: It is defined as the maximum of all the resource cycle times, i.e.,

$$T_c = \max_{r \in V - \{s,t\}} \{T_{c_r}\}. \qquad (2)$$

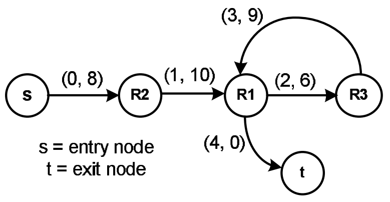
Packet flow graph construction from a given solution of ILP and the calculation of maximum cycle time is shown in Figure 6. Figure 6(a) shows a particular task graph and 6(b) has a corresponding solution given by the ILP. Figure 6(c) shows the corresponding PFG. The calculation of maximum cycle time is shown in 6(d).



(a)

Mapping from ILP	
Task – Resource	Execution Time
T1 – R2	8 ns
T2 – R1	10 ns
T3 – R3	6 ns
T4 – R1	9 ns

(b)



(c)

$r \in V - \{s, t\}$	$a=\min_{e \in \delta^-(r)}(x(e))$	$b=\max_{e \in \delta^-(r)}(x(e))$	T_{cr}
R2	0	0	8
R1	1	3	10 + 6 + 9 = 25
R3	2	2	6

(d) $T_c = \max(8, 25, 6) = 25$ ns

Figure 6. PFG Construction

In order to detect the resource contention in the allocation and scheduling corresponding to the ILP solution, the packet-flow graph is built first. Since the maximum cycle time in the PFG is a measure of the maximum timespan a resource is busy processing a packet, there must exist at least one resource contention amongst packets in the system if the maximum cycle time is greater than the packet arrival rate. However, this technique may fail to detect an existing resource contention if two consecutive packets follow different branches in the same task graph. In order to detect the resource contention in this scenario, Gantt chart method is used. Resource allocation time-lines are drawn for each path. Any overlapping allocation of the same resource for more than one packet is considered to be a resource contention. An iterative approach to resource contention detection and elimination is described next.

2.2.3 Resource-Contention Elimination

To speed up the exploration of solution space, an iterative procedure is used. It finds the least cost feasible solution which is free from resource contention. A flowchart representation of the iterative improvement process is shown in figure 7.

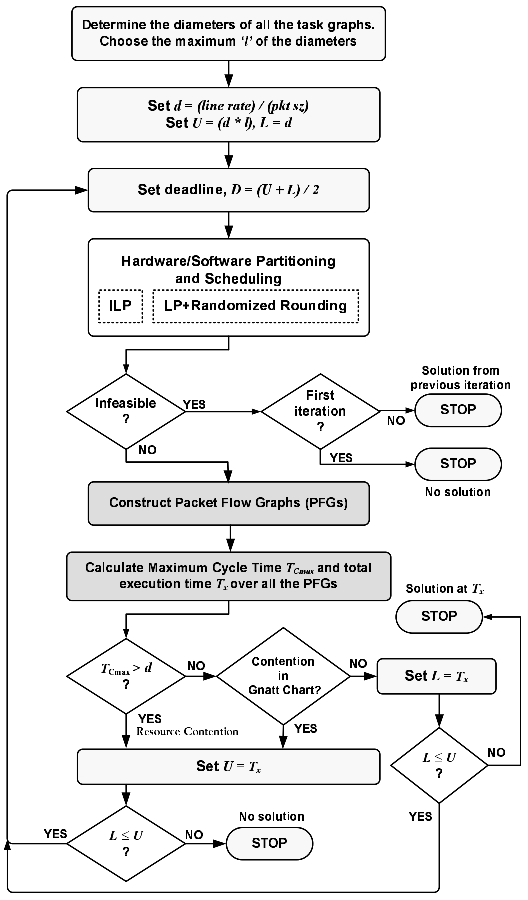

Determine the diameters of all the task graphs. Choose the maximum 'l' of the diameters
Set d = (line rate) / (pkt sz)
Set U = (d * l), L = d
Set deadline, D = (U + L) / 2
Hardware/Software Partitioning and Scheduling
ILP
LP+Randomized Rounding
Infeasible ?
YES
NO
First iteration ?
NO
YES
Solution from previous iteration
STOP
STOP
No solution
Construct Packet Flow Graphs (PFGs)
Calculate Maximum Cycle Time T_Cmax and total execution time T_x over all the PFGs
Solution at T_x
STOP
T_Cmax > d ?
NO
Contention in Gnatt Chart?
NO
Set L = T_x
YES
Resource Contention
YES
Set U = T_x
L ≤ U ?
NO
YES
YES
L ≤ U ?
NO
No solution
STOP

Figure 7. Iterative Improvement

3. Experimental Results

We discuss the quality of solution provided by the randomized rounding approach compared to the ILP solution. Later we present a case study for design of a Network Processing system that will support a designated set of applications.

3.1 Performance of Randomized Rounding

The quality of randomized mapping can be evaluated based on the solution time and the objective value. To test the effectiveness of our technique, we use the task graphs shown in figure 1 as the problem input which has 737 variables and 868 constraints in the ILP formulation. We vary the timing deadline and conduct ten independent experiments for each deadline. Figure 8 is the solution time comparison between the randomized rounding (RR) approach and ILP technique. Figure 9 reflects the deviation of heuristic solution from the optimal objective value obtained by ILP. The figures show that our approach produces a solution with near optimal objective value in a fraction of the time needed to find an optimal solution.

3.2 Task-to-Resource Mapping Case Study

To demonstrate the validity of the model, we applied our mapping method to design an IXP2400-*like* system. We designed the system for 3 common and representative applications for NPSs defined by the Network Processor Forum benchmarking implementation agreements [3].

The applications are shown in figure 10: Ingress and egress processing for the Ethernet IPv4 unicast forwarder and Diffserv on the ingress path. IPv4 packet forwarding is based on RFC1812 is the core in many NPS applications. Diffserv is a method of facilitating end-to-end quality of service (QoS) over an existing IP network.

In all three applications, the tasks to be used were decided based on the Intel Application Building Blocks Design Guide [4]. A resource set was chosen to be the set of resources in IXP2400. This helped us have cycle-accurate simulation model available for all the resources. The resource set with associated design parameters is shown in table 1.

Each possible implementation for a task was profiled in the IXP2400 cycle accurate simulation environment. To obtain average execution cycles per task, the application was tested with worst case input traffic. An instance of each implementation of a task was run on the hardware and software resource options with the appropriate traffic. The line rate was set to 2.5Gbps and packet size was set to minimum 64 bytes (size of mpacket for IXP2400) to simulate the worst case. The Intel Developer Workbench was used with its cycle-accurate simulator and transactor to get the

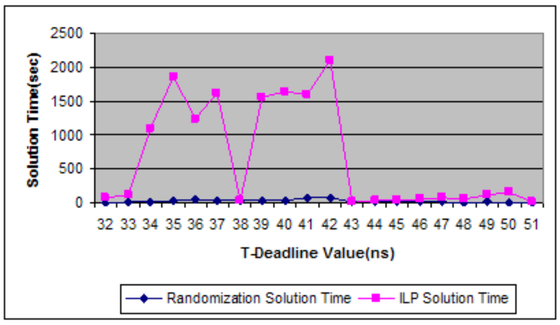


Figure 8. Solution Time Comparison

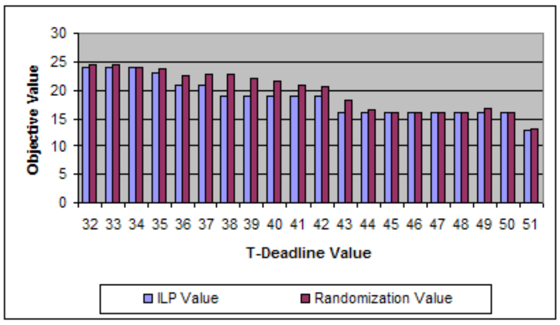


Figure 9. Objective Value Deviation

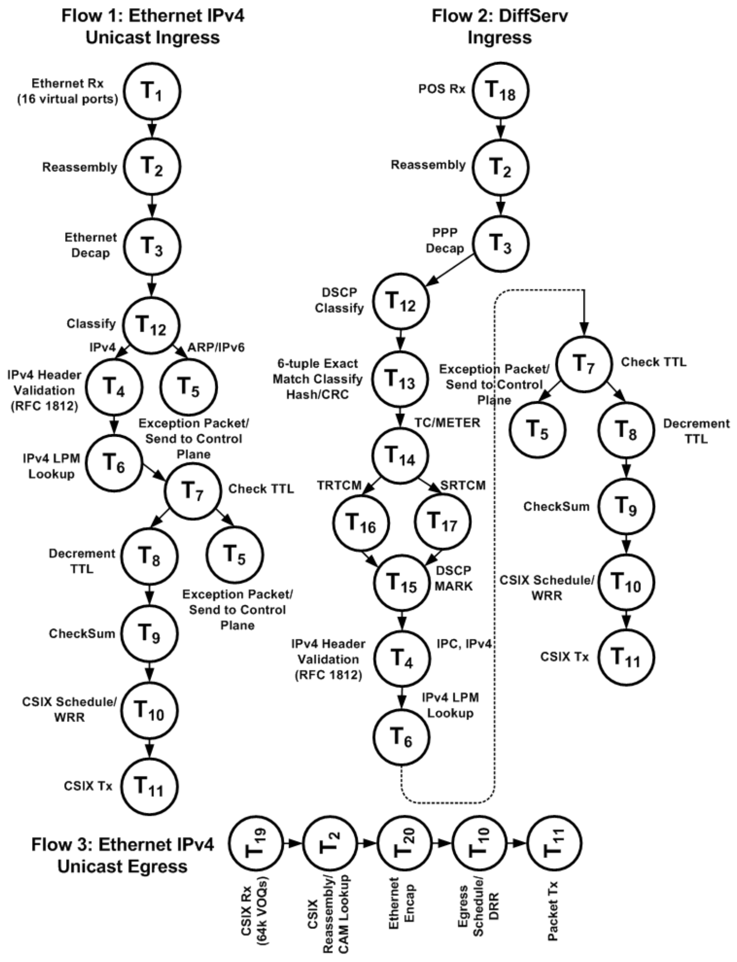


Figure 10. Task Graphs based on NPF Benchmark Applications for NPSs

Resource	Label	Cost	Area
RISC Processors	$R_1..R_8$	7	6
Hash Unit	R_9	11	3
CRC Unit	R_{10}	9	2
CAM Accelerator	R_{11}	17	4
Route Lookup Engine	R_{12}	13	4
Checksum Unit	R_{13}	12	2
POS PHY	R_{14}	9	3
CSIX	R_{15}	7	3

Table 1. Set of Resources

timing values. Communication delays were obtained based on the bus speeds and the amount of data being transferred between tasks. Communication delays are not shown here.

ILOG CPLEX 8.1 was used to solve the constraint system of the ILP formulation on an Intel XEON 1.5GHZ with 1GB RDRAM memory running RedHat Linux 8.0. The Intel IXP2400 workbench was configured for 600MHz microengine speed.

For the task graph in figure 10 with *22 tasks, 16 resources, and 3 applications*, the ILP generated 202 variables and 297 constraints. Packets were set to arrive every 205ns with no inter-packet gap. The value 205ns corresponds to an OC-48 configuration with packet size 64 bytes. The maximum length of a path in the task graph specification was 14, thus the initial relaxed deadline was set to (14 * 205 =) 2870ns.

The solution was found in **6 seconds** after 7 iterations of binary search for a feasible solution. The deadline was relaxed from *51ns* to *556ns*. The total cost of hardware was 97 units. The mapping and schedule generated is shown in table 2.

		Start times		
Task	**Resource**	**IPv4 Ingress**	**Diffserv Ingress**	**IPv4 Egress**
T_1	R_{14}	0	–	–
T_2	R_1	15	10	10
T_3	R_7	93	88	–
T_4	R_6	146	298	–
T_5	R_7	146	392	–
T_6	R_{12}	189	341	–
T_7	R_7	238	390	–
T_8	R_6	240	392	–
T_9	R_{13}	242	394	–
T_{10}	R_{15}	434	–	434
T_{11}	R_{15}	468	468	468
T_{12}	R_7	123	118	–
T_{13}	R_9	–	141	–
T_{14}	R_{11}	–	201	–
T_{15}	R_6	–	270	–
T_{16}	R_3	–	215	–
T_{17}	R_6	–	230	–
T_{18}	R_{14}	–	0	–
T_{19}	R_7	240	–	–
T_{20}	R_{15}	–	–	0
T_{21}	R_6	–	–	414

Table 2. Output Mapping

We compared our mapping against the task-resource mapping provided by Intel in the preconfigured applications of [4]. Aggregate throughput, end-to-end packet latency and resource utilization were the parameters used to compare the models. We found our results to be within 7-10% of the Intel mapping. Our method used less resources with better resource utilization and achieved close to desired throughput figures.

The important aspect of the mapping produced by our approach is the cost per performance. Intel's mapping costs 134 for the same performance achieved with our mapping with cost 97. Moreover, our mapping method generated the mapping in less than 7 seconds while Intel's hand-tuned mapping must have taken days to arrive at.

Figures 11 through 14 give the comparison of the performance of the two mappings.

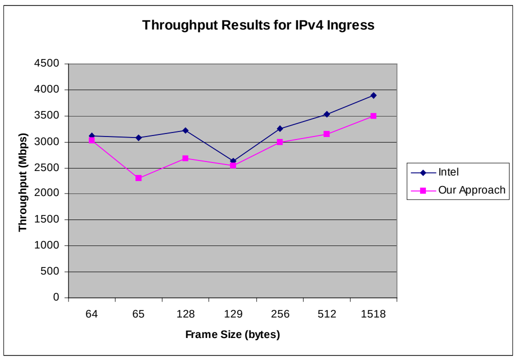


Figure 11. IPv4 Ingress Throughput

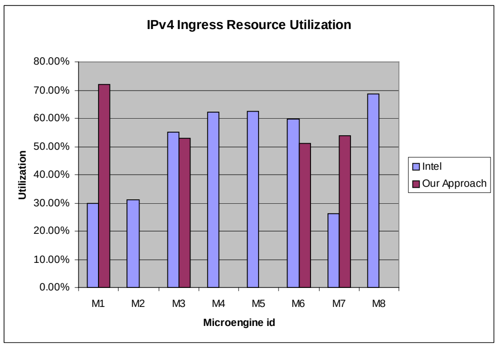


Figure 12. IPv4 Ingress Resource Utilization

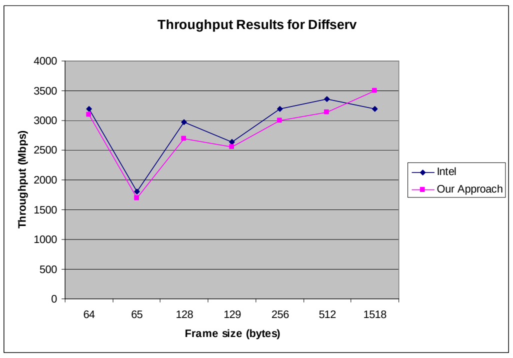


Figure 13. Diffserv Ingress Throughput

4. Conclusion

In this paper we have presented a new methodology for task to resource mapping problem. We show that our methodology produces near optimal solution in a fraction of time needed to find the optimal solution. In addition, we have identified the resource contention problem that can arise in case multiple packets are allowed to be within the

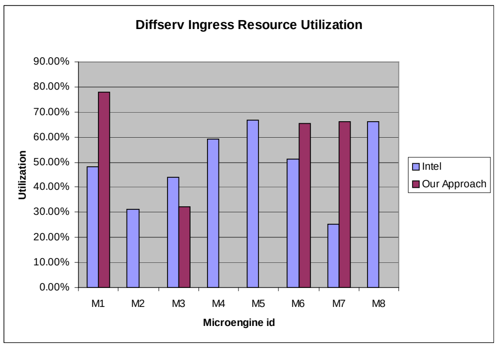


Figure 14. Diffserv Ingress Resource Utilizn

system at the same time. We have provided two different solution techniques for this resource contention problem. Finally, we provided a case study of Network Processor system design using our tool.

5. REFERENCES

[1] K. Chatha and R. Vemuri. Hardware-software partitioning and pipelined scheduling of transformative applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10:193–208, 2002.

[2] R. Dick, D. Rhodes, and W. Wolf. Tgff: task graphs for free. *International Workshop Hardware/Software Codesign*, pages 97–101, Mar 1998.

[3] N. P. Forum. Benchmark implementation agreements. *Network Processing Forum, http://www.npforum.org/techinfo/IA.*

[4] Intel. Intel application building blocks design guide. *http://www.intel.com/design/network/-products/npfamily/sdk.htm.*

[5] A. Kalavade and P. A. Subrahmanyam. Hardware/software partitioning for multi-function systems. *IEEE Transactions on CAD of ICs and Systems*, 17(9):516–521, Sep 1998.

[6] R. Niemann and P. Marwedel. Hardware/software partitioning using integer programming. *Electronic Design & Test Conference*, pages 473–479, 1996.

[7] M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. *Tenth International Symposium on Hardware/Software Codesign*, pages 67–72, May 2002.

[8] A. Prasad, W. Qui, and R. Mahapatra. Hardware software partitioning of multifunction systems. *Design Automation for Embedded Systems*, Dec 2002.

[9] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proof. *Combinatorica*, 7:365–374, 1987.

[10] R. Ramaswamy, N. Neng, and T. Wolf. Application analysis and resource mapping for heterogeneous network processor architectures. *Proc. of Third Workshop on Network Processors and Applications (NP-3) in conjunction with Tenth International Symposium on High Performance Computer Architecture (HPCA-10)*, pages 103–119, Feb 2004.

[11] L. Thiele, S. Chakraborty, M. Gries, and S. Kunzli. Design space exploration of network processor architectures. *First Workshop on Network Processors at the 8th International Symposium on High-Performance Computer Architecture (HPCA8)*, pages 30–41, 2002.

[12] H. Vin, J. Mudigonda, J. Jason, E. Johnson, R. Ju, A. Kunze, and R. Lian. A programming environment for packet-processing systems: Design considerations. *In the Workshop on Network Processors & Applications - NP3 in conjunction with The 10th International Symposium on High-Performance Computer Architecture*, Feb 2004.

APPENDIX (ILP Formulation)

G_i is the number of candidate algorithms for task i. $T^k_{i,j}$ is the running time for task i with algorithm j on resource k. Each resource k is associated with an area A_k, whereas A is the maximum allowable area for the system. $C^{k,k',f}_{i,i'}$ represents the communication time, if task i (in flow f) is assigned to resource k and task i' (in flow f) is assigned to resource k' and task i' follows task i in the task graph associated with flow f. Each flow f is associated with a deadline T and all the tasks in a flow must be completed with the deadline.

The ILP model has the following *decision variables*: $x^{k,f}_{i,j}$ is equal to 1, when task i runs on resource k with algorithm j in flow, 0 otherwise y_k is equal to 1, when resource k is used in the target architecture, 0 otherwise $d^f_{u,v}$ is equal to 1, when task u starts no later than task v in flow f, 0 otherwise $z^{k,k',f}_{u,v}$ is equal to 1, when in flow f, task u runs on resource k and task v runs on resource k', 0 otherwise s^f_i is the starting time of task i in flow f

The *objective* of this ILP model is to find a task-to-resource mapping with minimum cost:

$$\text{Minimize} \quad \sum_{k=1}^{N} C_k y_k$$

The ILP model enforces the following *constraints*:
Area constraint: The total area occupied by the resources in the target architecture should not exceed the maximum allowed area,i.e.

$$\sum_{k=1}^{N} A_k y_k \leq A \tag{3}$$

Timing constraint: In each flow, every task should finish before the timing deadline, i.e.

$$\forall f, \forall i, \; s^f_i + \sum_{k=1}^{N} \sum_{j=1}^{G_i} T^k_{i,j} x^{k,f}_{i,j} \leq T \tag{4}$$

Unique task constraint: In each flow, every task runs exactly once, i.e.

$$\forall f, \forall i, \; \sum_{k=1}^{N} \sum_{j=1}^{G_i} x^{k,f}_{i,j} = 1 \tag{5}$$

Exclusive resource constraint: In each flow, if two tasks occupy the same resource, they have to be scheduled sequentially, i.e.,

$$\begin{aligned}\forall f, \forall k, \forall u\&v,\ & s_u^f + \sum_{j=1}^{G_u} T_{u,j}^k x_{u,j}^{k,f} + \sum_{\forall i, u\to i} \sum_{k'\neq k}^{N} C_{u,i}^{k,k',f} z_{u,i}^{k,k',f} \\ & -s_v^f \leq (3 - d_{u,v}^f - \sum_{j=1}^{G_u} x_{u,j}^{k,f} - \sum_{j=1}^{G_v} x_{v,j}^{k,f})T\end{aligned} \tag{6}$$

Communication delay constraint: In each flow, a communication delay may be defined when two adjacent tasks use different resources.

$$\begin{aligned}\forall f, \forall k\&k', \forall u\&v, \quad z_{u,v}^{k,k',f} &\geq \sum_{j=1}^{G_u} x_{u,j}^{k,f} + \sum_{j=1}^{G_v} x_{v,j}^{k',f} - 1 \\ z_{u,v}^{k,k',f} &\leq \sum_{j=1}^{G_u} x_{u,j}^{k,f} \\ z_{u,v}^{k,k',f} &\leq \sum_{j=1}^{G_v} x_{v,j}^{k',f}\end{aligned} \tag{7}$$

Task-to-resource mapping constraint: A resource is included in the target architecture if and only if it is used by at least one task, i.e.

$$\forall f, \forall i, \forall k, \quad \sum_{j=1}^{G_i} x_{i,j}^{k,f} \leq y_k \tag{8}$$

Task dependency constraint: If two tasks (in the same path)have to be scheduled sequentially, the task starting earlier should finish before another one begins. For two adjacent tasks using different resources, the communication delay should also be included in the total running time of the earlier one, i.e.,

$$\begin{aligned}\forall f, \forall u\&v, \quad & s_u^f + \sum_{k=1}^{N} \left(\sum_{j=1}^{G_u} T_{u,j}^k x_{u,j}^{k,f} + \sum_{u, u\to i} \sum_{k'\neq k}^{N} C_{u,i}^{k,k',f} z_{u,i}^{k,k',f} \right) \\ & \leq s_v^f\end{aligned} \tag{9}$$